CASE Tools and the Human Computer Interface: Implications for designers.

Alistair Sutcliffe

Dept of Business Computing Systems,
City University
Northampton Square,
London EC1V 0HB,
U.K.

Summary

Human computer interaction (HCI) is a subject which software developers can ill afford to ignore, yet there is little evidence that software engineers HCI seriously. The paper sets out to describe the software engineering and HCI issues which have to be addressed, in a comparative framework of the development life cycle. From this the necessary conceptual models produced by specification methods are defined. The requirements of CASE tool for constructing conceptual models to support HCI and SE methods are outlined and the ability of CASE environments to address HCI issues is reviewed. The concluding contention is that current CASE tools, combined with HCI training for software developers could deliver results in the short term, although to encourage HCI expertise in software design in the longer term necessitates development of knowledge based HCI CASE tools.

Keywords: Human computer interaction, software engineering, CASE-tools, methods

1. Introduction

Awareness of the human computer interface as an issue in software development has increased in recent years, design of interactive software according to human factors principles is still infrequent (Gould 1987). Software developers may criticise the human computer interaction (HCI) community for not producing practical methods and tools for specification and design of interactive software. However, methods have been proposed which integrate HCI with existing software engineering practices, e.g. SSADM (Damodaran et al 1988) and JSD (Sutcliffe 1988a) and the User System Engineering method, a HCI method based on Structured Analysis techniques, does have tool support (Wasserman 1984, Wasserman et al 1987). To deliver more usable systems, HCI authors need to integrate their findings within software engineering and likewise, CASE tool developers and software development method authors should support HCI principles and practices.

This paper sets out to describe the HCI issues which should be addresed during system development and then reviews how current CASE tools may be used to partially fullfil those requirements. As CASE tools are frequently sold as being configurable to support method independent or toolkit type development, it follows that these tools should be a capable of addressing a wide variety of system specification issues, including the human computer interface.

2. Framework of HCI Issues.

Human computer interaction embraces many issues, some of which relate directly to specification of interactive software, while others concern human organisation, operational procedures and workplace ergonomics. A survey of some of these issues and how they related to HCI and SE methods can be found in Sutcliffe (1988b, 1989) and more coverage of HCI issues is given by Shneiderman (1987).

HCI has produced few methods which provide good coverage of software development (Sutcliffe 1989), although there are a plethora of techniques with narrower aims, such as theoretical cognitive analysis of interaction, evaluation of the cognitive complexity of user system dialogues, and specifying the user's knowledge necessary to carry out a task (see reviews by Simon 1988, Murray 1987, Whitefield 1987, Johnson and Johnson 1988). In spite of this incoherent picture a consensus of basic requirements of HCI specification and design can be established by examining the commonalities in these different approaches.

A comparative framework of software engineering and human computer interaction is presented in Figure 1. This shows activities in the two disciplines grouped according to the scale of investigation from whole organisations, to part of a system concerning one person and then activities within systems. This classification is intended to draw out the activity phases found in methods. In software engineering the activities mirror the classic life cycle of development. Although this cycle is also present in HCI no comprehensive methods exist, instead techniques have been proposed to address specific topics. Some of these techniques are enumerated in fig 2. Since space precludes a review of these techniques the reader is referred to Sutcliffe (1989), Simon(1988) and Wilson et al(1987) for further descriptions. However, a brief summary of the similarities and differences of HCI techniques and software engineering methods, as embodied in typical examples, e.g. JSD (Jackson 1983, Sutcliffe 1988c), SSADM (Longworth and Nicholls 1987), will be given. Figure 2 illustrates a comparison of selected HCI and SE methods, using the software development life cycle and development issues as dimensions to show coverage by methods.

At the scale of organisation analysis some software engineering methods espouse strategic planning and enterprise analysis, e.g. Information Engineering (Macdonald 1986). HCI methods follow a similar path in goal oriented analysis of an organisation's activity but more emphasis is placed on people, their abilities, and expectations. Representative methods are Open Systems (Eason 1989), and User Skills-Task Match (Fowler et al 1988). Both methods address the person level of activity analysis, but do not follow specification through into

software design. At this level of analysis the Soft Systems method group, SSM (Checkland 1981) and Ethics (Mumford 1983), lies between the SE and HCI communities and both "hard" software engineering methods and HCI methods have borrowed soft system's concepts.

