In *Advances in Human - Computer Interaction I,* N. Avouris & N. Fakotakis (Eds), Patras: Typorama, pp. 65-70. 2001.

Ecological design of informatics systems: a new paradigm to achieve usability and effectiveness

Dimitris Nathanael
National Technical University of Athens
157 80 Zografos,
Athens, Greece
dnathan@central.ntua.gr

Nicolas Marmaras
National Technical University of Athens
157 80 Zografos,
Athens, Greece
marmaras@central.ntua.gr

ABSTRACT
After a short presentation of the contribution of ergonomics in traditional human-computer interaction and the concepts of usability engineering, we shift to the main design processes which aim to achieve usability and effectiveness. In a first phase, a critique is made to the systematic consideration of user needs from the initial stages of the traditional design process (waterfall process), as well as to the user-centered design approach. The rest of the paper deals with a new design paradigm for developing informatics systems supporting human work. The ecological design paradigm paradigm adopts a work system-centered design approach, and attempts to exploit the strengths of the existing approaches and to overcome their weaknesses. The proposed new paradigm derives from the concepts of cognitive engineering and the authors' experience in participating in the design of it systems.

KEYWORDS: usability engineering, user requirements analysis, user-centered design, ecological design, cognitive engineering.

INTRODUCTION

Ergonomics has first entered in the domain of information technology through the specialization of its man – machine interaction paradigm to become human – computer interaction (HCI). This research area has provided over the last decades with invaluable data for the design of input devices, and more importantly graphical presentation of information, so that computers become more adapted to human physiology and cognition. Nevertheless traditional HCI focuses on optimizing the so called "computer tasks" i.e. The interaction between human and computer without considering the context of use. Although research in this domain is still strong mainly towards new interaction types like natural language or 3d virtual environments, it can be said that for mainstream applications the contemporary solutions (e.g. Guis) are more or less adequate and have stabilized over the last decade.

The conviction that effective human computer interaction is not a goal by itself in real work settings, has led to the emergence of usability engineering as a new approach for ergonomics in the domain of information technology. Usability engineering is broader in spectrum than traditional HCI in that it gives emphasis to the actual work tasks that have to be executed by humans by the medium of information technology. Hence, in usability engineering the computer is seen as a tool for humans to accomplish a task in the context of their working environment. The main focus of usability is not to optimize human computer interaction *per se*, but rather to optimize the human work results both through increase of ease of learning, ease of use and human satisfaction.

Usability engineering has developed a number of methods and tools that are now widely used in industry, mainly because of their realistic business approach and informal character [11]. Most of these methods and tools follow a corrective strategy, that is to say, they influence the resulting interface by iterative evaluations at different stages of development. When a document under the name of "usability specifications" exists, it actually denotes the specification of measurable empirical evaluation criteria (which is indeed important, of course, but analytical), not the specification of user interface designs from an ergonomics perspective (which is constructive) [18]. The conceptual design of the interface, which we advocate in this paper, is of outmost importance for the final result of a software product, is dealt primarily by the development team and not by the methods of usability engineering.

If we consider informatics system (IS) design as a successive process of constraints satisfaction, it becomes evident that the prioritization of the constraints to be met, is a designer's task which influences heavily the final product. Thus the conceptual design usually results from the priorities (i.e. Constraint satisfaction) that are perceived as most important by the designer. In the great majority of commercial projects, design is ensured by the development team. This results in the so-called "technology driven approach" to software design. Typically, preferred platform constraints, similar past solutions and automation possibilities are of high priority for it developers (as they are more familiar with this type of constraints) and so they tend to shape the conceptual design of the software product to the detriment of the context of use. In the words of bailey [2]: *"if we do not clearly state the requirements from the beginning we cannot expect human performance considerations to be taken seriously. In fact, in the absence of human performance requirements, other system requirements will take precedence — those related to software or hardware development. When this happens the entire development process will focus on meeting these other requirements. And when the system is operational, people will be left to perform as best they can without adequate provision for ensuring an acceptable level of human performance".*

The technology driven approach results in an economical product with some consideration of usability. Through the past decades this was the most sensible way to proceed since the technical constraints posed for the development were far greater than they are today [13].

