Evaluation Techniques: Exploring the Intersection of HCI and Software Engineering

Joëlle Coutaz
Laboratoire de Génie Informatique, IMAG
B.P. 53, 38041 Grenoble Cedex 9, France
Tel. +33 76 51 48 54, fax: +33 76 44 66 75, e-mail: Joelle.Coutaz@imag.fr

Abstract

This article discusses the improvement in usability of interactive systems through the transfer of methods, tools and concepts between software engineering and HCI. It explores commonalities between these two communities, and shows how usability testing can be incorporated within different models of the software development process. Techniques and methods for usability testing differ widely depending on the development stage, resources available, and the competence of the assessors. This paper shows that each usability technique has a natural anchor point within the software development process. However, in practice such techniques are not exploited by the software community. This article suggests complementary ways of improving the situation.

Keywords

Usability engineering, usability testing, evaluation techniques, software quality assurance.

1. Introduction

Software Engineering has provided the Human-Computer Interaction (HCI) community with tools and techniques for the design and implementation of interactive systems. Typical examples include User Interface Management Systems (UIMS), toolkits and architectural models. UIMS's provide a cost-effective way of building prototypes and thus support iterative refinement of user interfaces. Toolkits define a specific "look and feel" that improves consistency across systems. Multi-agent architectural models make possible multithread dialogues that match the opportunistic behavior of users. Although these models and tools improve usability in their own way, they do not explicitly address its evaluation. Usability testing is an important activity in HCI that has been underestimated in software quality assurance.

Software quality assurance is an active area of research in software engineering. It has been claimed that "the 1990's will be the quality era, in which software quality is quantified and brought to the center of the development process" [BM91]. Software quality covers three closely related activities: quality goal setting, quality control, and quality evaluation. Quality goal setting identifies the desirable properties that a product

and its development process should satisfy. Quality control is concerned with the methods and tools used to check quality goals. Quality evaluation assesses the soundness of the software product and its development process based on some measurement of the desirable properties.

Software quality goals are fundamental. Unfortunately, there are no universal formal definitions of quality goals on which engineers can safely rely. Instead, software quality goals cover a complex mix of properties that vary across applications, customers, and software engineers. In order to clarify this complexity, Boehm and then McCall have proposed a useful classification of factors that affect software quality [McC77]. More recently, the IEEE [Iee87] and the AFCIQ (part of the French association for standards) have published recommendations directly inspired from McCall's work [Afc89]. All of these proposals show true considerations for the user of the final product but these concerns are expressed laconically in elusive terms such as "usability" and "training".

This paper will use "usability testing" as an anchor point between HCI and software engineering evaluation techniques. After an overview of the evaluation techniques developed in HCI, we analyse how design for usability fits within software engineering development processes. Having established this framework, we focus the discussion on how usability testing can contribute to two specific steps of the development process: requirements analysis and system design. We note that HCI tools, methods and notations are available but not incorporated into software engineering practice. We conclude with suggestions on how to improve the situation.

2. An Overview of Evaluation Techniques in HCI

As shown in figure 1, evaluation techniques developed in HCI may be predictive or experimental. Predictive methods are applicable at the early stage of the development process of a system. They do not require any system implementation, nor do they need real users. Experimental techniques on the other hand, rely on the existence of a physical apparatus ranging from a mock-up of the real system to an implementation of a running prototype.

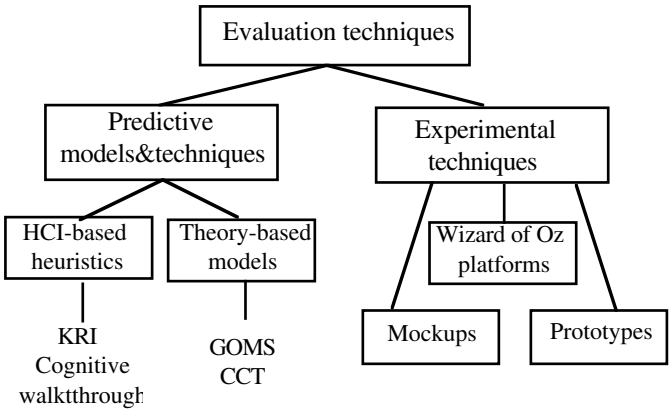

Fig. 1. An overview of HCI evaluation techniques.