Software engineering methods generally do not have the concept of individual users as a unit of analysis, instead requirements analysis focusses on functionality and users' objectives. HCI methods, in contrast, pay more attention to individual users, for three principal reasons. First is matching system functionality, or tasks, to people to ensure that their work has the correct task mix for their abilities. User-task matching employs many heuristics, see for instance Bailey 1982, although more methodical guidance is given in USTM and Open Systems. The second reason is to model human mental processes for task operation to predict human errors and cognitive limitations on task performance. Typical methods in this class are GOMS and CCT. The third purpose is to obtain a model of the user's characteristics and knowledge about a system. This HCI activity is collectively termed user modelling, and embraces a diverse techniques and objectives, many of which have little relevance to systems development (see Murray 1987, Whitefield 1987 for reviews). The activities more relevant to system development are task modelling and analysis of user abilities and characteristics as profiles of user groups.

At the scale of activity analysis SE and HCI methods show considerable similarities. SE methods frequently use a goal oriented hierarchical decomposition approach (e.g. SA/AD De Marco 1978), and HCI task analysis methods follow the same approach (Bailey 1982), although it is not always explicit. Some HCI methods add modelling levels to the hierarchical decomposition as in the Command Language Grammar of Moran(1981), while others take a knowledge based approach creating models of user knowledge about tasks and the system (TKS, Johnson et al 1988). In spite of the similarities in approach, HCI methods differ in attempting to address cognitive issues of human information processing, which are ignored by SE methods. Here the endeavour is to ensure that tasks do not impose demands on users that exceed limitations of memory and mental abilities.

HCI methods utilise simple metrics of counting rules and state variables within task sequences to estimate task complexity (Cognitive Complexity Theory -Kieras and Polson 1985), however, techniques are still crude and design deliverables ill defined. More psychologically valid approaches employ knowledge based systems but these are a long way from being practical design advisors (Barnard et al 1988). The follow through of HCI analysis methods to later life cycle stages and the link between analysis and design is poor (Sharrat 1987, Knowles 1988, Sutcliffe 1989). The more comprehensive examples either produce specifications in knowledge based formalisms (CLG Moran 1981, TKS-Johnson et al 1988) making integration within SE specifications difficult, else transition network diagrams are used to model user-system dialogues but no integration with functional processing is given (Knowles 1988).

Clearly, HCI techniques do not exhibit the maturity to become utilisable methods in software engineering, although there is a considerably body of utilisable knowledge and techniques . Software engineering methods can not afford to ignore HCI issues, as many system development problems can be ascribed to lack of human factors input into the software design process (Gould 1987). Much of the HCI input to design in is the form of guidelines (Smith and Mosier 1986) which need to be interpreted in the context of each application. While this knowledge can be employed effectively by software engineers, to automate this support for the development of interface software would require an intelligent HCI designer's assistant. This may realisable when the present HCI knowledge based models of interaction (Barnard et al 1988) are integrated with the next generation of intelligent development support environments (e.g. Requirements Apprentice Rich et al 1987). However, following the lessons in software engineering, automated support is a prerequisite for practice of structured methods and techniques. HCI specification produces conceptual models, as do software engineering methods, hence it is appropriate to examine how CASE tools could support the development of HCI models.

3. Conceptual models and CASE Tools.

One of the main functions of CASE tools is to record the conceptual models created by development methods, usually in the form of diagrams or structured texts. CASE environments reflect the origins as either generic tool-sets or method linked support tools. Generic toolsets provide configurable editors which can create diagrams for different methods (e.g. Eclipse-Elliston 1989), whereas method linked tools are tailored to a specific method's notation and sometimes including guidance to help analysts follow the method's recommended development steps, e.g. Speedbuilder, LBMS Automate, IEW, IEF etc.

