Cognitive Principles to Support Information Requirements Agility

Jeffrey Parsons[1] and Yair Wand[2]

[1] Faculty of Business Administration,
Memorial University of Newfoundland St. John's, Canada
[2] Sauder School of Business,
The University of British Columbia Vancouver, Canada
jeffreyp@mun.ca, yair.wand@ubc.ca

Abstract. Despite the growing interest in agile information systems development approaches, we contend that existing approaches retain traditional assumptions about the structure of information, assumptions that inhibit agile responses to emerging and evolving information requirements. We suggest an approach, based on cognitive principles, to model information requirements by separating conceptual views of data from logical models, allowing the former to be changed without requiring changes to the latter.

1 Introduction

Traditional approaches to information systems engineering were based on some version of the waterfall model, in which development is undertaken in well-defined stages, with each being completed before the next stage started. Such a process entails developing a comprehensive set of system requirements early in the process. This approach assumes the stakeholders of an information system are a well-defined group, whose functional requirements are known and can be elicited systematically [1, 2].

In addition, these approaches make a number of (usually implicit) assumptions about data requirements, including: (1) the sources of the data are well-understood; (2) there is some centralized management of data (a global conceptual schema), possibly shared by several applications; and (3) the information structure (conceptual and logical design) is relatively stable over time.

In view of the limitations of the waterfall model, the focus in information systems engineering has shifted to approaches that emphasize modularity and agility. In particular, requirements emerge and are refined through several iterations, during which additional functionality is added incrementally based on interaction with users and other stakeholders and the availability of a prototype or partially implemented system.

Notwithstanding these developments, we contend that agile development approaches retain implicit assumptions about the structure and stability of information requirements characteristic of waterfall methods. Specifically, although agile methods recognize that functionality and user interfaces might change often across iterations, the structure of information requirements (and therefore, of data in the

X. Franch and P. Soffer (Eds.): CAiSE 2013 Workshops, LNBIP 148, pp. 192–197, 2013.

database) is assumed to be relatively stable. There are two reasons why the data aspects of applications might be less volatile than the processing and interface aspects. First, data reflect a view of the application domain in terms of a set of concepts and relationships among them. While the required functionality might change relatively easily, it is usually implicitly assumed that stakeholders' views of the domain are quite stable over time. Second, functionality might often be changed or added without a need to "retrofit" the developed system, whereas changes in data structures might force changes to the implementation of many functional components. It follows that, during the development process, one should not expect much agility with respect to the structure of the information requirements for the system.

We claim, however, that modern application environments do require more flexibility and agility to accommodate changes to the information itself. The reasons for such changes include: (1) the emergence of new (not originally anticipated, and often external) sources of data (e.g. on the Internet), the structure of which might not match the conceptual model elicited during initial requirements determination; (2) the appearance of new users or stakeholders who might have different domain views that were not captured during initial requirements determination; and (3) the discovery of new information requirements that were not anticipated earlier. Such changes would typically call for changes in data structures and consequently the functionality already implemented might need to be re-implemented.

Information requirements are manifested via the structure of the data in the application. Databases were introduced in order to decouple issues of access to data from the physical structures used to store data, making it possible to change physical data structures without affecting the implemented functionality or requiring changes to the mechanisms used to store and retrieve data. However, this independence does not extend to changes to logical or conceptual data structures. Specifically, logical data structures often are designed to reflect specific views (as embedded in conceptual data models). Thus, when new or different user views emerge, data models need to be changed accordingly, and the functionality that has been already implemented based on these structures might need to be changed as well.

In this position paper, we suggest an approach, based on cognitive principles, that enables flexibility in supporting different views of a domain, without having to re-implement existing functionalities. If this can be done, then agility can be extended to changes in users' views of a domain. Thus, an earlier commitment to a specific user view will not constrain the possible changes that can be accommodated in later development cycles. As user views are manifested in the form of information requirements, we believe our approach will result in agility with respect to these requirements.

2 An Example

We begin with an example demonstrating how information requirements changes may cause changes of data structure (which in turn might affect processing elements of a system). We first present a specific view and the related data structure design. We then discuss changes to the view arising from changing application requirements, and

how these might impact the database design and accordingly query implementation. Later we revisit the example to show how such view changes can be accommodated in our proposed approach without impacting the implementation of functionality.

Consider a university information system that manages student data. A graduate school is responsible for its thesis-based graduate students. It manages personal student information. In addition, each student has a supervisor, and the system needs to keep track of the assignment of students to supervisors.