2.1. Predictive Techniques

In general, predictive techniques are theory-based. For example, GOMS [CMN83] and its related models such as the Cognitive Complexity Theory (CCT) [KP85], rely on an explicit hierarchical decomposition of the user's tasks. This static representation is supposed to model the user's plan for accomplishing a particular task with a particular system. The original GOMS is a purely analytic model of errorless performance: it is able to predict the time required to accomplish a task without errors. Recently, GOMS has been extended to predict errors due to cognitive overload [LMO89]. CCT, on the other hand, is useful for comparing several designs in terms of learnability and knowledge transfer.

At the opposite of the static task descriptions of GOMS and CCT is the PUM dynamic model of the user's plan [YW90]. PUM (Programmable User Model) predicts errors through a programmable cognitive architecture implemented on top of SOAR. The designer specifies the knowledge that the user needs to accomplish a particular task. This description, which includes domain knowledge as well as knowledge about the user interface, is compiled in terms of rules. These rules represent the user's ability to accomplish the task with the specified user interface. Based on this knowledge, the PUM cognitive architecture tries to elaborate a plan. If no plan can be generated, then the designer is notified of a potential usability or learnability problem.

The theoretical model "Interacting Cognitive Subsystems" (ICS) can be used to predict cognitive overload in terms of the mental resources or processes involved in accomplishing a particular task with a specific system [Bar87]. ICS subdivides the human information processing system into a set of specialized subsystems. The sensory subsystems transform sensed data into specific mental codes that represent the structure and content of incoming data. These representations are then handled by subsystems that are specialized in the processing of higher-level representations: the "morphonolexical subsystem" for processing the surface structure of language, the "object subsystem" for processing visuospatial structures, and the "propositional and implicational subsystems" for more abstract and conceptual representations. The output of these higher representational subsystems are directed to the effector subsystems (articulatory and limb). Cognitive overload will be detected if, for example, task accomplishment requires a large number of loops within the high level representational subsystems.

Predictive evaluation techniques may also be based on HCI heuristics. Typically, the assessor looks for properties in the user interface design that he/she knows from experience lead to usability or learnability problems. Such knowledge, exemplified by Smith & Mosier's work, may be embedded in an expert system such as KRI [LN90]. KRI is able to detect anomalies from a formal description of the user interface. However, only the lexical and syntactic levels of the interaction are covered by the software critic. Task modelling and any high level cognitive activity are not supported by the KRI model.

Assessing a design through HCI heuristics is a difficult task. The task-based "cognitive walkthrough" method proposed by Lewis et al. provides a useful framework for extracting evaluation guidance from a formal theory of human-computer interaction [LPW+90]. It consists of "a list of theoretically motivated questions about the system"

such as how will the user access the description of an action or how will the user associate a description to an action?

In summary, the main benefit from predictive models and techniques is that they allow the evaluation of user interfaces at the design stage. A design can be improved before a costly implementation takes place. On the other hand, specifying data to a predictive model may be as time consuming as an implementation. As shown by Pollier [Pol91] and Nielsen [NM90], heuristic evaluation is incomplete. At least three assessors are necessary to discover a reasonable number of usability problems (i.e., half of the problems at best!). Predictions made by theoretical models are based on hypotheses, not on real data. Experimental techniques are intended to overcome these limitations.

2.2. Experimental Techniques

Experimental techniques deal with real data observed from real users accomplishing real tasks with a physical artifact. This artifact may include paper scenarios, mock-ups, computer system prototypes, or Wizard of Oz (WOz) platforms. With a WOz setting, "designers can illustrate how users will interact with yet-to-build software" [Mac88]. Through the observation of users' behavior, designers can identify users' needs when accomplishing a particular set of relevant tasks and evaluate the particular interface used to accomplish the tasks.