While all SE method ignore development of the human computer interface, apart from simplistic models of screens and interactive dialogues, their tools can be used for HCI purposes. First some explanation is required of the models produced by HCI within the design process. Fig 3 shows a comparison of conceptual models produced by software engineering methods and HCI. Not all SE methods produce all models as found in comparative studies of development methods (Loucopoulos et al 1987, Olle et al 1988). Likewise the complement of HCI models is not complete, and indeed some areas -interface displays- have no formalised models at all. However, the more formalised models are amendable to computerised support, and this can be effected by the current generation of CASE tools. The following section examines how both method dependant and open CASE toolsets can support the different model classes.

4. Tool Support for HCI models

4.1 Organisation/Enterprise models

HCI organisation models are generally informal and therefore may not be amenable for automated support. However HCI methods attempt to link analysis of organisations and users with system functionality to specify optimal matching of functionality to people and organisational units (e.g. USTM- Fowler et al 1988). This activity requires list handlers and grid/matrix tools to cross reference variables. Few such tools are provided by either method specific or open CASE environments even though developing such tools is not complex. A further justification for grid/matrix type tools can be found in Soft Systems methodologies which undertake similar cross referencing/trade off type analyses.

List handler tools are also required to support user models. While a bewildering number of user models have been propounded in HCI, probably the most relevant to software designers are simple list/grid based models characterising user abilities, knowledge of the system, and expected frequency of usage. Other tools may be required for this level of analysis to support workload planning. For instance job activity analysis, as proposed by Damodaran et al (1988) is a temporal analysis of task effort and the necessary manpower. Here generic time series analysis tools are needed, and Gnatt chart diagrammers could be made configurable for this sort of activity.

While configurable tools could be provided to support the above modelling activities, it is doubtful that such tools would be useful to software engineers without the HCI knowledge necessary to interpret results and make trade off decisions. Consequently HCI methodical guidance would have to be embedded within the CASE environment. Alternatively HCI training should be given to software developers, in which scenario, generic CASE tools could be used for documentation support.

4.2 Task/Function models

Task models can be developed with the current generation of CASE diagramming tools. Data flow diagrams can describe task networks in the same way as they are used for functional specification, and a functional/task specification using LBMS's Automate is illustrated in fig 4. However, DFD task descriptions record only the functional aspects of tasks. Tools are also

required to support cognitive analysis. One possibility would be to use specification held in data dictionaries to give counts of state variables, data items input and processing rules associated with task actions, these could be used to estimate task complexity following Kieras and Polson's method. Simple analyses list handling tools linked into specification data dictionaries would suffice.

Task functionality can be also be recorded in event models such as the Process Structure Diagram of Jackson System Development (JSD). Fig 5 depicts task specification as a Process structure diagram, created by the PDF tool, which formed part of an design study integrating HCI specification within JSD (Sutcliffe 1988a). One appeal of the JSD event specification is that it provides an easy transformation path from task specification to dialogue and process design; an important advantage if the full potential of CASE tools is to be realised by automatically generating interactive software code from specifications. This consideration leads to detailed HCI design models.

4.3 Process/Dialogue design models.

HCI methods have adopted two principal approaches to dialogues specification: adaptations of state transition diagrams (e.g. GTNs-Kieras and Polson 1985) and command grammars, a well known example being the Command Language Grammar of Moran (1981). Some Software Engineering methods do pay cursory attention to dialogue design in terms of high level dialogue maps (see LBMS-SDM version 3), although their respective CASE tools do not currently support this specification activity.

Several SE methods use state transition models to specify event dependencies in processing (e.g. SA/SD-Ward and Mellor 1986; IE -Macdonald 1986) and these method vendors do supply appropriate CASE tools. Unfortunately these tools assume fairly simple event models and have cumbersome notations unsuitable for complex models with a large number of states and transitions. To be tractable, interactive dialogues have to be specified in levelled hierarchies, also notations should be able to express concurrency, as in two dialogues in separate windows. Current CASE tools, either method specific or open do not provide adequate tools for dialogue specification, though to be fair most of HCI derived dialogue diagrammers do not support concurrency.

Dialogue specification diagrammers based on state transition formalisms, are frequently implemented in the nearest HCI equivalent of CASE tools, User Interface Management Systems (UIMS). These software environments aim to specify and in some cases automatically generate interactive software, which then communicates with the application software via parameters (Cockton 1987). Regrettably creators of UIMS have ignored mainstream software engineering methods and propound the philosophy that user interface and applications software can, and should, be developed separately. This approach has many theoretical difficulties and to date UIMS have had an even worse track record for real-life application than CASE tools.

