*International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero'98), Montréal, Canada, 27-29 may 1998. p. 55-60.*

# Taxonomy for Human Error and System Fault Recovery from the Engineering Perspective

**Francis JAMBON**
LISI / ENSMA[1]
Téléport 2, BP 109
86960 Futuroscope cedex, France
+33 5 49 49 80 70
Francis.Jambon@ensma.fr

## ABSTRACT

This paper deals with human error resistance. In the first part of it, a short state-of-the-art of human error resistance, i.e. error prevention and error handling is presented. Then, error handling, which is usually divided into four sequential tasks – error detection and explanation, recovery planning and execution – is described.

The second part of this paper put emphasis on error recovery, which is our main object of study. First and foremost, through an example, we can see what makes the distinction between forward and backward error recovery in current taxonomy. Then the limits of this distinction are going to be highlighted.

In the third part of this paper we propose and illustrate, from the engineering perspective, our own taxonomy of error recovery suitable for real-world and dynamic systems.

### Keywords

Human error recovery, system fault recovery, taxonomy, human-computer interaction, man-machine interface engineering.

## INTRODUCTION

Although human errors are often observed in human-computer interaction, they are usually overlooked or ignored by most interface designers. Poor support for human error management in interactive system development may be an explanation of the designers' attitude. In this paper, taking into account designer's point of view, our aim is to support design for error recovery practices in interactive system development thanks to models.

We all know that "to err is human". So, designers must support human error resistance functions into the user interface life cycle. In this article, we propose a new taxonomy of human error and system fault recovery in order to help practitioners to choose among recovery alternatives.

## RELATED WORK

### Human Error

Human Error has been widely studied from the cognitive science perspective [2, 11, 16, 17] as well as from the applied science perspective [14, 18]. The definition of "human error" is an important topic, but cannot be reviewed here. For an overview on this topic, we suggest reading Reason [17].

One of the most significant result of human error studies – from the designer's point of view – is the Human Reliability Analysis [7]. As an example, the rather old THERP method [18] focuses on the evaluation of the Human Error Probability of nuclear power plant control rooms. Other contributions suggest that – extended – interface specification notations can be used to detect a breakdown during human-computer interaction [6].

With these results, designers can evaluate human error rates and their consequences on the system state, at the very first step of the design process. So, at this stage, human error resistance strategies can be defined in order to prevent or limit the consequences of human errors.

### Error Resistance Strategies

Error resistance can be achieved by means of two strategies: error prevention and error handling [12, 20]. Error prevention can result from forcing functions [15], operator's selection, or training. But error prevention will never get rid of all human error occurrences. That's why error handling – sometimes called error correction or error management – has to be actually supported in interactive systems.

Human error resistance is mostly supported – when that is the case – by user interface designers through prevention. Designers ought to follow the prevention strategy, for self-evident reasons. Unfortunately, they usually deal with human error only with the prevention strategy. And yet, scientific work [9], as well as aircraft accident report [13] stress the need for handling. In the former report, the authors criticize the fact that the human-system interface has given the crew too little

[1] Laboratory of Applied Computer Science of the National School of Engineers in Mechanics and Aerotechnics (Laboratoire d'Informatique Scientifique et Industrielle de l'École Nationale Supérieure de Mécanique et d'Aérotechnique).

chance to correct their disastrous error. Now, let's explore error handling.

### Error Handling

As an operator, handling an error or a system fault, can be achieved by following four sequential tasks as shown figure 1: Error detection, i.e. the user needs to know that an error occurred ; Error explanation, i.e. the user must understand the nature of the error – do note that for the undo function, the user does not always need to know that ; Error recovery planning and execution, i.e. the user has to counteract the effects of the error [21]. Some systems have embedded auto-correction features [12]: in such systems, the error handling is achieved by the system which is in charge of the handling tasks – diagnosis and recovery.

Error detection and explanation are the first steps – diagnosis – of error handling. They are primarily related to the interface presentation. In this paper, we focus on error recovery execution which involves the definition of the recovery procedures. Although error detection and explanation, and recovery planning are not our topic, we assume that they are compulsory in a perspective of design for error. For self-evident reasons, error recovery is void if the user cannot neither detect nor understand the error. Figure 1 summarizes the handling process of human error and highlights our topic of study.