Most existing WOz systems have been primarily developed to study the usage of natural languages for retrieval information systems [DJ89] [DJA92]. Telephone information services such as telephone directories, flights or trains information and reservation services, have been an interesting field for experiments [FKP92]. Questions and answers are tape-recorded for later manual transcription and analysis. Other case studies involve databases or advisory systems interrogation [GS90] [JD88] [WS89] as well as dialogues with expert systems [Dia89]. Most of them aim at collecting vocabulary corpora in order to tune and augment the robustness of spoken or written natural language recognizers. The platform described in [HHH92] is an attempt to support the observation and capture of graphical direct manipulation combined to natural language. A recent paper [MGM93] describes Turvy, an intelligent agent simulated using a WOz, that can be taught using speech and direct manipulation. In our laboratory, we are building NEIMO, a multi-wizard generic WOz platform that automatically digitizes and records behavioral data ranging from low level physical actions to high level commands performed by a subject using a multimodal user interface [SC93].

Analysis from experimental methods are performed on real data, not on approximated values. This benefit is counterbalanced by the volume of behavioral data and the difficulty of identifying the appropriate parameters for the experiment. We believe that a WOz computer platform which automatically captures selected behavioral data, provides a good basis for the development of evaluation and design tools. It does rely however on a method. One of them is the influential "design for usability" approach.

3. Design for Usability and Software Development Processes

Design for usability is primarily user centered and iterative. From the software engineering perspective, "user centered" means that focus on the user should be incorporated at the very early stage of the software life cycle and carried out all the way through the development process. "Iterative" implies formal usability testing and redesign until usability goals are met. In other words, "design for usability" implies that software life cycles accept as many backtracks as necessary in a cost effective way. Typically, the waterfall model does not address these constraints [Roy70]. The V model MR84] is acceptable in terms of backtracking but explicit usability testing appears too late in the development process: although tests are devised for each step early in the development process, actual testing occurs once implementation has taken place. (See Section 9 for a more detailed description of the V model.)

A single late evaluation of usability is too risky in terms of resources and quality objectives. The literature witnesses too many cases where late usability testing results in substantial changes that are unfavorably received by the implementers or too costly to implement. Figure 2 highlights the increasing cost of finding mistakes later in the development cycle. According to a study performed by HP, costs to fix a defect ranges from a couple of dollars when detected in the specification stages to millions of dollars when detected after the system has been released. Clearly, early and continuous testing is a useful practice provided that it is performed in a cost effective way.

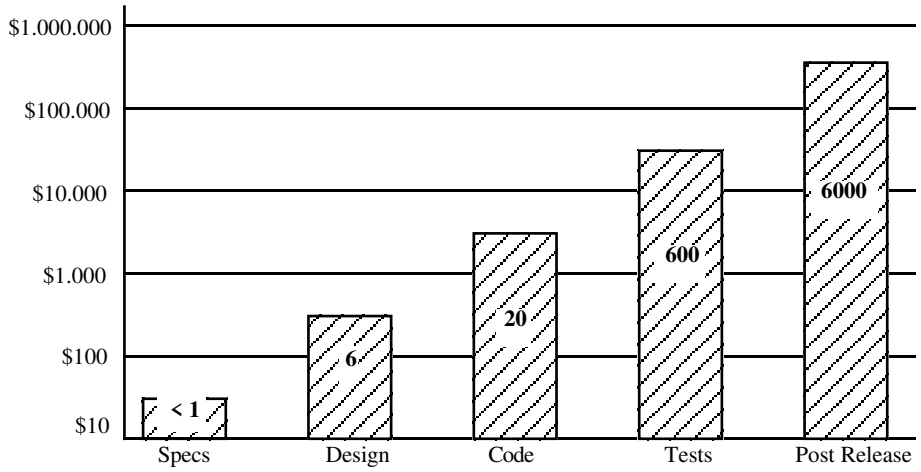

Fig. 2. Cost to fix defects (private communication from HP). Values within bar charts denote the number of hours used to fix the defect.