One attempt to synthesise dialogue modelling with standard SE methods and tools which has met with some success is the User System Engineering method of Wasserman (1984). This combines data flow diagramming techniques of "top down" SA/SD (after De Marco 1978) with state transition modelling of interactive dialogues in an "outside in" specification. Although the method does not detail how these two approaches are reconciled, it does have extensive tool support and automatic code generation from diagram specifications (Wasserman et al 1987). CASE tool developers could profitably import state transition notation from HCI for the purpose of dialogue specification. Models of screen layout and presentation design, however, are too poorly formed even within HCI to merit tool support in their current state of development.

5. Conclusions

To be truly effective CASE tools may require methodical guidance to be built in, although this proposition is still a matter of active debate within the software engineering community. Advanced CASE tools encapsulate method and domain knowledge to provide intelligent assistance for the analyst (e.g. ASPIS Peitri et al 1987, Analyst Apprentice, Rich et al 1987). Ultimately HCI knowledge should be added to intelligent design environments to guide analyst in HCI design procedures and assist with more specialised cognitive aspects of interface design. While this will entail a sophisticated expert designers assistant based on theoretically sound psychological models of human computer interaction, as demonstrated in prototype form by Barnard et al 1988, more modest systems could be developed in the short term to codify HCI guidelines and design expertise.

Although it is acknowledged that human computer interface issues are important, little attention is paid to this aspect of design by software developers. The HCI community are primarily responsible for their own failure by inventing a plethora of fragmented techniques, and not integrating human factors principle and practices with those of software engineering. The HUFIT project is building support tools for task analysis based on HCI methods (Talyor 1988, Zeigler 1988), although repeating the mistake of providing stand alone tools rather than integrating HCI specification with standard software engineering specification methods.

Indeed the only notable exception to this rule is the USE method of Wasserman (1984) and even this has enjoyed limited success and can be criticised as addressing a minimum of HCI issues. More recently HCI authors have been turning their attention to integrated HCI-SE methods (Damodaran et al 1988, Sutcliffe 1988a). Experience in software engineering has taught that tool support is essential for method acceptance. The investigation reported in this paper demonstrates that it is possible to use current CASE tools for a considerable degree of HCI specification; so with successful HCI method education, HCI design could be practiced by software engineers with current technology.

CASE tools, in particular the vendors of open tool sets, should be more aware of the requirements for support tools, not only from HCI, but also from soft systems methods. Clearly there is a need for simple tools such as list handlers and grid/matrix analysers to cover analysis early in the life cycle and for organisational issues. While simple list handlers are provided by some method dependant CASE tools (e.g. Speedbuilder), these are invariably tied into the method's procedure.

To be successful the next generation of open toolsets will have to be flexible enough to allow configuration not only of specification recording tools but also knowledge bases holding procedural "how to do it" knowledge, design heuristics and rules. These requirements will be necessary to satisfy the demands of HCI and non functional aspects of system development. The advice to method authors, and inter alia, method linked toolsets, is that integration of HCI specification is a issue which can not be ignored. Tool support, particularly of dialogue design needs to be provided, before methods can claim to cover all the major issues of systems development.

Acknowledgements and Trade marks

Speedbuilder is a trademark of Michael Jackson Systems Ltd, London.
PDF is copyright of the United Kingdom Atomic Energy Authority.
IEF (Information engineering facility) is a trade mark of James Martin Associates.
IEW (Information Engineering Workbench) is a trade mark of Arthur Young Ltd.
Automate is a trade mark of LBMS, Learmonth and Burchett Management Systems.

References

Bailey R.W., (1982); Human performance engineering: A guide to systems designers. Prentice Hall, N.J.

Barnard P., Wilson, M. and Maclean A.; (1988); Approximate modelling of cognitive activity with an expert system: A theory-based strategy for developing an interactive design tool. The Computer Journal, Vol 31(5), pp 445-456

Card, S.K., Moran T.P. and Newell A.; (1983); The Psychology of human computer interaction, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Checkland, P.; (1981); Systems thinking, systems practice
J. Wiley.

