Understanding HCI methodologies

Peter Warren

University of Natal, Pietermaritzburg peterw@cs.unp.ac.za

Abstract

The paper proposes a framework for discussing methodologies for developing interactive computer systems. It uses this framework for a sample of systems and shows how certain insights about missing aspects of the methodologies emerge.

Keyword: *Human computer interface. Methodologies, Software engineering, interactive computer systems.*

Computer Review Categories: *D.2.1, D.2.2*

1 Introduction

Usability problems have been cited as a major contributory cause for the slow productivity gains following six trillion US dollars of investment in information technology [4]. If there ever was a software crisis then this is it. Not that there is a shortage of advice as many books, articles and journals pour out a veritable flood of information on human computer interaction (HCI). Much of it is sound, some of it is contradictory but either way it is hard for even a moderately well read practitioner to make sense of all this material. No wonder that Warren and Viljoen [11] (amongst others) found that students have very little idea about how to design an interface. There is a need, therefore, to be able to understand and compare these solutions, and for a more overarching point of view.

Our approach will be to provide a reference model (RM) which can be used to compare solutions. This will allow the various approaches to be placed in a context. The RM model is analogous to the software development life cycle (SDLC) in that it provides a way to reason about how software is created. But unlike the SDLC the RM is not a set of phases, but a number of "objects" that are constructed and this provides another and complementary perspective. The model itself cannot be validated any more than the SDLC or Norman's stages of interaction [7] can be validated. Nor will it, per se, provide a method of choosing between various methodologies as the RM is neutral. However, what it does do, is to provide a framework which will allow various writings on HCI to be discussed.

In the rest of the paper we will firstly describe the model and secondly describe how it was applied to a number of approaches in order to gain insight into the scope and techniques used. No doubt some readers will argue that the RM makes distinctions that are not real and others will argue that important boundaries are missing. This is similar to the way the SDLC has been used where, for instance, the object orientated community has argued that the boundary between design and analysis should be blurred. This is not a weakness, it is part of the reason for having a model so these issues can be discussed. The discussion of some sample"methodologies" is done to indicate how the RM can be used to provide an insight into these techniques. It is

important to understand that we are not attempting to create a new methodology but a way of analysing and understanding methodologies.

2 The submodels

It is contended that the software development process consists of creating a "model" of the software and its environment, which in turn is decomposed into a number of submodels. We are not going to be very precise about what each submodel contains, rather we will consider them to be containers which might be seen differently by the various methodologies. It might be argued that some of these submodels require further decomposition and that could well be so, but we will argue that to combine any two of them will be harmful to our understanding of the process. Our purpose is to look at the content of what is to be achieved independently of the software engineering processes that can be used to achieve it. We can now look at the individual submodels in more detail.

The psychological model (Psycho) The psychological model embraces all those issues that affect the actual end user interacting with the system. It contains all the material discussed under the umbrella of cognitive psychology, the user's mental model of the system and even motivational concerns. Cooper [1] even goes as far as to distinguish between the user's needs on the one hand and desires on the other.

The sociological model (Socio) The sociological model has to do with the broader community that might be impacted by the running of the system. A warehouse clerk may well have to interact with a stock system and one would have to create a psychological model for such a person. The sociological model, however, would include the customer indirectly attempting to seek service via the clerk, and the clerk's superiors needing reports to control the business. It would also involve any other person who may be consulted as the clerk attempts to interact with the system.

The task model (Task) The task model is concerned with modelling the user's activities. The task model is considered to be independent of the way in which those tasks are carried out in any particular system. Techniques such as task analysis [6], and use case analysis [3] are often used to document the tasks that have to be carried out. The task model does not include any interaction with a computer.

The conceptual model (Concept) Techniques such as use case analysis contain much temporal information, and there is a need for a more abstract view of the services provided by the system. The conceptual model is a view of the system as seen by the user and is often expressed in an object-oriented way.

The developer's model (Develop) The conceptual model discusses what the system should do from the point of view of the person who interacts directly with the system. The developer's model contains much more than that, since developers are concerned with issues such as cost of producing code, the maintainability of that code and the efficiency with which it runs. In a sense the developer's model is concerned with the implementation of the conceptual and interaction models.