To overcome the weaknesses of the technology driven approach, two solutions were proposed. The first was to incorporate in the traditional design process, a systematic consideration of user needs from the initial stages. The second is known as *user-centered design*. In what follows, we summarize the main characteristics of these solutions and propose a new design paradigm, the *ecological paradigm* to IS design. This new paradigm attempts to exploit the strengths of the existing approaches and to overcome their weaknesses. The proposed new paradigm derives from the concepts of cognitive engineering [14],[17],[19] and the authors' experience in participating in the design of IT systems either from the initial stages or at a later evaluation stage [5],[7],[8],[9].

USER REQUIREMENTS ANALYSIS IN THE CONTEXT OF THE "WATERFALL" PROCESS

The traditional development process of IS follows a sequential approach, the so-called waterfall process, which essentially consists of the following steps: (i) user requirements analysis (URA), (ii) functional specifications, (iii) information – system architecture, (iv) construction, (v) evaluation – validation.

The user requirements analysis (URA), intends to define the objectives of the system, its users, the tasks to be performed by the users (or the functions of the system) and the context in which the tasks are to be performed (i.e. Organizational, legal, physical environments).

In the context of a waterfall IS development process, URA has the following characteristics:

- it is the first step of a sequential process
- it is performed mainly through interviews for the elicitation of knowledge from prospective users or user representatives
- it utilizes type scenarios (use cases) to define future tasks with the system and to communicate them to the users
- the result of the analysis is in text or structured analysis form.

Our experience with projects from the greek it industry, as well as references from the literature [3], lead us to suggest some major pitfalls of the above process:

URA being the first step of the development of an IS, is fast outdated through incremental changes that inevitably take place in the development process. The usual lack of updating means that it cannot be consulted at the later stages of development.

URA may or may not consider the current tasks and procedures. In any case what is mapped from the current tasks and procedures is purely descriptive. Correspondingly what is defined for the future tasks is purely prescriptive. That is to say that URA, more often than not, does not include elements at the intentional and semantic levels of tasks. The absence of this information results in a poor understanding of what is important as opposed to what is less important. Furthermore, the lack of semantics makes the data vulnerable to arbitrary rationalizations during development.

For ISs developed in organizational contexts, the elicitation of user requirements is usually performed by interviewing indirect users (i.e. managers). This is mainly because it is considered faster and safer than to talk to the actual users since they cannot have an overall picture of the procedures and objectives of the system. Another reason may be that evaluation of the final product is done through management, so it is the sensible way to go. This process can at best identify the official procedures of the organization and falls short of considering the actual practices of task accomplishment.

The description of use cases is performed for the most typical tasks with little consideration of unusual but probably critical scenarios. Although it is impossible to analyze or predict the totality of a work situations, the consideration of unusual scenarios is invaluable for understanding the affordances and constraints that shape the actual work practices [14].

Textual descriptions of tasks and context of use (i.e. use cases) is a poor way to transfer knowledge to users and developers alike. Users may understand the text but they are not good at evaluating abstract descriptions of future tasks, unless they experience them. To developers, textual descriptions of use cases, unless written with great care and discipline, seem rather ambiguous and finally an inappropriate way to transfer knowledge [15]. Our experience has shown that what developers really use from a URA are actual paper forms and data sets.

Structured methods for the representation of use cases (e.g. IDF-0) are also unpopular from an it management point of view. One of the main reasons is because structured methods are not designed to include human and organizational aspects and thus an important part of the knowledge acquired from the URA cannot be transferred further-on at the development [3]. It is also evident that this form of presenting the results of the analysis is not adapted to communication with most users.