Cockton G.; (1987); A new model for separable interactive systems.In Interact-87; Eds Bullinger H-J and Shackel B, pp 1033-1037, North Holland.

De Marco T., (1978); Structured analysis and system specification. Yourdon press, N.Y.

Damodaran L., Ip, K and Beck, M.; (1988); Integrating human factors principles into structured design methodology: A case study in the U.K. civil service.
In Information technology for organisational systems; Eds Bullinger H.J. et al,pp 235-241; Elsevier.

Eason K.D.; (1989); Information technology and organisational change., Taylor and Francis, London.

Elliston A.; (1988); Eclipse version 2, functional specification; Software Sciences, Handforth, Cheshire, U.K.

Fowler, C, Kirby, M, Macauley, L. and Hutt A.; (1988); User skills and task match (USTM): A human factors based methodology for determining product requirements.
In Proceedings of the 4th Alvey Conference, Swansea, Wales.

Gould J.D.; (1987); How to design usable systems; eds Bullinger H-J and Shackel B.; Proceedings Interact-87; North Holland.

Jackson M.A.; (1983); System Development, Prentice Hall, London.

Johnson P. and Johnson H.; (1988); Practical and theoretical aspects of human computer interaction. In The World yearbook of fifth generation computing research; Ed Aleksander I.; Kogan Paul, London.

Johnson P, Johnson H., Waddington R. and Shouls A.; (1988); Task related knowledge structures: Analysis, modelling and application.
In People and Computers IV (HCI-88); Eds Jones D.M. and Winder R.,pp 35-62, Cambridge Univ Press.

Kieras D. and Polson P.G.; (1985); An approach to the formal analysis of user complexity. Int. J. Man Machine Studies; Vol 22; pp 365-394.

Knowles C.; (1988); Can cognitive complexity theory (CCT) produce an adequate measure of system usability?
In People and Computers IV (HCI-88); Eds Jones D.M. and Winder R.; pp 291-308; Cambridge Univ Press.

Longworth G. and Nicholls, D.; (1987); The SSADM manual, NCC Publications.

Loucopoulos P., Black, W.J., Sutcliffe, A.G. and Layzell P.; (1987); Towards a unified view of system development methods.
Int J. of Information Management, Vol 7(4)

MacDonald I. G.; (1986); Information Engineering - An Improved, Automatable Methodology for the Design of Data Sharing Systems. In Proc IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the practice; Eds Olle T.W., Sol, H.G. and Verrijn Stuart, A.A.; North Holland.

Moran T.P.; (1981); The Command Language Grammar: a representation for the user interface of interactive systems.
Int. J. Man Machine Studies; Vol 15; pp 3-50.

Mumford, E.; (1983); Designing participatively. Manchester Business School Publications.

Murray D.M.; (1987); Embedded user models.
In Interact-87; Eds Bullinger H-J and Shackel B, pp 229-236
North Holland.

Olle T.W., Sol, H.G. and Verrijn Stuart, A.A.; Eds; (1986);
CRIS-3. IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies: Improving the practice. North Holland.

Pietri, F, Puncello, P.P., Torrigani, P, Casale, G., Innocenti, M.D., Ferrari, G., Pacini, G, and Turini, F.; (1987). ASPIS: A knowledge based environment for software development. In Esprit-87: Achievements and Impact, pp 375-391.

Rich C., Waters R.C., and Reubenstein H.B.; (1987); Towards a requirements apprentice. In Proceedings 4th Int. Conf. on Software Specification and Design; Monterey, California.

Sharrat B.; (1987); Top down interactive system design: some lessons learnt from using command language grammar.
In Interact-87; Eds Bullinger H-J and Shackel B, pp 395-402,
North Holland.

Shneiderman B. (1987); Designing the user interface. Addison Wesley, Reading.

Simon T.; (1988); Analysing the scope of cognitive models in human computer interaction: A trade off approach.
In People and Computers IV (HCI-88); Eds Jones D.M. and Winder R.,pp 79-96, Cambridge Univ Press.

Smith S. and Mosier J.N.; (1986); Design guidelines for user-system interface software. Mitre Corp, Bedford, MA.