The spiral model offers an appropriate framework for our problem [Boe88]. In particular, one cycle of the spiral may include sub-spirals to conduct targeted usability tests on exploratory subparts of the system. User interface design can thus be progressively refined through successive and complementary tests, minimizing the replication of expensive experiments. This notion of refinement matches nicely the COSITUE model developed for user interface design [Ber93]. The difficulty however is to instantiate the appropriate phases for the particular case at hand and for each step, to

identify the appropriate usability tests. The identification of the phases relies on a risk and cost analysis that requires a lot of expertise in project management. In addition, it is too tempting for a software project manager to define the phases in relation to technical constraints, and to ignore or underestimate the importance of the user-centered principle.

The spiral model is certainly suitable for the development of prospective and innovative systems whose ultimate functions are unclear. It is less practical for well structured projects whose development is limited in time. In this case, software engineers would favor the V model easier to apply and more tractable. One way then to accommodate "design for usability" with the V model is to perform targeted and complementary usability tests at every step of the development process. Although the strict application of the V model advocates document validation and verification before proceeding to the next step, usability testing is poorly integrated in the early steps of the V model. Let us see how the situation could be improved for the various steps.

4. Usability Testing and Requirements Analysis

Requirements analysis is usually considered as the first step of the V model. It results in a number of documents, two of them, the requirements document and the quality plan, being of interest to our discussion.

4.1. Requirements Document

The requirements document describes and specifies the functional needs and constraints as well as the quality factors required by the client. The description provides an informal account of the purpose of the system whereas the specification describes the requirements in a non ambiguous way, possibly using formal notations such as SADT, HOOD, and OMT. Although these tools are valuable for the expression of functional requirements, none of them are aimed at describing users' tasks. They are system-biased not user-centered. DIANE [Bar88] is an exception. It is not surprising then that most functional requirements are based on system considerations, not on a preliminary task analysis. Task analysis is one activity in system design that helps designers to understand and model the task domain.

Our first recommendation applicable to the very first step of the V cycle would be to use task analysis as the foundation for functional specification and exploit tools such as MAD [PDS89], TKS [JMW+92] and UAN [HG92] to formally express the results of the analysis. In addition to usability improvement, software designers may find it useful to exploit the notions of "mandatory" and "optional" tasks, and make explicit task allocation (i.e., who is in charge of the task?). These attributes can be used as hints to define priorities within the software development plan.

The second concern of the requirements analysis is the identification of the quality factors that are relevant to the case at hand. Factors set the software quality goals from the client's perspective. Criteria are the practical means to achieve those goals from the implementor's perspective. The quality plan specifies the criteria that will be used to achieve quality goals.

4.2. Quality Plan Document

For each criteria, the quality plan should contain an informal definition showing the relationship of the criteria to a quality factor, the metrics adopted to assess the criteria, the method and/or the tools used for the measurement, the desired value for the metrics, and the range that is expected and acceptable for positive assessment. It is interesting to observe that Software Quality Assurance adopts similar principles to those advocated in usability engineering.

In usability engineering, "evaluation is an assessment of the conformity between a system's performance and its desired performance" [WWD91, p. 70]. Here, a system includes a user and a computer engaged in a set of tasks. Performance is expressed in terms of the quality of the task product and the resource costs necessary to accomplish the task. Resource costs cover both computer and human costs. Computer costs can be refined using software engineering criteria whereas human costs recruit from ergonomics and cognitive psychology knowledge. In [DJ89], Dowell and Long provide a useful framework that divides human costs into structural and behavioral costs. Both of them may be mental or physical. In turn, mental costs may be cognitive, conative or affective, "relating to knowledge, motivation, and emotion, respectively".

In [WWD91], Whitefield illustrates the notion of "desired performance" with a train ticket selling machine: "A specified desired performance might be that the correct ticket is indeed exchanged for the correct money (the required quality of the task product) and that resource costs are within some limits. Examples of such resource cost limits might be not more than two minutes spent in planning and decision-making (a cognitive behavioral costs); no increase in frustration (an affective behavioral cost); not more than one reference to a Help facility (a cognitve structural cost); no discomfort in entering the relevant data (a physical behavioral cost); and a computer response time of less than one second (a computer processing cost)".

