Вестник Омского университета. Серия «Экономика». 2014. № 2. С. 20–25.

УДК 65.011.56

THE IMPLEMENTATION, REPRESENTED AND MENTAL MODELS OF BUSINESS SOFTWARE IN THE CONTEXT OF BRIDGING THE GAP BETWEEN RESEARCH AND DESIGN

M. Sakal, D. Pantelić, L. Raković
University of Novi Sad, Faculty of Economics in Subotica (Serbia)

The worldwide omnipresence and significant impact of digital products on a substantial number of complex variables of business community and everyday life by no means render the software development projects less risky endeavour. Starting from the significance of the behavioural dimension of software products for corporate purposes, based substantially on the research conducted by Cooper and Reimann, this article adopts a theoretical aspect in considering the relation between the implementation model, represented model and mental models of business software, as well as in overcoming the gap between research and design.

Key words: user-oriented interface development, implementation model, represented model, mental model.

1. Introduction

High-quality engineering and marketing processes used to suffice for producing an appealing product in the early days of industrial manufacture. However, the need to differentiate a product in relation to functionally identical competing product has opened space for the development of industrial design. The conscious inclusion of design into the production process, therefore, heralded the emergence of the powerful contemporary triad, identified by Larry Keeley (Figure 1): feasibility – sustainability – appeal. If only one of these three foundation stones of a product's market performance is significantly weaker than the other two, the product will not pass the market test. A model correspondent to this assumption is the two-dimensional product design model, with the form on the ordinate and meaning on the abscissa. However, with the emergence of an essentially novel type of products – digital products, which, if programmed properly, are capable of behaving in a practically unlimited multitude of ways, interactively, the two-dimensional product design model is incremented with an additional dimension, that is, behaviour.

2. The implementation model vs. the mental model

Interactiveness is the most significant dimension of digital products (including business software) which, in addition to complying with the rules of visual composition, demands exceptionally good knowledge of the context, and is not (only) a matter of aesthetic choice, but rather of comprehending the user's cognitive processes. Interaction with digital products is said to be «first-person exchange», or «non-verbal natural language» between humans and designer artefacts. Anticipation and design of such dialog is the essence of the design of interaction with digital products. Whether the business software is devised predominantly or exclusively from the viewpoint of engineering (i.e. point of view based on a set of professional imperatives stemming from the technological approach), the software product viewed as a represented model will be closer to the implementation-oriented than the mental model.

Drawing the represented model of the software product closer to the user's mental model, i.e. removal or at least major diminishment of antagonism between these two models by way of represented model is one of the driving forces of the user oriented interface development.

The **implementation-oriented model** (also known as system-based model) is the engineers' view of software. It can be said to be a set of information expressed so as to represent the manner the software *actually* functions, i.e. describes the details of the way the software product is implemented by means of coding. The building trajectory of the implementation-oriented model is defined by predominantly technical, but also commercial limitations.

The **mental model** (also termed the **user conceptual method** is a set of information expressing how users conceive, perceive and comprehend the mode of interaction with the software product. The mental model is a reflection of the user's viewpoint of the system, a sort of cognitive shorthand record of the use of the software product through interaction with its interface. The mental model rarely overlaps with the product's intrinsic logic.