Sutcliffe A.G.; (1988a); Some experiences in integrating specification of human computer interaction within a structured system development method.
In People and Computers IV (HCI-88); Eds Jones D.M. and Winder R.; pp 145-160, Cambridge Univ Press.

Sutcliffe A.G.; (1988b); Human computer interface design. Macmillan, London.

Sutcliffe A.G.; (1988c); Jackson System Development. Prentice Hall, London.

Sutcliffe A.G.; (1989); Task analysis, Systems analysis
and design: Symbiosis or synthesis ?
Interacting with Computers, Vol 1 pp

Taylor B.C.; (1988); Tools for Human factors in IT product design. Summary paper Esprit Project 385 HUFIT.

Ward P.T. and Mellor S.J. (1986); Structured design for real time systems. Yourdon Press, N.J.

Wasserman A.I., Pircher, P.A., Shewmake D.T. and Kersten, M.L. (1987); Developing interactive information system with the User Software Engineering methodology. In Readings in Human Computer Interaction: A multidisciplinary approach; Eds Baecker R.M. and Buxton W.A.S; Morgan Kaufman.

Whitefield A; (1987); Models in human computer interaction: a classification with special reference to their uses in design.
In Interact-87; Eds Bullinger H-J and Shackel B, pp 57-64
North Holland.

Wilson M.D., Barnard, P.J., and Maclean, A.; (1986); Task analysis in human computer interaction; Hursley Human Factors Laboratory Report HF 122, IBM (UK) Ltd.

Zeigler J.; (1988); Tools for the design of integrated Interfaces.
Summary paper, Esprit Project 385 HUFIT.

Fig 1

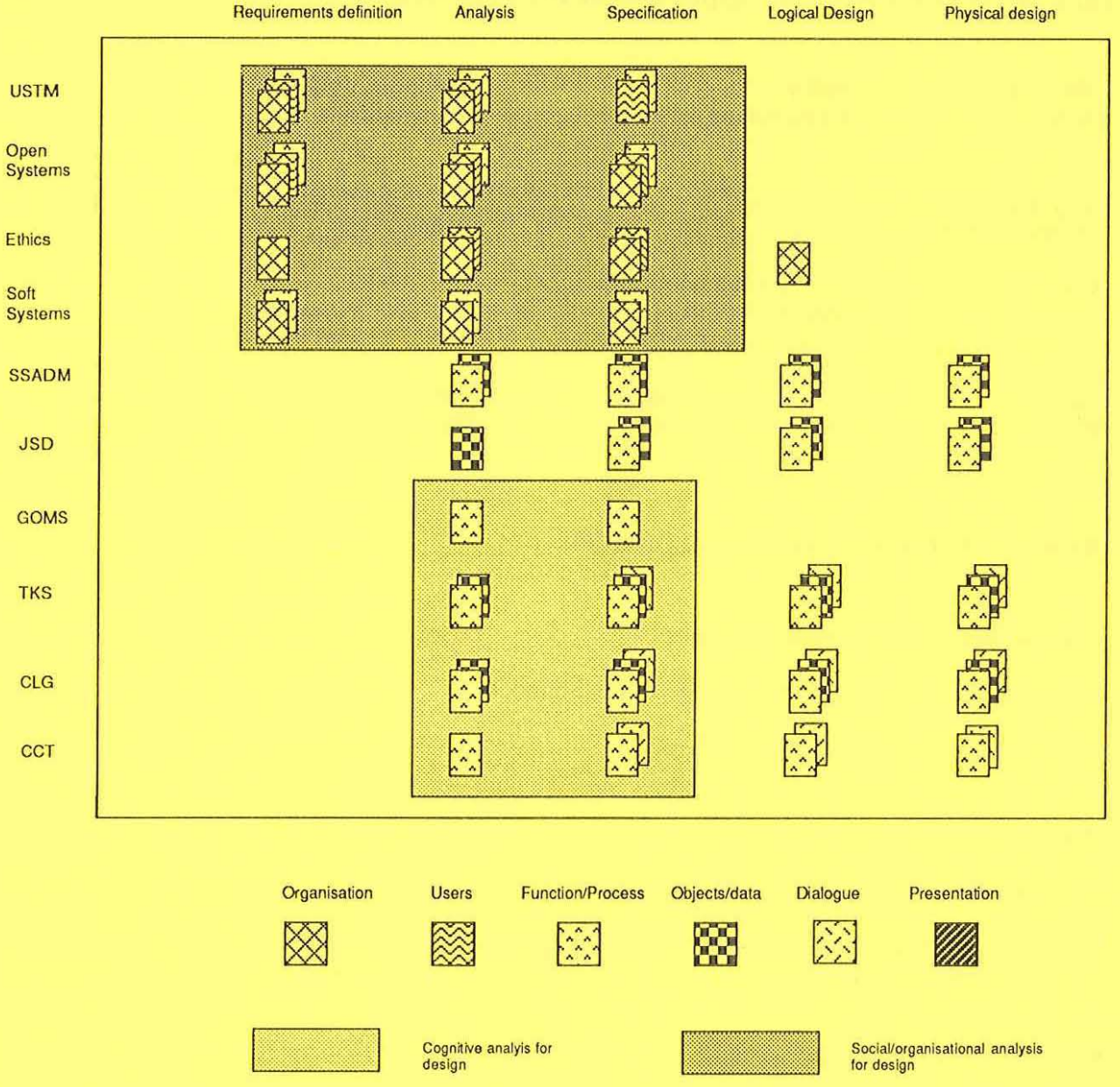
Development Issues in Software Engineering and Human Computer Interaction