The interaction model (Inter) The interaction model is the other side of the system from the task model. It provides the user with a way of manipulating the objects and services provided for by the conceptual model. It is not to be confused with the task model, the task model describes what has to be done, the interaction model implements the user interface issues associated with tasks. In a good system there will be a close relationship between the two, but to see them as identical is to confuse the problem with its solution. Nevertheless the distinction is a fine one, and one can expect some disagreements on the boundary between the two.

2.1 Context

The submodels operate inside a given context that defines other dimensions for the development. These contexts could be thought of as further submodels but a context seems more appropriate as they impinge on virtually all of the other submodels.

Stakeholders The stakeholders are those who could or should influence the development of the software. It includes the various kinds of developers such as interaction designers, analysts and programmers, but also the users and those who might pay for the project. The methodologies will vary depending on which groups are involved in the various stages and submodels.

Software engineering Software engineering (SE) describes the process by which software is created, and thus the order in which the various submodels are visited, and who is consulted. Variations such as waterfall versus spiral, participatory versus developer based, and techniques to create the code are the kinds of issues discussed.

Technology Just as the wider social setting in which a computer system operates is important, so is the technological environment. For instance a photo realistic virtual reality system is not a reasonable design specification using the current kinds of computers found on desktops.

2.2 Combining the submodels

It would be tedious and not very enlightening to discuss all the ways in which the submodels can be combined. Rather some inter combinations are undesirable and we will discuss these.

The conceptual and developer's models Object oriented (OO) programmers contend that these techniques are natural representations of the world [5] and they may well be. However programmers are not only concerned with modelling power but many other issues as well like code reuse, efficiency, simplicity of the programming and so forth. While OO techniques are powerful and useful, the concerns of programmers are very different from the concerns of end users [2]. If the conceptual and developer's models are combined there is a grave danger that the programmer's view of the world will intrude upon the user interface. This is considered by one and all to be an undesirable outcome [7].

The task, conceptual and interaction model Dayton *et al.* [9] have argued that the conceptual model (which they call the "Task object Model") is only necessary if one wishes to map the essentially procedural task model onto a non-procedural approach, such as object oriented user interface. If the interface maps very closely onto the task model then (they argue) the conceptual model is unnecessary. Even in this case it would seem a mistake to abandon the conceptual model. The conceptual model is necessary to shield the interaction model from the task and developer's models. However it does not necessarily exist in code, and could simply be part of a design document.

3 Interactions

Figure 1 shows the design influences (represented as arrows) at play during the development of a computer system. It is important to distinguish these from information flows and temporal relationships. For instance, information will flow from the developer's model to the conceptual model and thus to the interface in the running system, but we do not show arrows in that direction. We are saying that the design of the developer's model should not influence the conceptual model because to do that would be tantamount to the solution dictating the problem. We indicate an arrow running in the opposite direction, namely from the conceptual to developer's model, indicating design influences.