An overall impression is that the tradition of soliciting user requirements only during the very early and very late stages of development is not adequate for assessing and improving usability. During early stages, the design is too amorphous for a user to really assess how the IS might enhance or constrain task performance. During late stages, the software structure has already solidified so much that user impact will be largely cosmetic. Finally as URA does not consider the semantics of the work, the user interface is defined by programmers and only after the information – system architecture step without significant input from URA.

USER-CENTERED DESIGN

The observation that a large number of IT projects fail as a result from the lack of careful consideration of the human and organizational aspects [3], has led to a new philosophy for the development of informatics systems supporting human work. The concept that usability should be the driving factor in software design and implementation is not particularly new; it has appeared in the literature under the guises of participatory design, and iterative design, as well as under the general name of the user-centered design [10],[12],[16].

The user-centred design (UCD) puts the future users of the system at the centre of the design process. The aim is to create IT systems that support the users within their working environment, rather than the users supporting the system. Adopting the user-centred design approach, the task requirements and constraints, as well as the way domain experts cope with them, would drive the choice of information technology to be used and the design of the systems [6],[12].

In general UCD methodologies share the following characteristics:

- UCD relies on a very thorough user requirements analysis. URA in the context of UCD adopts an ethnological approach and examines in detail the current work system. It is interdisciplinary in nature (i.e. analysis at cognitive, organizational and business process level) and uses various techniques (systematic observation – recording of activity, interviews, log analysis etc.). One of the main differences with traditional URA is that this type of analysis extends to the totality of users' tasks not just the ones that are to be incorporated into functions of the IS.
- The results of the URA are directly translated in design concepts and to mock-ups of user interfaces. This process is highly iterative through design – evaluation loops to ensure that the result is adapted to user needs.
- As mock-ups are converted into prototypes and then to full implementations, evaluation with users takes place at incremental stages, permitting feature-by-feature refinement in response to specific sources of user confusion or frustration.

The main difference of UCD from the traditional interface design in the waterfall approach of IS development, is that in UCD software tends to be designed from the outside to the inside (i.e. interface → technical specifications). Consequently, decisions concerning system architecture and technology solutions are constrained by the conceptual interface design and are performed at a later stage of the design process. Another important difference is that in UCD user evaluation is iterative and takes place incrementally at different stages of development.

Applying UCD for the design of IS presents some important advantages:

- the generation of user interface concepts early in the design process, ensures that they derive directly from the results of URA without influence from software platforms.
- early prototyping allows user's evaluation to begin long before implementation efforts have been invested in features or platforms that will have low usability payoffs
- the increased involvement of users throughout the development process has a very positive influence on user acceptance.

Despite these advantages, UCD is criticized for a number of inconveniences:

- The generation of the interface design concept independently from system architecture and development environment, puts more constraints to the development team. The interface concept might be hard to implement in a software environment that is adequate for the systems - information architecture.
- The development cycle of UCD requires a large number of iterations for user evaluation at most stages of the development. Thus, the earlier delivery time of a software product designed through UCD is generally longer than the traditional approach.
- Users may express conflicting needs. It may be very difficult, if almost impossible, to satisfy the needs of all users and still result in a coherent solution.
- It is not a very concrete methodology. Although a number of formal methods have been derived from

UCD (i.e. ISO 13407 Human-centered design process) there is not yet a firm consensus on the way to elicit user requirements and on how to translate these into design concepts. The process still remains highly creative thus very much dependent on subjective skills.

THE ECOLOGICAL DESIGN PARADIGM

One can distinguish two extreme paradigms in design. Analyze current solutions and redesign to optimize (the natural evolution of cyclical redesign [1], or design with a predefined conceptual solution. In the context of IT systems intended to support human work, these two extremes can be translated as follows; once the goals of the IS are set,

- the designer analyses thoroughly the current task execution, he identifies weak points and designs the IS to overcome the weak points identified (e.g. UCD).
- the designer starts with a concept or a powerful it tool and designs by adapting his concept or tool to the specific requirements of the task that is aimed to support (e.g. traditional software development).