Unit of Analysis	Software Engineering	Human-Computer Interaction
Organisation /whole system	Strategic Planning	Human Activity/ organisation analysis
Individual person/job/ sub system	Requirements analysis Conceptual modelling	Task/job/work analysis User modelling
Activity sub-system	Functional analysis Data analysis	Task analysis User modelling
Procedure /Process	Process/data structure specification	Dialogue/display specification

Classification of issues according to analysis activity is correlated to an extent with the development life cycle, thus organisation analysis occurs in early phases while processes are subject of later analysis/design stages.

The data related software engineering activities (Data models, etc) have been omitted as there is no valid comparision in the HCI field.

Fig 2. Method analysis by coverage of development life cycle and systems design issues



The life cycle categories are requirements definition, early analysis to decide the scope of investigation and appropriate users; descriptive analysis of the current system and its users; specification of what the system should do in terms of processes and organisation; design of how the system and its users cary out tasks in logical terms; and finally physical design taking in environmental constraints, budget, hardware, etc. The final stages of implementation, testing and maintenance have been omittted.

Key to methods: USTM - User Skills Task Match (Fowler et al 1988); Open Systems (Eason 1989); Ethics (Mumford 1983); Soft Systems (Checkland 1981); SSADM - Structured Systems Analysis and Design Method (Longworth and Nicholls 1987); JSD - Jackson System Development (Jackson 1983); GOMS- Goals, Operators, Methods, Selection rules (Card et al 1983); TKS- Task Knowledge Structures (Johnson et al 1988); CLG- Command Language Grammar (Moran 1981); CCT- Cognitive Complexity Theory (Kieras and Polson 1985).

Fig 3

Conceptual models produced by Software Engineering and Human Computer Interaction and CASE support tools.

(a) Conceptual models in approximate life cycle order.

Life cycle phase phase	Software Engineering	Human-Computer Interaction
Early analysis strategic planning	Enterprise model	Organisation/User model
Analysis	Functional model Data/object model Event/object model	Task model User profile model
Logical design	Process specification Data structure spec	Dialogue specification Screen designs

(b) Support tools for conceptual models

List handlers; Matrix-grid analysers

Enterprise, Organisation/User model; User profile model

User-function matrix.....function lists

Diagram editors

Functional model, Data/object model, Event/object model, Task model

Data-flow .....Entity relationship...Entity life history....Task-info flow..
Dialogue transition network

Structured text editors

Process specification, Dialogue specification, Data structure spec

Structured English....Dialogue grammars.....Data dictionary

Conceptual models are grouped according to support tool types. Examples of modelling notations are listed in italic.

Fig 4. Initial task specification recorded as a Data Flow diagram using LBMS-Automate.