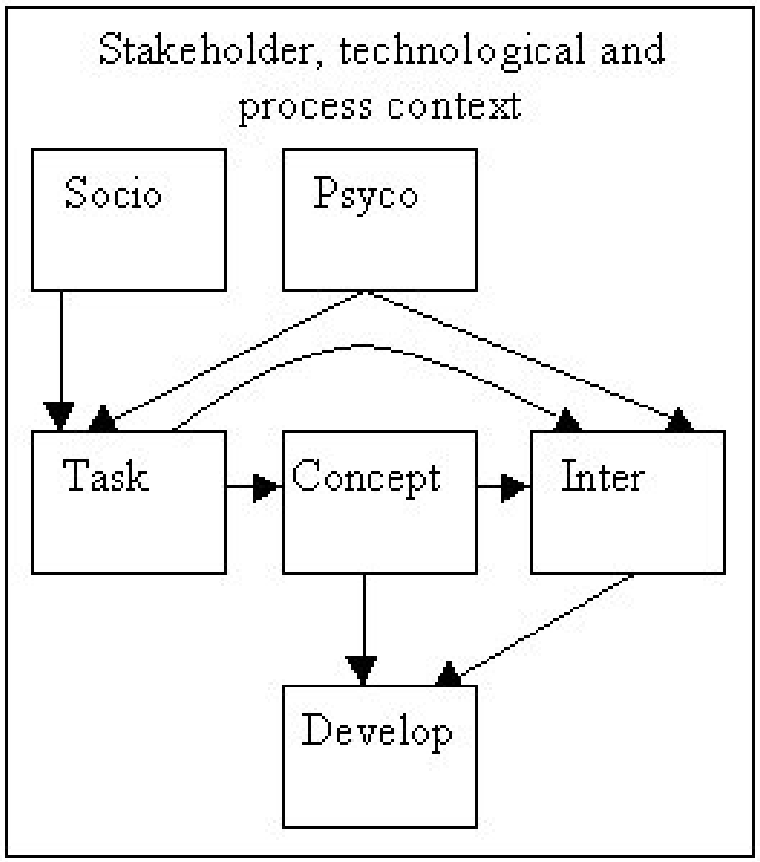

Figure 1: The figure shows the various sub-models that make up the reference model. The arrows show the influences that take place during the design of the system.

The diagram emphasises that the system is embedded in a context, namely the technology that enables and constrains the design, the stakeholders who play a role in shaping its form, and finally the process that is used to construct it. Inside this context, the sociological model influences only the task model. The sociological model does not influence the user directly but rather the sociological model influences the tasks the user carries out. This may be a point of disagreement. It is also noted that the sociological model has no direct influence on the way the interaction takes place with the computer. Such an influence exists but it is indirect. The psychological model, on the other hand, has a direct influence on both the tasks to be carried out, and the way the interface is constructed to carry out those tasks. The task model describes the tasks that the user performs. The conceptual model is an abstraction of those tasks, representing the data and services that the user interacts with. Some information may be lost when the conceptual model is abstracted from the task model, and the task model thus directly influences the interface. Finally the developer's model is determined by (but does not influence) the interaction and the conceptual models.

The main undesirable influence is the interaction model dictating to the task model what is to be done. This would mean "how" it is done dictates "what" is to be done! The other undesirable influence is from the developer's model to anything else. The problem is the same as the above.

4 Some methodologies

The idea of comparing "methodologies" below is to show how the RM can be used to reason about them and their scope. They are not intended to be a representative sample nor is the discussion intended to rank or select a method. We simply draw attention to interesting aspects of the methodologies.

4.1 The naïve developer's approach

The main problem with nave developers is the total lack of any planning or structure to their efforts. This results in any and all of the undesirable interactions taking place between the submodels. In particular the developer's model influences all of the rest of the system, as, what is easy to do, is what gets done. Careless rapid prototyping may also suffer from this malaise as eventually the system becomes the definition of the system.

Warren and Viljoen [11] studied a number of students attempting to design a user interface for a video store. What they found was that these designers created an interface directly in terms of tasks that had to be carried out. This in turn lead to the one screen per task syndrome, a malady that Cooper [1] calls dialog box pollution. A large number of dialog boxes is undesirable because at each page switch the user looses context making exploratory learning difficult. In RM terms the intermediate conceptual model is missing.

A similar problem can be expected to occur when the developer's model directly influences the interaction model. This particularly manifests itself in the one screen per relation approach which some database management systems tend to encourage. But a relation does not necessarily map onto a concept in the user domain so this technique may not be appropriate.

4.2 Cooper's analysis

Cooper's [1] [2] highly entertaining accounts of what is wrong with modern software can largely be attributed in RM terms to the developer's model influencing the rest of the system. An

example is that programmers tend to see each branch with equal importance because they expend the same effort writing the code. The end user might prefer the most frequent branch to be defaulted to avoid making the same choice each time. For instance, it is irksome to have to choose printer parameters every time when they hardly ever change. Cooper also dwells on the programmer's zero, one, many view of the world which again does not match an end user view where 10 items is a very different phenomenon from 1000 items. Cooper further claims that once code starts to be written this code is caste in stone, resulting in the developer's model harmfully influencing the final product if the code has been designed without reference to interface issues. His prescription is to ensure that the developers are only involved in the developer's model and nothing else, and only after the other submodels have been constructed. Cooper also makes an important contribution (or claim) regarding the development of the psychological model. Cooper is concerned about software that attempts to be all things to all people with the inevitable consequence that it is nothing to anybody. In turning Shneiderman's [8] stricture to accommodate diversity around, Cooper introduces the concept of a "persona" and the software is designed to cater for at most a few persona. The persona are roughly worst case users of their type. He claims that interface designers can visualise such a persona and draw accurate conclusions about their needs without the requirement of abstract psychological models. While Cooper's writings are worth reading for their entertainment value alone, they also make many telling points. From the point of view of the RM, however, they could not claim to be a complete description of how to design and implement interactive computer systems.