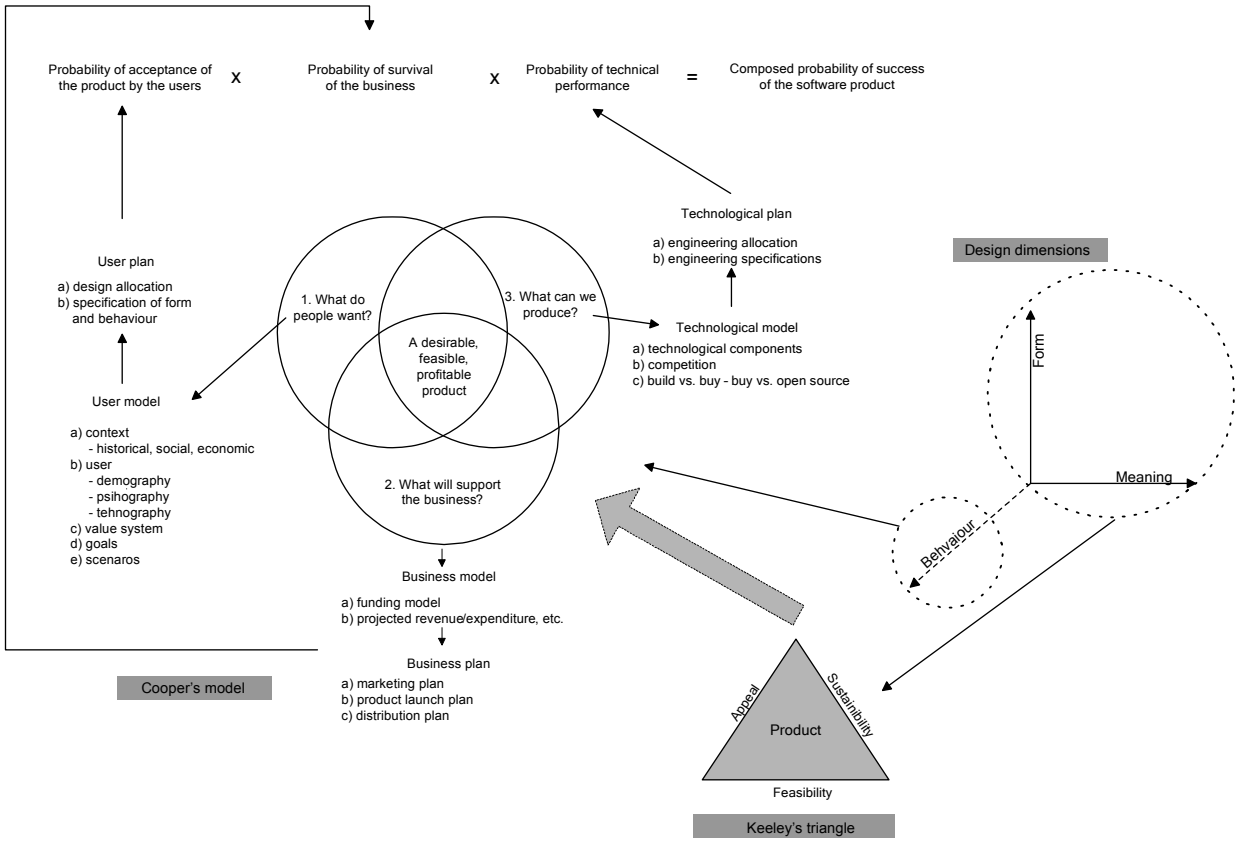

Figure 1

Source: adapted from Cooper & Reimann, 2003, p. 10.

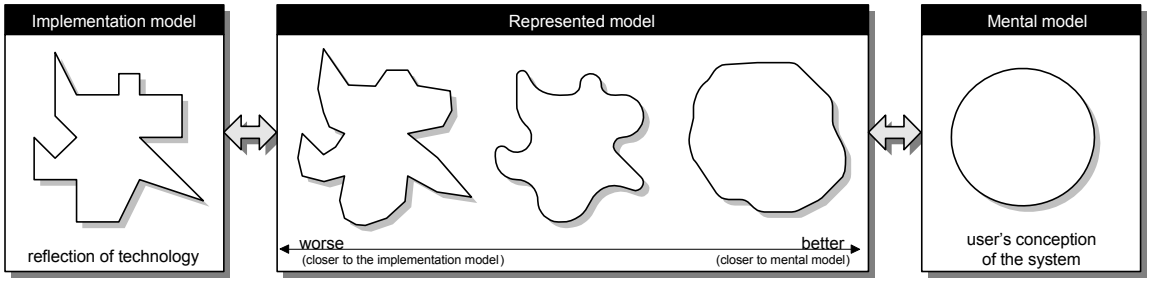

Figure 2

Source: adapted from Cooper & Reimann, 2003, p. 23.