Clearly, the notion of "desired performance" is what software engineers would call "usability goals". The "assessment" mentioned in Long and Whitefield's definition of usability evaluation involves a method to check the conformity of actual performance with the desired performance and a statement about the findings. The method depends on the development stage and the resources. It may include the use of predictive models or it may be based on user testing. In the latter case, a user test plan is elaborated that specifies the tested task and the scenario (such as instructions given to the user before the experiment), the hardware and software resources that are necessary to the experiment, profiles of users, number of users per profile, the schedule, etc. The statement reports observed performance in terms of the quantified metrics. It is also desirable that it describes the causes of the problems in order to take sounded corrective actions.

In summary, the method advocated in usability engineering is consistent with the principles of software quality assurance. The similarity goes even further when considering the difficulty in applying the method. One very first common difficulty is the definition of the reference situation. The second common feature is that criteria are not always quantifiable. Subjective criteria are the common fate to both software engineering and usability testing.

In software engineering we tend to express subjective criteria in terms of quantifiable metrics. For example, code readability may be evaluated against the ratio of the number of lines of comments over the total number of source code lines. Such approximation does not necessarily guarantee readability but it makes it possible to apply quantification in a consistent way. In human factor studies, "user satisfaction" is an important feature which must be rated in objective terms. For example, to evaluate the quality of error messages, one typically records the type and frequency of errors, the rate of help, time spent reading help messages, duration used for thinking, the successful correction of the error, etc. Unfortunately, these objective measures are not sufficient to assess the informative quality of the message content. According to Senach [Sen90], qualitative data such as users' comments obtained from "thinking aloud techniques" are more valuable.

The similarity of approaches and problems between HCI usability engineering and software quality assurance opens the way to their mutual integration. The pending problem is cross-discipline education. As long as "usability" will be loosely defined, the software community won't be able to address the appropriate quality goals for it. An attempt to refine "usability" in terms of more specific criteria such as observability, predictability and honesty, can be found in [ACN92].

System design is the second step of the V cycle where software engineering and HCI may collaborate towards improving usability.

5. Usability Testing and System Design

One relevant output of system design is the external specifications document. This document describes the system as it will be perceived by the user. In general, this description is expressed informally using natural language as well as screen dumps.

Usage at this stage of formal notations such as UAN [HG92] opens the way to automatic usability testing before implementation proceeds. To illustrate the potential of the approach, we consider one example of rules drawn from our own experience using UAN to describe the user interface of MATIS [NC93], a Multimodal Air Travel Information System. If the precondition for a task execution cannot be expressed in terms of perceivable features, then the "observability" criteria is broken: the user is not aware that the task can be performed.

Formal specification is one promising way that needs more research. As outlined above, it opens the way to automatic evaluation before implementation can occur. It also sets the foundations for the automatic generation of user interfaces including the capability of automatically recording behavioral data at various levels of abstraction ranging from physical actions such as mouse clicks, to high level concepts such as begining or finishing a task. By so doing, we bridge the gap between specification, implementation and ecological testing in a single powerful tool. The analysis of behavioral data can then be envisioned in an automatic way as a four step process: 1) description of the task model 2) acquisition of behavioral data, 3) identification of behavioral patterns, and 4) critic per se.

A task model defines the optimal way of performing the task in a particular context for a particular domain. It is a behavioral reference model. Recorded data, as those captured by NEIMO, reflect the effective behavior of the user performing a task in a quasi-realistic setting. As in the MRP technique [SH91]), behavioral patterns are repeated user actions that may reveal usability problems. The critic per se combines general heuristic HCI knowledge with data specific to the case at hand: it detects deviations of the behavioral patterns from the reference task model. These aspects are developed in [CSB93].