4.3 USDP

The unified modelling language (UML) has rapidly established itself as the standard notation for modelling software. An important adjunct is the Unified Software Development Process (USDP)[3] and thus it is interesting to see the extent to which the USDP is in harmony with the RM. USDP is a traditional software engineering methodology in which virtually the only attention given to the user interface is the identification of a "worker" called a "User Interface Designer". USDP is "use case driven" where a use case is defined as "a piece of functionality in the system that gives a user a result of value". The entity making this interaction is called an "actor" and therefore described by the psychological model. However very little seems to be done to model the actors in the USDP. While the USDP claims that use cases have been adopted "almost universally" there are some important critics [5]. Use cases seem to describe actors interacting with what we have called the interaction model. It is not clear that a separate "solution independent" task model is being constructed which could later be mapped on to one of a number of different interaction models. The RM would regard this apparent failure to distinguish between the tasks and the implementation of the tasks, as a weakness. The fact that more than one person at a time may be interacting with the system does not seem to be modelled, nor does the technological context. The situation is summarised in Table 1.

The RM would thus make the following predictions regarding USDP. In so far as it does not have a clear distinction between the Task, Conceptual and Interaction model we can expect a one to one mapping between the way the tasks are described and the resulting user interface. In the USDP [3] a use case is described via a state transition which in turn will encourage the largely obsolete "question and answer" type interface [6]. Meyer [5] considers that use cases provide too many temporal constraints and discusses similar kinds of problems.

Thus USDP

- It is an incomplete theory needing more techniques to model the sociological and technological setting of the interaction.

Table 1: USDP and RM

USDP	**RM**
Actor	Psychological model
Use case	Task model / interaction model
Control + entity class	Developer's model
Boundary class	Conceptual and interaction models
Workers	Developers in the stakeholder model
USDP process	Software engineering model
Not modelled	Sociological model
Not modelled	Technological context

Table 2: Newman and Lamming and the RM

RM	**Degree supported by NL**
Sociological	Continually emphasised.
Psychological	Extensive material.
Task	Task models used to document the user interaction with the system.
Conceptual	No "interaction independent" conceptual model is created.
Interaction	Extensive treatment of user mental models.
Developer	This is not discussed.
SE	Extensive coverage of process.
Technological	No discussion of technology.
Stakeholders	Very little discussion.

- It is a process largely involved with construction of the developer's model and is weak in handling the user interface issues. We can anticipate further effort to integrate human factors into the UCSP, an expectation that echoes Wesson's [12] agenda.

4.4 Newman and Lamming

An analysis of Newman and Lamming's (NL) approach is given in the Table 2.

The approach is compatible with the RM but does not really discuss how to construct the final product, but more the process that needs to be adopted to construct it. The RM would predict that the user of the NL approach will be a little uncertain on how to actually construct the various parts.

4.5 The Bridge

The Bridge is, according to its authors [9] "a comprehensive and integrated methodology for quickly designing object-oriented (OO), multi-platform graphical user interfaces (GUIs) that definitely meet user needs". This is a bold claim. It is a proprietary methodology taught by Belcore requiring very definite procedures to be adopted. As this methodology is part of the inspiration for the RN it is not surprising that it is compatible with it. The Bridge deals largely with the issue of creating the task, conceptual and user interaction models, and the processes that are needed to make this happen. Interestingly the developers are involved in the design of the user interface. This is not entirely consistent with the notion that there should be no influence from the developer's model to the conceptual model or with Cooper's views.