The **represented model** (also known as the **designer model**) is a set of information selected by the designer to present the manner the software functions to the user. In other words, it is the "behavioural face" (often referred to as "the look and feel") of the software product, interface created by the developer. The role of the represented model is to remove (or at least diminish to a great extent) the disparities between the implementation model and conceptual model. The significant differences between the implementation model and conceptual model are typical of the software product, as the complexity of implementation makes it almost impossible for an average user to see the mechanicistic connection between his actions and the software's reactions. Although the importance of the designer model in software industry is extremely high, the represented model of a large number of modern-day software products is, unfortunately, still only a fairly identical copy of the implementation model. In this critique of this phenomenon, A. Cooper (2007), the father of Visual Basic and one of the world-renown interaction design gurus, claims that the

so-called computer literacy required from the user is nothing but a euphemism aimed at forcing the user to understand extraterrestrial logic, rather than adapting the user interface of software products to the users' thinking patterns.

The smaller the difference between the represented and mental model, the higher the efficiency of the software product use, and vice versa, the closer the represented model to the implementation model, the lower the chances for users to be able to use the software product efficiently.

Similar views existed sixteen years ago as well, but comparable to the idea of iterative design, they were not widely accepted. For instance, Norman (1998) gave a framework shedding light on the relationship between the conceptual model design and the user's conception of it (Figure 3). In his opinion, interaction comprises three components: the designer, the user, and the system. Interlinked conceptual models stand behind each of them:

– the designer model describes how the designer imagines that the system should work;

– the system image describes how the system actually works; and

– the user model describes how the user sees and perceives the way the system works.

If the system image does not enable the user to understand the design model appropriately, the user is bound to use the system inappropriately and inefficiently.

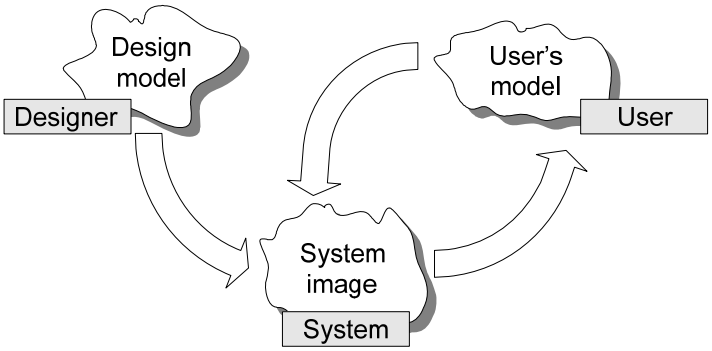

Figure 3

Source: Interaction Design Foundation (2006).

Software product users tend to form the simplest possible model, which is why creating a represented model simpler than an implementation model raises the quality of understanding the manner of system functioning. For instance, in a typical spreadsheet program such as Excel, the cells not seen in a given view become visible by moving the scroll bar. From the implementation viewpoint, cells do not exist, but only sets of data structures, with their values and mutually indicating relations, from which the coded algorithm synthesises a new image (with a new row of cells) refreshing the display, in response to the click on the vertical scroll bar.

Through the analysis of the evolution of the need for multidisciplinarity in the development of the user interface over various epoch of the development of software industry (although the expression "development stages" might be more suitable, in view of the pace of change of individual "epochs"), it becomes evident that:

a) in the early days of software industry, the design and implementation of user interface was mostly done by engineers;

b) that interface fairly accurately reflected the implementation-oriented model.