In the real world, most IS developments lie in between these two approaches. The existence or not, of a well defined work system greatly influences towards one or the other approach. One can distinguish two basic cases concerning the current work system and tasks:

- the tasks to be supported by a new IS are currently executed in some way within a given work system (e.g. The monitoring of an industrial process), or
- the tasks to be supported by the IS are new in a work system that is also new (e.g. The web masters work a few years ago).

The latter case is not so common. Usually a work system is already in existence and/ or the tasks to be supported by the IS are already executed in some way; that is to say that individual task goals exist and are currently achieved by a more or less specified sequence of activities.

But even if no work system exists and the tasks are totally new, the designer, unless driven by a breakthrough technology, will most probably search to find some reference situations in order to “grasp a concept” and facilitate his work by “studying the solutions of others” (e.g. the influence of the first successful e-commerce sites such as Amazon® to subsequent ones).

Thus what acts as a starting point of the design is the study of reference situations. The most informative is the actual work system that the IS will support (if in existence). Other preferred reference situations include situations where similar ISs are already functioning, as is the usual case of ready-made customizable software solutions (e.g. warehouse management applications).

Therefore we suggest that the designer whether adopting a UCD approach or driven by the opportunities of an existing technological solution, starts by studying reference situations. In doing this, the designer either explicitly or implicitly needs to:

- break down the objective to a specific set of goals,
- identify the main entities and their interrelations in order to ensure effectiveness (i.e. functional specifications in software engineering terms) and
- see what the new design should pay attention to, in order to ensure efficiency and user acceptance (non functional specifications in software engineering terms).

In all current approaches, the third point, which is the specification of the “non functional requirements”, is performed implicitly or explicitly through the URA.

In the traditional software development process the URA is performed with a normative approach (i.e. it prescribes how the system will behave *vis-a-vis* to its users). In the UCD process, URA adopts a more pragmatic approach (i.e. through the iterative user evaluations, it tends to describe how the system really behaves in practice).

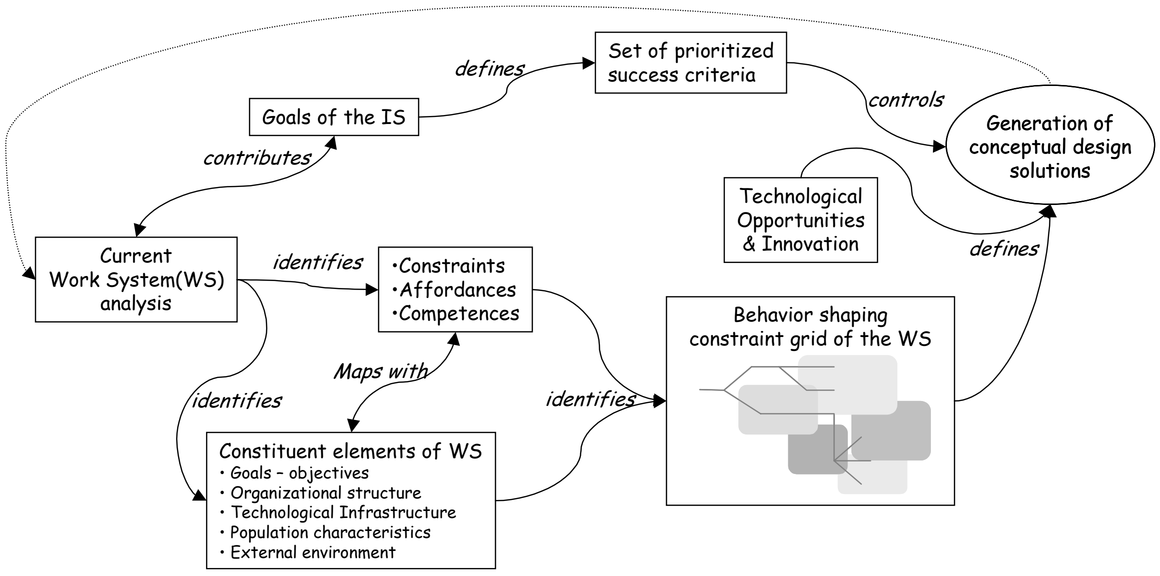
However, it is a well-known fact that work systems cannot be exhaustively prescribed; neither can their behavior be *a priori* fully described. Work systems (i.e. socio-technical systems) are goal directed and adaptive, to a large degree because of the very nature of humans.