In this highly simplified system, the information requirements can be captured in a relational data model, consisting of a single table:

STUDENT(Student_ID, Name, Address, Supervisor).

Note that, if it is necessary to keep track of supervisor information, this can be extended by adding a table with supervisor information. Suppose further that a basic reporting requirement is to list the name of students from a particular location, X, and their supervisors. This can be satisfied with the following SQL query:

```
select Name, Address, Supervisor
  from student where Address=X
```

Now suppose new requirements arise from the introduction of a course-based graduate program (in addition to the existing thesis-based program). It is necessary to maintain the same personal information about students in this new program. However, these students do not have a supervisor; instead, they participate in a designated major. The existing logical data model does not accommodate this new reality very well. It is possible to extend the definition of the STUDENT relation by adding a `major` attribute and allowing null values for both 'major' and 'supervisor'. However, given the resulting ambiguity in interpreting the meaning of null ('not applicable' versus 'unknown'), a more appropriate solution is to create two new relations capturing the information unique to each specific subclass of student (thesis and course-based)[1]:

STUDENT(Student_ID, Name, Address)
THESISSTUDENT(Student_ID, Supervisor)
COURSESTUDENT(Student_ID, Major)

Given this change in the logical data model resulting from the evolving requirements (manifested by a new view of the kinds of entities about which information must be maintained), the existing query to extract student information and supervisor names for students with a particular address will no longer work. Instead, it will be necessary to join the STUDENT and THESISSTUDENT tables to answer this query. Thus, any functionality relying on the original query will not be supported until the query is rewritten. Moreover, it will be necessary to repopulate the tables in the new design based on the changes made. In this case, both the original database structure and the queries can be considered ***brittle***, as they might not work (without modifications) when the requirements of the application domain change. Note also that the queries cannot be written or adjusted to the new requirements without detailed knowledge of the specific schema chosen.

[1] It is possible to create only two tables for the new requirements and maintain the names and addresses of students in the tables for the particular type of student (thesis vs. course-based). In that case, it will be more complicated to query information common to all students.

To address this brittleness, we introduce cognitive principles that can guide the design of flexible data structures to naturally accommodate changing views of the domain without requiring changes to the queries that support information requirements.

3 Cognitive Principles

We aim to identify principles that will enable changes to the conceptual views of the application domain without requiring changes to the underlying data structures. To identify such principles we turn to cognitive science, specifically, to ***classification theory*** [8,9], which deals with how humans form concepts based on their experiences. In particular, we focus on two cognitive principles of classification: (1) instances possess properties and exist independent of how they are classified; (2) classes do not exist independent of human cognition, but instead are abstractions of useful similarities formed based on observed properties of instances. Different abstractions are formed to reflect different views (reflecting different uses).

We suggest these principles be applied to information systems design by using the following design rule [4]:

Design Rule 1: Instances in an information system should be stored with their properties, independent of any specific classification.

To understand the significance of this rule, note that the common approach places data in pre-determined "containers", which usually reflect a given view (i.e., a particular set of classes). The most common type implementation of a container is the relation in the relational model. The proposed rule actually means that database design will not be driven by (or reflect) any particular user view, except that a view may indicate the relevant properties of instances.

The above design rule does not deny the importance of classes in information systems engineering. Classes still have a vital role in requirements analysis, system design, and data access. Rather, it considers the issue of which classes are relevant to information requirements as completely separate from the mechanisms to store data. Hence, we term this as applied to design ***class independence***.

As user views as well as accessing and entering information will still be defined in terms of concepts or classes, we add a second design rule:

Design Rule 2: Mechanisms must be provided to define classes in terms of properties and to retrieve or access instances of a given class.

Given that, in our approach the database stores only instances. Retrieving all instances of a class means identifying all instances possessing a given set of properties.

These design rules imply that classes can be defined and modified, and data accessed in terms of classes, without a need to change the way instances are stored. Classes thus become completely abstract, and do not serve as "containers" for data.

Consider now how class independence can support flexibility in domain views. This requires first that instances be related to classes. Such relationships reflect how classes are defined. The simplest way to do this is the classic approach where a class is defined in terms of a set of properties. Any instance possessing this set will be considered an instance of the class. A class-independent implementation will require operations to support identifying the instances of a class and the classes to which an instance belongs.

4 Implementing the Cognitive Principles to Support Information Requirements Agility

Consider again the student example (Section 2). We can envision a domain of instances possessing properties as the foundation for the logical design of data. There are various ways to implement this idea. For consistency with the example above, we assume a relational database implementation.[2] One way to realize the instance-based approach is to have a separate (binary) table representing each property of interest. In our example, given the initial information requirements, we could have three tables:

NAME(Student_ID, Name)
ADDRESS(Student_ID, Address)
SUPERVISOR(Student_ID, Supervisor).