So, as a rule, software designed by engineers and mathematicians (who are familiar with the processes coded in the program down to the tiniest detail) is closer to the implementation model. Criticising the phenomenon that decision-making on which functionalities are to be incorporated into a software is guided simply by the existence of technical possibility of executing the given functionalities within the available technology, A. Cooper (2007) calls engineers (and programmers) "victims of technological intrigues", blind to the users' needs and objectives. Admittedly, software interface that reflects the implementation model is much easier to design: a command button must be planned for each function, a field for every input, page (or view) for each step of a transaction and a dialog window for each individual module. The appropriate reflexion of engineers' logic leads to a less coherent reflection of users' goals. If, for instance, left-click drag-and-drop is used without pressing the Control key, in the Microsoft Window Explorer, in case of a file posi-

tioned, for instance on the C partition of the hard this, this will execute a **relocation** (Copy+Delete) operation if the destination folder is on the same partition, whereas, if the destination is, for instance, a flash disk, the result will be **copying.** This revokes one of the basic heuristics of good interface design: consistency. The cause lies in the literal application of the implementation oriented model. Or, for instance, the SQL query that should generate the list of employees in Department A AND Department B will use the conjunction OR.

3. Bridging the gap between research and design

A higher degree of similarity of the represented model of business software products to the implementation model (rather than the mental model) results from a consistent application of a range of views that mostly used to be valid in the past, but the rapid technological development has caused them to become obsolete and turn into their own opposite. First of all, user interface design is not a visual "facelift" of the implementation model, but a detailed plan of behaviour and appearance of a software product defined on the basis of properly identified user requirements. User interface design should give the essential definition of a software product, based on users' goals and business requirements, respecting the limitations of the current technologies. User interface designers should therefore be possessed of a much broader range of knowledge and roles in comparison with traditional (graphic and industrial) designers.

A substantial amount of problems was also caused by over-specialisation and low-quality communication between experts of various profiles involved in software product development: researchers research the market, the research results are analysed by the same researches (and, only seldom, usability experts), and the conclusions are forwarded to designers, or even programmers, which is completely wrong. Although this may seem completely proper at first sight, a systematic cohesion is still missing when translating and synthesising the research into design solutions. Research is too frequently reduced to market research modelled on the marketing approach, and design to graphic design or superficial industrial design. Designers (or programmers) receive information on the users' tasks (the so-called task level information), rather than on their goals. Undoubtedly, information on users' tasks is useful for defining the layout, workflow and access to function by means of interface controls, but it is insufficient for defining the framework that can be used for defining what the product is, what it does, and in what way it should meet a broad spectrum of users' requirements.

The solution lies in including user interface designers into research. One of the main benefits the designers gain that way is empathy, that is, the ability to feel what the others feel. Thus, they will start thinking about the users long before beginning to design. On the other hand, research specialists frequently do not remember which information is essential from the designers' perspective. Involving designers into the research can significantly diminish this shortcoming as well. Focussing on qualitative, ethnographic data during the research has turned out to be a highly efficient solution.

A. Cooper and R. Reinmann (2007) see the solution to the problem of gap between research and design in applying the design framework shown in Figure 4.

A brief description of the model is given below.

Research

According to Cooper (2007), research should be conducted with a predominant application of ethnographic research techniques, but also the classical techniques such as interviews and questionnaires, conducted with the existing and potential users, stakeholders, software developers, experts in relevant areas (SME – subject matter experts), etc, then studying the literature, audit of competitive products, etc. Generally, the aim is to obtain qualitative (as well as quantitative) data on potential and/or current users, and the context of using the existing/future software product. For instance, one of the most significant outputs of observation and contextual interview is the set of the so-called usage patterns, which aid in categorising the modes of using the potential or existing product, i.e. setting the users' goals and motivation (specific and overall desired results of the use of the existing/future product). The usage patterns are a reflexion of the users' lifestyle, and in the business and technical domain, these patterns also help to map the professional roles. Usage patterns lead to the creation of the personae (user models).

It must be said that this is also the stage when business plans, marketing plans, branding strategies, production plans and other documents are studied, so as to get a picture of the objective circumstances in relation to business goals, technical, time-related and financial limitations.

