Int. J. Human-Computer Studies 86 (2016) 149–163

Contents lists available at ScienceDirect

Int. J. Human-Computer Studies

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

A mental model perspective for tool development and paradigm shift in spreadsheets ☆

Bennett Kankuzi, Jorma Sajaniemi *

University of Eastern Finland, School of Computing, Joensuu Campus, Joensuu, Finland

ARTICLE INFO

Article history:
Received 11 December 2014
Received in revised form
8 October 2015
Accepted 26 October 2015
Communicated by Francoise Detienne
Available online 9 November 2015

Keywords:
Spreadsheets
Mental models
Paradigm shift
Debugging
Errors

ABSTRACT

To address the problem of errors in spreadsheets, we have investigated spreadsheet authors' mental models in a hope of finding cognition-based principles for spreadsheet visualization and debugging tools. To this end, we have conducted three empirical studies. The first study explored the nature of mental models of spreadsheet authors during explaining and debugging tasks. It was found that several mental models about spreadsheets are activated in spreadsheet authors' minds. Particularly, when explaining a spreadsheet, the real-world and domain mental models are prominent, and the spreadsheet model is suppressed; however, when locating and fixing an error, one must constantly switch back and forth between the domain model and the spreadsheet model, which requires frequent use of the mapping between problem domain concepts and their spreadsheet model counterparts. The second study examined the effects of replacing traditional spreadsheet formulas with problem domain narratives in the context of a debugging task. Domain narratives were found to be easy to learn and they helped participants to locate more errors in spreadsheets. Furthermore, domain narratives also increased the use of the domain mental model and appeared to improve the mapping between the domain and spreadsheet models. The third study investigated the effects of allowing spreadsheet authors to fix errors by editing domain narratives, thus relieving them from the use of traditional low-level cell references. This scenario was found to promote spreadsheet authors using even more of their domain mental model in a manner that completely overshadowed the use of their spreadsheet mental model. Thus, from a mental model perspective, it is possible to devise a new spreadsheet paradigm that uses domain narratives in place of traditional spreadsheet formulas, thus automatically presenting spreadsheet content so that it prompts spreadsheet authors to think in a manner that closely corresponds to their mental models of the application domain.

1. Introduction and motivation

Despite the popularity of spreadsheets, the current spreadsheet paradigm has its drawbacks. Many spreadsheets contain non-trivial errors that not only go unnoticed but also have led to significant economic losses (Galletta et al., 1996; Panko, 1998, 2000; Powell et al., 2009). Although many tools and techniques have been developed to help address the problem of errors in spreadsheets, the problem nevertheless persists. Current tools (e.g., Abraham and Erwig, 2005, 2007; Ayalew et al., 2000; Clermont et al., 2002; Davis, 1996; Erwig et al., 2005; Grigoreanu et al., 2010; Kankuzi and Ayalew, 2008; Mittermeir and Clermont, 2002; Sajaniemi, 2000) are based on their developers' intuition; however, it would be more appropriate if tools were based on empirical evidence such as user studies based on cognitive psychology.

A common assertion is that humans have mental models of the systems with which they interact, and it is often difficult to explain many aspects of human behavior without resorting to a construct such as mental models (Rouse and Morris, 1986). It is therefore important to first understand what types of mental models are activated when spreadsheet authors are conducting different spreadsheet tasks. Understanding the mental models will lead to a better understanding of why the spreadsheet process is so error-prone and to enable the development of new tools and techniques that better correspond to spreadsheet authors' cognitive abilities.

Realizing that the term “mental model” has been defined differently by different researchers, in our context, we borrow the definition of Forrester (1971) which was also cited by Doyle and

☆This paper has been recommended for acceptance by Francoise Detienne.

* Corresponding author.
E-mail addresses: bfkankuzi@gmail.com (B. Kankuzi), jorma.sajaniemi@uef.fi (J. Sajaniemi).

http://dx.doi.org/10.1016/j.ijhcs.2015.10.005
1071-5819/

Ford (1998): a mental model is a mental image of the world around us that we carry in our heads depicting only selected concepts and relationships that represent real systems. A mental model, therefore, is a representation in a person's mind of the physical world outside of that person, i.e., a mental model is a type of mental representation. A mental model also depicts only selected concepts and relationships regarding a system and as such, does not represent the entire knowledge one has regarding a system but rather a specialized subset of that knowledge. Knowledge is, however, indispensable for understanding something, and knowledge without understanding is useless. Therefore, a user's mental model of a system reflects the user's understanding of what the system contains, how the system works and why the system works in that manner (Carroll et al., 1987). Thus, for example, a spreadsheet developer's mental model of a spreadsheet does not contain all possible information, but only those aspects of the spreadsheet that the developer deems appropriate for the task.