6. Conclusion

This article has discussed how HCI and software engineering may cooperate through usability testing during the main phases of the development process: requirements analysis, system design, and implementation. We have demonstrated that the spiral model is consistent with design for usability. We have shown how the V model, favored for its simplicity, can be arranged to support iterative design and incremental usability testing.

Techniques and methods for usability testing differ widely depending on the development stage, resources available, and the inclination and competence of the assessors. This paper shows that each usability technique has a natural anchor point within the software development process:

- Theory-based tools such as GOMS, CCT, PUM and ICS, should be favored in the early stages of the development process as predictive tools. One may also resort to them as explanation tools after implementation has occurred. As mentioned above, it is important to detect usability problems early in the development process. It is equally important to understand the causes of the defects in order to take corrective actions.

- HCI heuristics and cognitive walkthrough are applicable once external specifications have been produced. They are cheap to perform because they do not require actual user testing. On the other hand, they rely on the competence of the assessors.

- Experimental approaches based on the capture of behavioral data for well-targeted tests are favored once implementation has occurred. Implementation may be performed "by hand" from toolkits and application frameworks, or produced automatically via user interface generators.

In summary, HCI tools, methods and notations are available but are not sufficiently incorporated into software engineering practice. As software researchers and educators, we can improve this situation by 1) developing tools targeted at usability testing, 2) promoting cross disciplinary education: effective integration can only occur through the roots of the future "deciders".

Tooling can be directed at formal specifications or at observational experiments. Formal notations for external specifications can be used as predictive instruments based on theoretically motivated principles. This initiative is not new (e.g., Payne's, Green's,

and Reisner's work) but must be pursued. Formal notations for task analysis such as ADEPT [JWM+93] can be extended to support user interface generation. The advantage of this approach is to increase code conformity with the intended functionalities. The drawback is that user interface generation from conceptual descriptions is limited by the set of presentation objects available. This technique requires the integration of two additional facilities that are currently missing: openness to modification of details of the presentation, and instruments to perform observational experiments.

Observational experiments are unavoidable as long as predictive theories are limited in scope. Observations are currently performed within usability laboratories. These laboratories are well equipped with analog recording that requires time consuming analysis. We need to computerize this task by extending the current Wizard of Oz techniques, by automatically instrumenting user interfaces and by providing high level critics based on formal descriptions of the intended task model and of the user interface. By doing so, formal specifications join the camp of experimental approaches forming an integrated environment for engineering usability.

7. Acknowledgements

This work has been supported by project ESPRIT BR 7040 Amodeus II. This article was influenced by stimulating discussions with Nadine Odet from Hewlett-Packard, Grenoble. We thank the reviewers and James L. Crowley for comments and suggestions.

8. References

[ACN92] G. Abowd, J. Coutaz, L. Nigay
Structuring the Space of Interactive System Properties, IFIP TC2/WG2.7 *Working Conference on Engineering for Human Computer Interaction*, Elsevier Publ., Ellivuori, Finland, August, 1992.

[Afc89] AFCIQ, Section logicielle, Groupe planification et coûts, S/Groupe Plan Assurance Qualité Logiciel PAQL. Recommandations de Plan d'Assurance Qualité Logicielle; version V0 23/03/89, 1989.

[Bar87] P.J. Barnard, "Cognitive Resources and the Learning of Humnan-Computer Dialogs", in *Interfacing Thought, Cognitive Aspects of Human-Computer Interaction*, J.M. Carroll Ed., MIT Press Publ., 1987, pp.112-158.

[Bar88] M.F. Barthet, *Logiciels interactifs et ergonomie, modèles et méthodes de conception*, Editions Dunod Informatique, 1988.

[BM91] V. R. Basili and J. D. Musa. The Future of Engineering Software: A Management Perspective. *IEEE Computer*, **24**(9): 90-96, September 1991.

[Ber93] N.O. Bernsen, “Structuring Design Spaces”, in *Adjunct Proceedings of INTERCHI’93* Conference, 1993, pp. 211-212.

[Boe88] B.W. Boehm, “A spiral Model of Software Development and Enhancement”, IEEE Computer, May 1988.