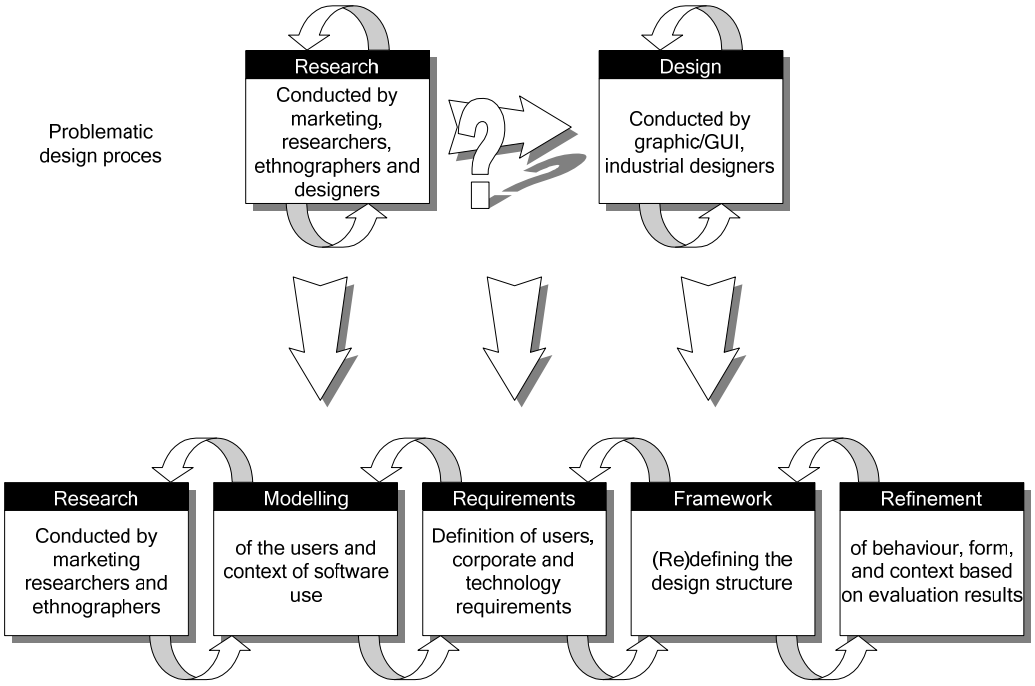

Figure 4

Source: adapted from Cooper & Reimann, 2003, pp. 15–16.

Modelling

In this stage, patterns are synthesised into domain models and user models, as follows:

– the **work flow model** comprises persons involved in the performance of the task, communication and coordination between them;

– the **sequence model** includes detailed steps to be taken so that the task can be performed; furthermore, it also describes the trigger that will lead to the execution of these steps, and the goal that will be achieved if the steps are taken properly.

– the **artefact model** describes physical objects, i.e. appliances required for task performance

– the **cultural model** describes system limitations arising from organisation culture;

– the **physical model** presents the physical structure of the task, for instance, office layout, communication network scheme, etc.

User models (personas) are detailed user archetypes representing various groups different in terms of behavioural patterns, goals and motivations. The user models:

– are the foundation of scenario-based narrative design approach, which will iteratively create design concepts later on in the Framework phase;

– provide feedback that improves the harmony and appropriateness of design in the Refinement stage;

– represent a powerful communication tool helping the developers and managers in explaining the design and setting priorities based on the user's requirements.

In this phase, designers use the methodological tools of synthesising, distinguishing and prioritising of individual user models, by researching the various types of users' *goals* and mapping the behaviour of identified user models, with the intention of establishing the (non)existence of gaps or overlaps. The extent of interaction between the needs of individual user models and future design is established. The relationship may be:

– **primary** – the user models' needs are unified enough to enable designing interface of a specific form and behavioural features;

– **secondary** – the primary interface serves the needs with minimum alterations or additions;

– **supplementary** – the needs are full met by the primary interface;

– **served** – the user model is not a direct software used, but is indirectly affected by its use;

– **negative** – the user model is an explicit, rhetorical example of *for whom* the product is *not* designed.