Table 3: Union and RM

RM	**Degree supported by UNION**
Sociological	Outside the scope of UNION.
Psychological	The psychological model is implicit.
Task	Is part of the Interaction model.
Conceptual	Called the Problem Domain (PD) Object Model and deals with the user's perception of the objects in the system.
Interaction	Split into the "User Interface Object Model" (static aspects), and the "Task Model" (dynamic aspects). Only OO models.
Developer	No clear difference with PD.
SE	Complete and detailed.
Technological	Would be taken into account when mapping "abstract task model" to interface.
Stakeholders	Very little discussion.

4.6 UNION

Wesson [12] has long considered that software engineering methodologies fail to cater adequately for the needs of interactive systems. One would expect wide spread coverage of RM concepts in her UNION methodology and this is what we find. Her methodology is analysed in Table 3.

The UNION follows a strongly user centred and task oriented approach with a detailed SE model to guide the process. Nevertheless some aspects of UNION's task model would be considered as part of the RM interaction model. At the same time UNION makes distinctions between the abstract task model and the techniques needed to implement this model for specific interface. This useful distinction is not made in RM which considers both to be part of the interaction model. The split between the task and interaction models is placed differently in UNION. The advantage of the UNION approach is that the user interface is closely coupled to the way the tasks are perceived, but there is also a danger that the solution may impose itself on the problem.

The RM makes the following observations about UNION. The methodology will require extension when groups are involved as the sociological model is missing. Additional attention, outside the methodology, may have to be given to selecting the technology. RM and UNION have different perceptions of the task model with tradeoffs between the two approaches. There is also less of a clear distinction between the developer's and conceptual models than there is in RM. On the other hand UNION is one of the few attempts to cover as much of the RM issues in one place and in the context of a strong SE model.

5 Discussion and conclusion

The RM is a framework for discussing methodologies that create interactive computer systems, and its primary purpose is descriptive rather than prescriptive. For instance it allows us to compare who is involved in the design process, in what order various issues are addressed and how the various issues are tackled. However it is also a model that makes certain predictions. It firstly says that if submodels are not present or two submodels are combined in a given methodology, then that methodology is incomplete.

Application of RM has allowed us to understand the nature of nave developers' problems.

It has allowed us to place the provocative writings of Cooper [1] [2] in context. It can take a standard textbook such as [6] and discuss in what areas the student will be getting advice and in what areas he will not. In this case many of the RM concerns are addressed. It reproduces from an HCI point of view some of the concerns about use cases that Meyer [5] has articulated from an OO point of view and others have found from experience. UNION [12] is seen to be a methodology that decomposes the problem slightly differently from RM and provides the user with guidance on the processes to be followed. We contend, therefore, that RM does allow us to reason about HCI methodologies without claiming that other perspectives will not yield other insights or that the RM framework is the only one.

What credence can be placed on the insights that RM gives? It can be judged on its own internal consistency, the predictions it makes about other techniques and whether others find it a useful approach. RM cannot be validated empirically because it is an interpretative view [10] where the models can never be shown to be right or wrong only more useful or less useful. It can only be left to readers to judge if the results are useful to them.

References

[1] Alan Cooper. *About Face*. IDG Books Worldwide, 1995.

[2] Alan Cooper. *The Inmates Are Running the Asylum*. SAMS, 1999.

[3] G. Booch I. Jacobson and J. Rumbaugh. *Software Development Process.* Addison-Wesley, 1998.

[4] T.K. Landauer. *The Trouble with Computers*. MIT, 1998.

[5] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1999.

[6] W.M. Newman and M.G. Lamming. *Interactive System Design.* SAMS, 1995.

[7] D.A. Norman. *The psychology of everyday things.* SAMS, 1988.

[8] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, 1998.

[9] Al McFarland Tom Dayton and Joseph Kramer. *L. Woods (Ed), User Interface Design*, pages 15–56. CRC Press, Boca Raton, FL., 1998.

[10] G. Walsham. *Interpreting information systems in organizations.* Wiley, 1993.

[11] P.R. Warren and M. Viljoen. Design patterns for user interfaces. In *Proceedings of the SAICSIT conference, Gordons Bay.*, 1998.

[12] J.L. Wessson. Integrating hci with the software development process. In *BitWorld'99, Cape Town*, 1999.