[CMN83] S.K. Card, T.P. Moran & A. Newell, *The psychology of Human Computer Interaction*, Lawrence Erlbaum Associates, 1983.

[CSB93] J. Coutaz, D. Salber, S. Balbo, Towards the Automatic Evaluation of Multimodal User Interfaces, Special Issue “Intelligent User Interfaces”, *Knowledge-Based Systems*, 1993.

[DA89] Nils Dahlbäck and Arne Jönsson: "Empirical studies of discourse representations for natural language interfaces", *Fourth Conference of the European Chapter of the ACL* Proceedings, 1989, pp. 291-298.

[DJA92] N. Dahlbäck, A. Jönsson and L. Ahrenberg: "Wizard of Oz studies — why and how", Third Conference *on Applied Natural Language Processing*, Trento, Italy, 31 march—3 April 1992.

[Dia89] D. Diaper: "The Wizard's Apprentice: A Program to Help Analyse Natural Languages Dialogues", proceedings of the *fifth conference of the British Computer Society, Human-Computer Interaction Specialist Group*, University of Nottingham, 5-8 September 1989, pp. 231-243.

[DJ89] J. Dowell, J. Long, Towards a Conception for an Engineering Discipline of Human Factors, in *Ergonomics*, 32, 1513-1535.

[FKP92] J.-M. Francony, E. Kuijpers and Y. Polity: "Towards a methodology for Wizard of Oz experiments", Third *Conference on Applied Natural Language Processing*, Trento, Italy, 31 march—3 April 1992.

[GS90] R. Guindon and K. Schuldberg: "Grammatical and Ungrammatical Structures in User-Adviser Dialogues; Evidence For Sufficiency of Restricted Languages in Natural Language Interfaces to Advisory Systems". In proceedings of the *25th Annual Meeting of the Association for Computational Linguistics*, 1990.

[HHH92] M.L. Hammontree, J.J. Hendrickson, B.W. Hensley, “Integrated Data Capture and Analysis Tools for Research and Testing on Graphical User Interfaces”, in the *CHI’92 Conference* Proceedings, ACM Press Publ., 1992, pp. 431-432.

[HD92] R. Hartson, P.D. Gray, “Temporal Aspects of Tasks in the User Action Notation”, *Human-Computer Interaction*, Laurence Erlbaum, vol. 7, No 1, pp. 1-45, 1992.

[IEE87] IEEE Software Engineering Standards; IEEE Press, 1987.

[JMW+92] P. Johnson, P. Markopoulos, S. Wilson, J. Pycock, “Task based design: Mapping between user task models and user interface designs”, In Proceedings of *second workshop on Mental Models in HCI*, University of Cambridge, England, 1992.

[JWM+93] P. Johnson, S. Wilson, P. Markopoulos, J. Pycock, “ADEPT-Advanced Design Environment fror Prototyping With Task Models”, In adjunct Proceedings, *INTERCHI’93*, 1993, pp. 56-56.

[JD88] A. Jönsson and N. Dahlbäck: "Talking to A Computer is Not Like Talking To Your Best Friend", Proceedings of the *Scandinavian Conference on Artificial Intelligence '88*, pp. 53-68.

[KP85] D. Kieras, P.G. Polson “An Approach to the Formal Analysis of User Complexity”, *International Journal of Man-Machine Studies*, 22, 1985, pp. 365-394.

[LMO89] F.J. Lerch, M.M. Mantei, J.R. Olson, “Skilled financial planning: The Cost of Translating Ideas into Actions”, in Proceedings of *CHI’89*, ACM Press, 1989, pp. 121-126.

[LPW+90] C. Lewis, P. Polson, C. Wharton, J. Rieman, “Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces, in Proceedings of *CHI’90*, ACM Press, 1990, pp. 235-241.

[LN90] J. Löwgren & T. Nordqvist, “A Knowledge-Based Tool for User Interface Evaluation and its Integration in a UIMS”, *Human-Computer Interaction-INTERACT'90*, pp. 395-400.