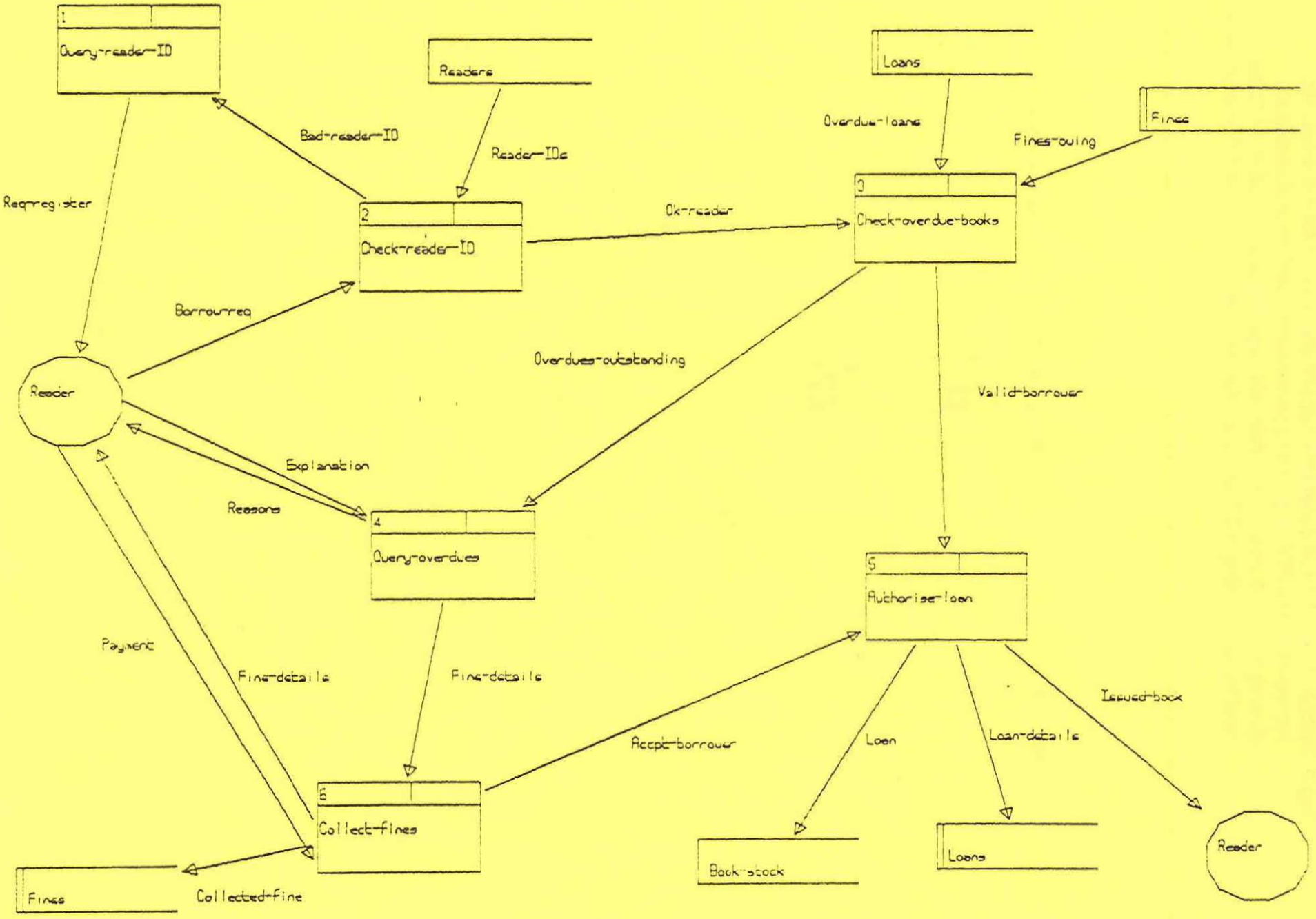


Fig 5. Specification of a computer task support process for loan authorisation recorded by the PDF tool. This specification could be elaborated to directly generate interactive code, although the sophistication of screen handling and dialogue I-O is dependant on the implementation environment.