A person can also have simultaneous or parallel mental models of a system that reflect the different levels of abstraction the person may perceive of a particular system when conducting different tasks on the system (Carroll et al., 1987; Markman and Gentner, 2001). Thus, for example, one mental model could describe the technical aspects of a system and another model could describe the purpose of the system. Flower and Hayes (1980) observed that, in the process of writing, a person must work at several levels of abstraction simultaneously, which again suggests the existence of multiple mental models when one is conducting a task. Flower and Hayes (1980) also observed that each level of abstraction creates constraints that must be considered when considering other levels. Therefore, it is important that simultaneous mental models map to one another accordingly for optimal behavior.

In our research, we applied the theory of mental models in an investigation comprising three studies. The first study explored the nature of mental models of spreadsheet authors when those authors are conducting various spreadsheet activities. The second study gauged the effects of replacing traditional cell references by problem domain concepts on spreadsheet comprehension and debugging, and the third study investigated the effects of editing spreadsheet formulas using problem domain concepts. These studies not only illustrate the role of mental models in spreadsheet activities and suggest recommendations for spreadsheet visualization and debugging tools but also suggest a paradigm shift in spreadsheet calculation.

The rest of the paper is organized as follows: Section 2 discusses related works examining how other researchers have used the concept of mental models for programmers in general and spreadsheet users in particular, and how spreadsheet tools have tried to visualize spreadsheets in problem domain terms. Sections 3, 4 and 5 describe the first, second, and third studies, respectively. Section 6 presents a general discussion of the three empirical studies and concluding remarks are given in Section 7.

2. Related work

Research into the psychology of programming (e.g., Brooks, 1983; Corritore and Wiedenbeck, 1991; Letovsky, 1987; von Mayrhauser and Vans, 1995; Pennington, 1987) has revealed that programmers have several *mental models* of a program, including models such as the programming language level description and the application or problem domain level description of the same functionality. For example, Brooks (1983) theorized that program creation involves understanding the problem domain representation; abstract data and algorithmic structure representation; the concrete data and control structure representation of the program; and the mappings between objects and relations in adjacent representations. Such representations can be understood as mental models, i.e., images of the program that depict only selected concepts and relationships that represent the program.

To reveal the contents of a mental model, one must use a knowledge elicitation technique (Cooke, 1994). A common technique to elicit programmers' mental models is *program summary analysis*—a special form of content analysis (Stemler, 2001). Program summary analysis has been used, for example, to characterize mental models of novice- (Corritore and Wiedenbeck, 1991) and expert- (Pennington, 1987) programmers who attain high levels of comprehension, to characterize the mental models of students capable of reusing program code (Hoadley et al., 1996), to describe how mental models depend on the underlying programming paradigm (Good, 1999) or task type (O'Shea and Exton, 2004), and to evaluate learning outcomes in novice programmers (Hughes and Buckley, 2004; Sajaniemi and Kuittinen, 2005). These studies suggest that program summary analysis can be utilized to reveal novice and expert programmers' mental models and that the contents of the mental models can be used to characterize the task performance of programmers.

Program summaries can come in various forms, such as transcripts of audio or videotaped interviews and written explanations of programs. The basic idea of program summary analysis is to ask participants to provide a free-form explanation, or summary, of a program just studied. Thus, program summaries allow participants to express their thoughts in their own words, at their chosen level of abstraction and detail (Good, 1999). By omitting detailed instructions regarding the form of the summary, participants' own preferences guide the selection of information in the summary; thus, a wide variation in the responses is generally achieved. The program summary methodology avoids the problems of false positive results often associated with binary choice questions, and the difficulties in designing sensitive and reliable multiple choice questions (Good and Brna, 2004). In program summary analysis, the focus is not on the correctness of the summary; instead, the abstraction level and the types of information are more important characterizations of the mental model than an error-free memorization of the program code.

Different techniques can be used to analyze program summaries. For example, Pennington (1987) analyzed program summaries by dividing them into statements and classifying the statements into procedural (control flow), data flow or function. Good (1999) improved Pennington's technique by analyzing program summaries using two independent measures: information types classification and object description categories. Later, Byckling et al. (2004) evaluated Good's technique and suggested some improvements. In information types classification, program summary analysis is based on the type of information each statement reveals regarding a program. Conversely, object description categories focus on the manner in which individual objects are described in program summaries. There are seven categories in Good's object description categories classification: program only—references to objects that can occur only in the program realm; program—references to objects that can be described at various levels in program terms; program-real-world—reference to objects that can occur in both the real-world and program realms; program–domain —references to objects that can occur in both the program realm and the problem domain; domain—references to objects that can occur only in the problem domain; indirect reference—anaphorically referencing an object; and unclear—objects that cannot be classified because of ambiguity, lack of clarity or lack of identification.

As an example of the findings obtained by program summary analysis, Pennington (1987) observed that a program modification