Based on the above fact, we believe that the conceptual design of IS intended to support human work call for a paradigm shift from a *normative* method of analysis to a *formative* one. Normative models prescribe how a system should behave, formative models specify the requirements that must be satisfied so that the system could behave in a new desired way [17].

According to this new paradigm, the *definition* of user requirements is substituted by the *identification* of the goals and constraints that shape the behavior of a work situation and the competences of its human actors. We propose the use of Ergonomic Work Analysis [4] for the study of the relevant reference situations, coupled by a translation of the findings in what is called the *behavior shaping constraints grid* of the work situation [14].

Ergonomic Work Analysis offers the possibility to adopt such an approach. The current work situation is analyzed to identify not only *what* is being done (i.e. the behavior) but also *why* it is being done in this way (i.e. the constraints that shape this behavior in conjunction with the competences of the human actors)

The *behavior shaping constraints* of a work system, are those elements that shape work practice as experienced by the goal-directed, competency-moderated individuals. Constraints can be prohibiting or delimiting *in form*, facilitating or restrictive *in function*. They may originate from different constituent elements of the work system, e.g. the organizational structure, the technological infrastructure, the population characteristics, the external environment, or a combination of the above.


Set of prioritized success criteria
defines
Goals of the IS
controls
Generation of conceptual design solutions
contributes
Technological Opportunities & Innovation
defines
Current Work System(WS) analysis
identifies
•Constraints
•Affordances
•Competences
Behavior shaping constraint grid of the WS
Maps with
identifies
identifies
Constituent elements of WS
• Goals – objectives
• Organizational structure
• Technological Infrastructure
• Population characteristics
• External environment

The identification of constraints *form* and *function* as well as the mapping of these with the constituent elements that they originate from, form the behavior shaping constraints grid for the work system. Constraints that originate from elements of the work system that will not change after the implementation of an IS should be considered as *invariants* in the context of the IS development.

The *invariants* are a cognitive construct to signify a subtotal of the behavior shaping constraints that tend to remain stable even when important aspects of a work situation change through time (voluntarily or not). Invariants can also be defined as *sources of regularity* of a work system.

The form and function of these invariants as well as their interrelations, in the context of a specific IS development, can be seen as the conceptual design grid. Put in other terms, they delimit the degrees of freedom for the new design (e.g. they sketch what the new design must satisfy so that the system can behave in a new desired way).

As it is presented in *Figure 1*, the work analysis identifies the constituent elements of the work system and the constraints that shape its behavior. The mapping of the above (i.e. which constraint originates from which element) forms the *behavior shaping constraints grid* of the work system.

The analysis of the system objectives deals, with what the IS is intended to do, by defining a set of success criteria or goals. How the system should do what is intended to (i.e. the conceptual design *per se*), is defined by examining the technological opportunities in conjunction with the behavior shaping constraints grid. This purely exploratory process is obviously controlled by the objectives of the IS by means of the set of success criteria or goals.

Figure 1: Overview of Ecological Design process

EPILOGUE

The short description of the ecological design paradigm, demonstrates it shares common scope with use-centered design, however:

(i) It is rather a work system-centered approach than a user-centered approach.

(ii) It permits the conceptual design of the IS based on a formative model of the context and thus reduces the empiric character and need for many assessment loops of user-centered design.

(iii) It affords innovative technological solutions to be tested for compatibility with the behavior shaping constraints grid at the conceptual design stage *per se*.