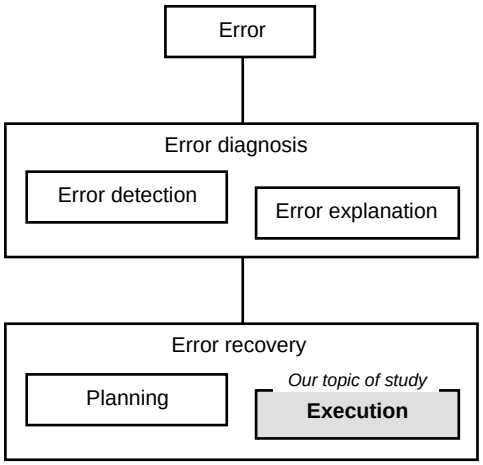


*Figure 1: The error handling process. Adapted from [21]*

## FORWARD VERSUS BACKWARD ERROR RECOVERY

Dix, Finlay, Abowd, & Beale [4] make a distinction between backward and forward error recovery. This distinction is now commonly used in interactive system development. However, we suggest that this distinction is not as useful as it could be from the designer's point of view. So we propose our own taxonomy.

### Backward Error Recovery

Backward error recovery is an attempt to restore the system state after an error has occurred. Backward recovery can be considered as the only real "recovery" function, since the unexpected effects of error are totally removed. Backward error recovery can be seen as a function to go back in time. As a result, novice users usually use it as a fail-safe learning method. According to Yang – in [9] – there are three kinds of backward error recovery commands: undo, cancel, and stop.

- The **Undo** function is the most famous one [3, 19], and most editors – for texts or graphics – implements an undo function. Unfortunately, the undo function is also the most complex one. Designers have to do with the presentation, the granularity, the scope, and the range of the undo function. Moreover, current implementations of the undo function seem to fall short of user expectations [10].
- The **Cancel** function is used to abandon commands under specification. For example, a user can cancel the typing of an e-mail message if the addressee has just entered the user's office. However, as for the undo function, the cancel function has to deal with its scope: the user must know which command(s) are concerned by the recovery.
- The **Stop** function is used to terminate the process under execution. For example, a user can make the choice to stop a long printing command when he realizes that many users are waiting for the printer. However, the stop function is not always implemented as a pure backward recovery: in the former example, the user can stop the printing command while the printer is working, so, some pages may have been printed uselessly.

### Forward Error Recovery

In forward error recovery, the user has to execute unexpected tasks to recover the fault. Usually the final result – the system state – is non-optimal. For example, if you break a dish plate, you have to use glue to recover your error. Of course, the final dish plate is not as nice as the unbroken one. In civil aviation, a lot of emergency procedures [1] are forward recovery procedures. In these procedures the "error" is a system fault. The result is generally a non-optimal state of the aircraft: after an engine fire, the latter can remain inoperative for the flight.

Forward recovery is commonly the only way to recover from technical failure or human error in critical systems like nuclear power plants, chemical plants, aircraft, vessels, etc. In these systems, side-effects of many actions cannot be easily removed. As Lenman & Robert [9] say "Forward error recovery is an important topic, and more research is needed concerning these questions". We do so.

### Limits of this classification

Dix et al. [4] use the example of an house of cards to show the difference between backward and forward error recovery: "for example, in building a house of cards, you might sneeze whilst placing a card on the seventh level, but cannot undo the effect of your misfortune except by rebuilding".

#### *Erroneous task granularity*

Dix et al. suppose that you cannot undo the fall of your house of card after placing a wrong card. Obviously Dix et al. consider that the erroneous task is placing one card on the house. But that's not so obvious, if we consider

that the real task is building an eight level house of cards. You may place a card in a wrong way, and then, the house falls down. Now collect all the fallen cards: that's a backward error recovery because you have undone the effect of your error. The system is exactly in the state before the beginning of your task "building the house of cards". Rebuilding the house of cards is now considered as a new task occurring after the recovery. So, the difference between backward and forward recovery is relative to the granularity of the erroneous task.