As explained above, in the instance-based we need a mechanism to define classes in terms of properties. For example the class STUDENT can be defined as:

STUDENT = {Name, Address, Supervisor}.[3]

For the emerging requirement to accommodate students in course-based programs, all that is needed is to add a new property for Major to each relevant instance (in our implementation example this will entail adding a binary table where each row represents an instance possessing the new property). In addition, a class structure to reflect the new view can be defined ***without making any changes to the underlying data***:

THESISSTUDENT = {Name, Address, Supervisor}
COURSESTUDENT = {Name, Address, Major}.

This structure is purely a view over the attributes. The base (binary) tables do not change as new classes are defined, existing classes are modified (their properties change), or classes are eliminated. For example, if a list of all students (both thesis and course-based) is needed, the class STUDENT = {Name, Address} can be added without any change to data structures or to other classes.

It can be readily shown that under this structure, any previously defined queries (such as to retrieve the names, addresses and supervisors of students having a particular address) will continue to work.

From this example, we see that a logical data model based on cognitive principles of classification offers considerable flexibility to accommodate changes in information requirements and to views of the domain. In contrast, models such as the relational model that embed a certain (class-based) view of the domain in the logical data model are cumbersome, requiring changes at the design level to accommodate changing requirements and/or views of the domain.

---

[2] Note that other ways of realizing this design are possible. Graph-based structures provide a natural mechanism to support domain conceptualization in terms of instances and properties. Similarly, column-oriented and key-value structures are similar to our binary table approach.

[3] Technically, STUDENT can be defined as a view through a query joining the Name, Address, and Supervisor tables on the Student_ID attribute. The identity of the student does not appear explicitly, as it is not part of a conceptual class, but rather an implementation mechanism. Such identity will exist however in the underlying data structures (e.g. relational).

5 Conclusions and Further Opportunities

Requirements in information systems engineering clearly fall within the domain of cognition. Developing an information system often begins without a full understanding of stakeholder requirements. Moreover, in modern technologies and applications, requirements cannot be expected to remain stable over time. The uncertainty and volatility surrounding requirements has been recognized in software engineering, leading to the rise of agile approaches.

However, one aspect of development has been largely ignored in the pursuit of agility – the nature of ***information*** requirements and their manifestation in database design. We contend that this is fundamentally a cognitive issue, as information requirements reflect stakeholder conceptualizations of the important phenomena of interest in the domain. We argue that the role of cognition, beyond being closely linked to users views and hence requirements, can be extended to data design principles that support changing information requirements that are now commonplace.

Here we have not addressed the actual process of eliciting users' views. Elsewhere we describe how additional cognitive principles can be used to determine user views [3,6], to evaluate alternative views [6], to guide conceptual data design [3], to integrate different views [Parsons & Wand 2003], and as underlying foundations for general information processing architecture [7].

References

1. Browne, G.J., Rogich, M.B.: An Empirical Investigation of User Requirements Elicitation: Comparing the Effectiveness of Prompting Techniques. Journal of Management Information Systems 17(4), 223–249 (2001)
2. Hickey, A.M., Davis, A.M.: A Unified Model of Requirements Elicitation. Journal of Management Information Systems 20(4), 65–84 (2004)
3. Parsons, J., Wand, Y.: Choosing Classes in Conceptual Modeling. Communications of the ACM 40(6), 63–69 (1997)
4. Parsons, J., Wand, Y.: Emancipating Instances from the Tyranny of Classes in Information Modelling. ACM Transactions on Database Systems 5(2), 228–268 (2000)
5. Parsons, J., Wand, Y.: Attribute-Based Semantic Reconciliation of Multiple Data Sources. Journal on Data Semantics 1(1), 21–47 (2003)
6. Parsons, J., Wand, Y.: Using Cognitive Principles to Guide Classification in Information Systems Modeling. MIS Quarterly 32(4), 839–868 (2008)
7. Parsons, J., Wand, Y.: Extending Classification Principles from Information Modeling to Other Disciplines. Journal of the Association for Information Systems 14(3) (2013)
8. Rosch, E.: Principles of Categorization. In: Rosch, E., Lloyd, B. (eds.) Cognition and Categorization, pp. 27–48. Erlbaum, Hillsdale (1978)
9. Smith, E.E., Medin, D.L.: Categories and Concepts. Harvard University Press (1981)