Defining requirements

This stage refers to the application of scenario-based design methods, where the focus is primarily on the goals and needs of a specific user model, and much less on individual specific tasks. User models become the pivotal characters of

these scenarios, and the designers research design trajectories by playing various roles, as well as analysing the functional needs (expressed in terms of objects, actions and context) for each interface-primary user model, based on the goals, behaviour and interaction and interaction of the examined model with other user models, in various contexts. This analysis is top-down by nature; it is performed through iterative purification of the contextual scenario, and begins with the description of the manner of using the current/future software product during the user model's typical working day. During the subsequent iteration, business goals and technical limitations are considered, and harmonised with the goals and needs of the studied user model. The output of this process is **requirements definition**, with balanced user-related, commercial and technical requirements of the design.

Designing the framework

This is the stage of synthesising the framework of user interface, by applying the conjunction between contest scenarios and:

– a set of general principles of interaction design

– a set of patterns of interaction design, providing general solutions for the classes of previously analysed problems.

The above mentioned patterns are hierarchically organised and evolve continuously as the new context appears. They should not be understood as limitations of designer creativity, but rather as solving problems with validated solutions.

Having been described at abstract level, the functional and data related requirements are then translated into element designs, taking into consideration the interaction design principles, and then organised by incorporating the patterns and principles into design sketches and behaviour description. The output is a definition of interaction framework, i.e. interface, a stable design concept providing a logical and general formal structure for implementing details.

A successive iteration of an increasingly focussed scenario delivers all these details to the Refinement phase. This approach often balances between top-down (pattern oriented) design and bottom-up (principle-oriented) design.

Refinement

Refinement is performed similar to the previous phase, but with a much more significant attention on the consistency of the tasks, applying walkthrough and scenarios of validation, concentrating on storyboarding paths through interface in high detail. User testing, experimenting, heuristic evaluation etc. methods are also used.

This phase culminates in in-depth written design documentation, form and behaviour specification delivered in paper or context-sensitive electronic form. All that is left is the physical design development.

4. Conclusion

The specific character of software products has resulted in the need for incrementing of the two-dimensional design model with an additional dimension – behaviour, i.e. interactiveness. Although interactiveness is the most significant dimension of digital product, business software designers are frequently guided by implementation-oriented rather than mental model when creating a represented model of a software product. This article presents a theoretical aspect of a possible systemic solution of this problem, which implies filling in the gap between research and design by modelling the users and the context of software implementation, by defining the requirements of the users, business and technology, and, if necessary, (re)defining the design structure.

1. Cooper A & Reimann R 2003, About Face 2.0 (The Essentials of Interaction Design), John Wiley & Sons, New York, USA.

2. Đurković J, Sakal M & Raković L 2011, 'Korisnički interfejs – očima korisnika', *XXXVII Simpozijum o operacionim istraživanjima (SYM-OP-IS)*, Ekonomski fakultet Beograd, pp. 296–272.

3. Galitz OW 2007, The Essential Guide to User Interface Design (An Introduction to GUI Design Principles and Techniques), 3rd edition, Wiley Publishing Inc., Indianapolis, USA

4. Hackos JT & Redish JC 1998, User and Task Analysis for Interface Design, John Wiley & Sons, New York.

5. *Mental models* 2006, Interaction Design Foundation, <http://www.interaction-design.org/encyclopedia/mental_models_glossary.html>.

6. Norman D 1988, The Design of Everyday Things, Basic Books, New York, USA

7. Sakal M 2007, Korisnički orijentisan dizajn softverskih proizvoda, doktorska disertacija, Ekonomski fakultet u Subotici.

8. Sakal M & Trninić J 2004, 'Uticaj kulture na razvoj korisničkog interfejsa', *Anali Ekonomskog fakulteta u Subotici*, br. 11, pp. 243–249.

9. Sakal M & Trninić J 2006, 'Grafički korisnički interfejs – istorijska perspektiva', *Anali Ekonomskog fakultet u Subotici*, br. 16, pp. 147–154.