#### *Additional cost of error*

It is assumed that backward error recovery – undo, cancel, & stop – is the best way for a user to recover from an unexpected error. Clearly, pressing the "undo" button after a typing error is easy. Yet to achieve this goal, the user must retype the right letter, word, paragraph, or why not, the document. The cost of an error is clearly the cost of the correction task, as well as the cost of the correctly executed task. So, many users may prefer to use forward error recovery in order to move the system to a non-optimal state but at a lower cost. This is a critical topic from the designer's point of view in dynamic systems, in which the cost of the recovery is, at first, Time.

#### *Time*

In dynamic systems, the recovery functions are not always available. In these systems, Time is also an interesting dimension. In order to illustrate the importance of time, let us study the example of a crew who has forgotten to extend the landing gear of the aircraft. The crew can easily correct this mistake for a few minutes by activating the extension of the gear. A few minutes later, the aircraft is at a too low altitude, so, the crew must follow a go-around procedure. Finally, the aircraft lands on the runway – without it's gear being extended – and no recovery is possible.

## PROPOSED TAXONOMY

Based on these limitations, we proposed, in a recent publication [8], three dimensions of analysis – Error Additional Cost, System State Degradation, and Time – in order to help designers to understand the critical issue of error recovery from the operator's point of view. Our approach is now to define a novel taxonomy of error recovery, suitable for both static and dynamic systems.

### System State Modification

We first propose a new distinction between forward and backward error recovery based on the expected state of the system.

#### *Initial, Final, and Erroneous states*

Three states from the operator's point of view are here below defined:

- The **Initial State** is the system's state before the error.
- The **Final Expected State** is the system's state that the operator wants to achieve thanks to his action.
- The **Erroneous State** is the system's state resulting from the operator's error or system fault.

We assume that backward error recovery must be viewed as an attempt to restore the system state after an error or a failure occurrence. In other words, backward error recovery can be seen as a way to go back in time. As a consequence, all actions that restore the system's state after an error occurrence – not only undo and cancel functions but also complex or planned actions – are backward error recovery actions. Our definition of backward error recovery differs from the previous one [4, 9] on that point. As an example, the stop function cannot always be considered as a backward error recovery in our taxonomy, because the achieved state is sometimes closer to the final expected state than to the initial one as shown in the former printer example – nearly all of the pages may have been printed.

On the contrary, all actions which are an attempt to reach the final expected state must be considered as forward error recovery actions. So, complex planned action as well as simple atomic actions can be forward recovery actions. As an example, the correct performance of a previously forgotten action – an error can be due to a lack of action – is a forward error recovery in our taxonomy. Do note that to get the final expected state, the operator may execute a backward recovery action – undo for example – and then performs the correct action. However, these two sequential tasks cannot be considered as a forward error recovery because the effects of the error are first counteracted by a backward error recovery.

So, the transitions between the three states – initial, final expected, and erroneous – define the notion of error recovery as shown on figure 2.

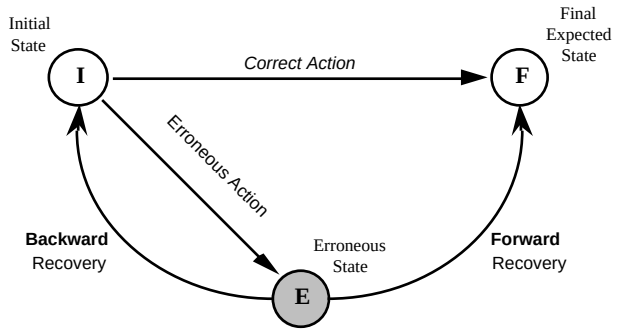


*Figure 2 : Error Recovery versus System State*

#### *System state degradation*

In real-world systems, the recovery – backward or forward – may not be perfect. In other words, neither the initial state nor the final expected state may not be reached from the erroneous state. As an example, the fire in an aircraft engine may be extinguished by the crew, but the engine often remains, then, inoperative for the flight. So, let us introduce a new distinction between perfect and imperfect recovery as shown on figure 3.

