Information Systems 47 (2015) 33–50

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

Conceptual independence: A design principle for the construction of adaptive information systems

Simon McGinnes*, Evangelos Kapros

School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, Ireland

ARTICLE INFO

Article history:
Received 13 April 2011
Received in revised form
20 August 2013
Accepted 8 June 2014
Recommended by: P. Loucopoulos
Available online 20 June 2014

Keywords:
Conceptual modelling
Conceptual independence
Data independence
Schema evolution
Software design
Adaptive information systems

ABSTRACT

This paper examines the problem of conceptual dependence, the coupling of software applications' internal structures and logic with their underlying conceptual models. Although conceptual dependence is almost universal in information system design, it produces a range of unintended negative consequences including system inflexibility and increased maintenance costs. Many information systems contain components, such as database tables and classes, whose design reflects the entity types and relationships in underlying, domain-oriented conceptual models. When the models change, work is involved in altering the software components. For example, an e-commerce system might include tables and classes representing product types, customers and orders, with associated code in methods, stored procedures and other scripts. The structure of the entity types and their relationships will be implicit in the tables, classes and code, coupling the system to its conceptual model. Any change to the model (such as the introduction of a new entity type, representing order lines) invalidates existing structures and code, causing rework. In large systems, this rework can be time-consuming and expensive. Research shows that schema change is common, and that it contributes significantly to the high cost of software maintenance. We argue that much of the cost may be avoidable if alternative design strategies are used. The paper describes an alternative design approach based on the principle of conceptual independence, which can be used to produce adaptive information systems (AIS). It decouples the internal structures and logic of information systems from the domain-specific entity types and relationships in the conceptual models they implement. An architecture for AIS is presented which includes soft schemas (conceptual models stored as data), an end-user conceptual modelling tool, a set of archetypal categories (predefined semantic categories), and an adaptive data model which allows data to be stored without conceptual dependence. The archetypal categories allow domain-specific run time behaviour to be provided, despite the absence of domain-specific software structure and logic. An advantage of AIS over conventionally-designed applications is that each AIS can be used in a wide variety of domains. AIS offer the prospect of significantly reduced maintenance costs, as well as increased scope for the development and modification of systems by end users. Work to date on implementation of the AIS architecture is discussed, and an agenda for future research is outlined including development and evaluation of a fully-featured AIS. The paper discusses challenges to be overcome and barriers to adoption.

* Corresponding author. Tel.: +353 1 896 2092.
E-mail address: simon.mcginnes@tcd.ie (S. McGinnes).

http://dx.doi.org/10.1016/j.is.2014.06.001
0306-4379/

1. Introduction

1.1. Meeting the growing demand for information systems

End-user expectations and business change continue to drive growing demand for information systems. As a result, more systems are being produced, by more people, than ever before [1]. This is occurring in the context of rapid technological evolution, which brings new ways of operating; current interest in technology procurement is focused around the hot topics of "analytics, big data, mobility, collaboration, and cloud" [2]. Information technology professionals face a significant challenge in trying to satisfy customer demand for new system functionality—whilst also meeting the perennial but essential requirements of security, reliability, usability, maintainability, and cost-effectiveness.

Fundamentally, information systems requirements can be satisfied in only two ways: (a) by using prewritten software such as commercial off-the-shelf (COTS) products or cloud services, and (b) by developing new systems. Many organisations use both methods; enterprise software can meet a proportion of needs, and more idiosyncratic requirements must be met through custom development.

Both methods have shortcomings, however. Prewritten software is inflexible, and it can be difficult to source products or services to meet the organisation's particular requirements. Integrating externally-sourced software with in-house systems is often complex and time-consuming [3]. On the other hand, custom development is typically slow, labour-intensive and risky, and it demands expert skills which the organisation may neither possess nor wish to acquire [4]. These factors can easily lead to delay and poor quality [5].

Regardless of how information systems are acquired, they must be maintained. Maintenance is necessary for many reasons: errors must be corrected, technological evolution accommodated, new needs met. Over time, the churn of maintenance tends to reduce system reliability and maintainability. Even well-designed systems become unmaintainable and must be replaced; this effect has been termed "software entropy" [6,7].

Maintenance is a growing cost for organisations. Information systems change more *after* implementation than during development [6]; maintenance accounts for most (e.g. 75–90%) of the lifetime cost of information systems [8]. It is estimated that, by 2015, the USA will employ 3.5 million IT professionals in software maintenance, an increase of over 300% since 1995. In contrast, the population of developers has remained static at only one-third of that level during the same period [9].

This paper focuses on one factor which contributes to the high cost of maintenance: schema evolution [10]. Database schemas and other parts of information systems are often designed around conceptual models, which depict the entity types and relationships in the application domain of interest. When these models need to be altered, maintenance activity is necessary to modify affected software components. The work is time-consuming and error-prone. Database tables must be restructured, data reloaded, and affected code altered in classes, stored procedures, queries, scripts, and so on. Each component must be unit tested, and the affected applications must be system tested before they can be rolled out. In large systems, most code changes are associated with conceptual model changes, each affecting between 100 and 1000 lines of code. Even identifying the impact of such changes can be challenging [11,12].

Yet conceptual models are inherently volatile [13]; they change for many valid reasons, including the correction of mistakes and misunderstandings, the evolution of requirements during system use and business change, because of the need for system integration, and in response to external changes such as new legislation. Because the resulting maintenance work is labour-intensive and costly, it can be difficult for organisations to adapt their information systems quickly. As a result they may be left unable to respond to business challenges with sufficient agility [14].

This is a significant "hidden" problem, which gets little attention. The link between conceptual models and software structures is, in fact, a form of software coupling [15]. In Section 2 we argue that it is undesirable and should be avoided.

Ways of lessening the impact of conceptual model change have long been sought, particularly in the field of schema evolution [16]. Many proposed solutions try to automate the job of updating table and program structures [17,18]. Despite promising research, few solutions have found widespread adoption. It remains costly for organisations to alter their information systems, once implemented, to match new or altered conceptual models [19].

1.2. Physical and logical data independence

Traditionally, the main defence against the cost of conceptual model change has been data independence, *"the decoupling of a database system's application-level interface from its data organisation"* [20]. The goal of data independence is to protect users from *"potentially disruptive changes in data representation"* and thereby to reduce the work involved when database structures are altered [21]. The well-known three-schema architecture was introduced in support of this goal, separating physical, conceptual and external schemas [22].

Historically, two main forms of data independence have been considered. The first, *physical* independence, is provided by the separation of physical and conceptual schemas; it allows data to be accessed by reference to its conceptual structure, regardless of physical structure [23]. The second, *logical* independence, is afforded by the separation of conceptual and external schemas. In theory, it allows programmers and users to access data using a consistent conceptual model, even when the underlying conceptual schema changes [24].

Physical data independence has been implemented comprehensively, in a range of database management systems. It provides many benefits: application programmers can safely ignore how data are physically stored, and database administrators can tune physical storage mechanisms without adversely affecting existing applications. Nevertheless, physical independence has been challenged recently by the "NoSQL" movement, which recognises that independence is often gained at the cost of performance and argues that