[Mac88] Wendy E. Mackay: "Video: Data for Studying Human-Computer Interaction", *CHI '88*, 1988, pp. 133-137.

[MGM93] D. Maulsby, S. Greenberg, R. Mander: “Prototyping an Intelligent Agent through Wizard of Oz”, *InterCHI’93* Proceedings, pp. 277-284.

[McC77] J. McCall. *Factors in Software Quality*; General Electric Eds, 1977.

[MR84] J. McDermid & K. Ripkin, *Life Cycle Support in the ADA environment*, Cambridge University Press, 1984.

[NM90] J. Nielsen, R. Molich, “Heuristic Evaluation of User Interfaces”, in *CHI’90* Proceedings, ACM Press Publ., 1990, pp. 249-256.

[NC93] L. Nigay, J. Coutaz, “A design space for multimodal interfaces: concurrent processing and data fusion”. *INTERCHI’93* Proceedings, Amsterdam, May, 1993, pp. 172-178.

[Pol91] A. Pollier, “Evaluation d’une interface par des ergonomes : diagnostics et stratégies”, *Rapport de recherche INRIA* no 1391(1991).

[Roy70] W.W. Royce, “Managing the development of large software systems”, Proc. *WESTCON*, Calif., USA, 1970.

[SC93] D. Salber, J. Coutaz, “Applying the Wizard of Oz Technique to the Study of Multimodal Systems”. In Human Computer Interaction, *3rd International Conference EWHCI’93, East/West Human Computer Interaction*, Moscow. L. Bass, J. Gornostaaev, C. Unger Eds. Springer Verlag Publ., Lecture notes in Computer Science, Vol. 753, 1993.

[PDS89] C. Pierret-Goldreich, I. Delouis & D.L. Scapin, "Un Outil d'Acquisition et de Représentation des Taches Orienté-Objet", *rapport de recherche INRIA no 1063, Programme 8 : Communication Homme-Machine*, août 1989.

[Sen90] B. Senach, "Evaluation ergonomique des interfaces homme-machine : une revue de la littérature", *Rapport de recherche INRIA n° 1180, Programme 8, Communication Homme-Machine*, Mars 1990.

[SH91] A. Siochi & D. Hix, “A study of Computer-Supported User interface Evaluation Using Maximal Repeating Patern Analysis”, in Proceedings of the *CHI’91* Conference, ACM Press, pp. 301-305, 1991.

[WWD91] A. Whitefield, F. Wilson & J. Dowell, "A framework for human factors evaluation", *Behaviour and information technology*, vol. 10, no. 1, pp. 65-79, 1991.

[WS89] S. Whittaker and P. Stenton: "User Studies and the Design of Natural Language Systems", *Fourth Conference of the European Chapter of the ACL*, proceedings 291-8, 1989.

[YW90] R.M. Young & J. Whittington, “Interim Report on the Instruction Langage”, *AMODEUS Project Document: Deliverable D5*, ESPRIT Basic Research Action 3066, 11th August 1990.

9. Annex: The V life cycle development process

The V life cycle model structures the development process into a set of successive phases. Once a phase is validated, the software development can proceed to the next phase. Although there is no canonical decomposition, the set of phases includes:

- the requirements analysis (description of the functionality of the system as well as constraints and quality goals. These specifications must be approved by the customer),
- the system design (specification of the solution as perceived by the client, e.g., the external specifications),
- the design of the global software architecture (description of the main components of the system),
- the specification of the detailed architecture (refinement of the global architecture in terms close to implementation), and
- coding.

These phases form the down slope of the V model and are conducted in that order with limited backtracking.

Parallel to the reification process (i.e., from requirements to runnable code), the V model enforces the definition of test plans. These plans form the up slope of the model. There is one test plan per phase of the down slope. For example, in the requirements phase, test acceptance must be defined and approved by the customer; in the global architecture phase, integration test plans must be defined, and during the specification of the detailed architecure, unitary tests are defined. Tests are performed in the reverse order of their specification. For example, acceptance tests are defined early in the development process but are performed before delivering the product to the customer.