(iv) It overcomes the problem of conflicting needs of various human or other work system's agents, by identifying the invariants (i.e. sources of regularity) of the context.

The ecological design paradigm was first adopted for the design of IS addressing cognitive tasks in complex work systems (e.g. IS supporting control of nuclear power plants). It requires a laborious and multidisciplinary analysis, however there is evidence that it is a promising design paradigm even for less complex work systems (see for example the bookhouse system developed by Rasmussen et al. [14] or the MEDICO system developed by Nathanael [8].

REFERENCES

1. Alexander, C. (1964). *Notes on the synthesis of form*. Cambridge, MA: Harvard University Press.
2. Bailey R. W. (1982). *Human performance engineering: a guide for system designers*. Englewood Cliffs, N.J.: Prentice-Hall.
3. Clegg, C. et al. (1997). Information technology: a study of performance and the role of human and organisational factors. *Ergonomics* 40(9), pp.851-871.
4. De Keyser, V. (1991). Work analysis in French language ergonomics: origins and current research trends. *Ergonomics*, 34(6), 653-669.
5. Marmaras, N. & Pavard, B. (1999). A methodological framework for development and evaluation of systems supporting complex cognitive tasks. *Cognition, Technology & Work*, 1(4), pp.222-236.
6. Marmaras, N. (2000). User needs analysis within user-centred design. In *International Encyclopedia of Ergonomics and Human Factors*, W. Karwowski (ed.), London: Taylor & Francis.
7. Marmaras, N. & Nathanael, D. (2001). *Usability study of the MIS of the Greek Ministry of National Economy*. Unpublished report (in Greek).
8. Nathanael, D. & Marmaras, N. (2000). User-centered design in practice: A medical tele-consultation management application. In *Proceedings of the 14th Triennial Congress of the International Ergonomics Association*, San Diego: HFES.
9. Nathanael, D., Marmaras, N. and Matsakis, I. (1997). Ergonomic work analysis for the development of a telematics application for medical assistance. In *From Experience to Innovation - IEA'97*, P. Seppala, T. Luopajarvi, C.H. Nygard, M. Mattila (Eds.), Helsinki: Finnish Institute of Occupational Health, Tome 7, pp. 489-491.
10. Nielsen, J. (1992). The Usability Engineering Life Cycle. *Computer*, March 1992, pp. 12-22.
11. Nielsen, J. (1993). *Usability Engineering*. San Diego: Morgan Kaufman.
12. Norman, D. A., (1986). Cognitive Engineering. In *User-Centered System Design: New Perspectives in Human-Computer Interaction*, D. A. Norman and S. W. Draper (eds). Mahwah: Lawrence Erlbaum Associates, pp. 31-62.
13. Norman, D.A. (1999). *The invisible computer*. Cambridge: MIT Press.
14. Rasmussen, J., Pejtersen, A.M.& Goodstein L.P. (1994). *Cognitive Systems Engineering*. NY: John Wiley & Sons.
15. Ravid, H., Berry D. M. (2000). A Method for Extracting and Stating Software Requirements that a User Interface Prototype Contains. *Requirements Engineering*, 5, pp225–241.
16. Rose,J., Shneiderman, B. & Plaisant, C. (1995). An Applied Ethnographic Method for Redesigning User Interfaces. *Proceedings of the Symposium on Designing Interactive Systems: Processes, Practices, Methods, & Techniques*, Ann Arbor, pp. 25-31.
17. Vicente K.J. (1999). *Cognitive Work analysis*. Mahwah: Lawrence Erblaum Assosiates.
18. Vossen P. H., Maguire M. (1998). *D4.2 Guide to Mapping Requirements to User Interface Specifications*. Telematics Applications Project IE 2016. Unpublished report.
19. Woods, D.D. & Roth, E.M. (1988). Cognitive engineering: Human problem solving with tools. *Human Factors*, 30(4), 415-430.