The imperfect recoveries introduce the notion of approach state for both initial and final expected states. These states are the result of an unsuccessful attempt to restore the initial state or to get the final expected state. In the former example of the dish plate, the use of glue to recover the error – the action of breaking the dish plate – is an imperfect attempt to restore its initial state –

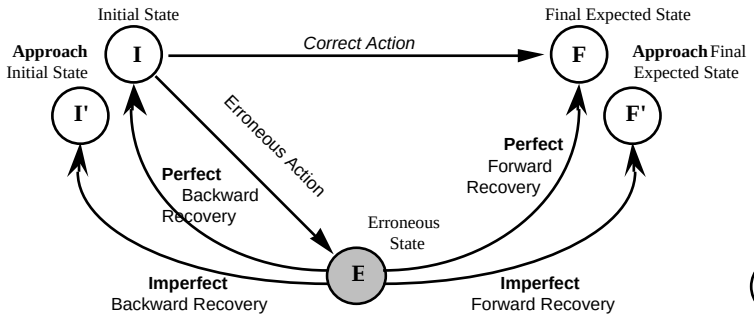


*Figure 3 : Error Recovery versus System State Degradation*

unbroken. Sometimes it is difficult to find which is the closest state of the reached state further to the imperfect recovery action. In this case, we suggest defining an intermediate state and not distinguishing backward and forward recovery.

First, let us study the failure of a aircraft landing gear system – as shown on figure 4 – to illustrate the concept of system state degradation. In our example, a failure in the extension of the landing gear occurs while the crew is attempting to extend the gear. Do note that this failure occurs during the operator's action, but must be considered as a system fault – in action – because the operator's goal cannot be achieved due to the system.

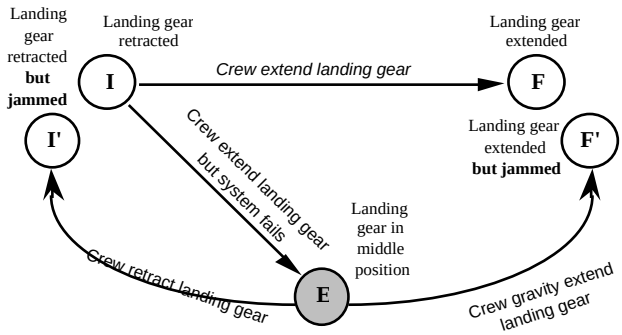


*Figure 4 : Landing gear failure example*

On the one hand, in order to recover the system's fault, the crew can extend the gear by gravity. This is an imperfect forward error recovery since the gear is extended, but also jammed in this position. The gear cannot be retracted – in our example – but this approached final state is although a safe state to land the aircraft.

On the other hand, the crew can retract the gear. This is an imperfect backward error recovery because the landing gear is retracted, but jammed in this position. We assume that this action is still possible. Then, the crew cannot get a normal extension of the landing gear: this is an approached initial state.

### *Sudden system fault recovery*

In case of a sudden system fault, or unwanted action, any final expected state cannot be defined. As a consequence, if we follow our taxonomy, only perfect and imperfect backward error recovery are possible as shown on figure 5.

Do note that the system fault which occurs during an action – for example the landing gear fault – must be distinguished from a sudden system fault – as an other example, an engine fire. A final expected state can be defined in the latter example and not in the former.

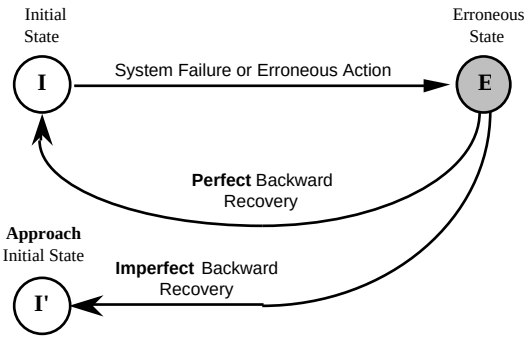


*Figure 5 :System Failure Recovery*

### Static versus Dynamic Systems

A static system, is defined as a system which state cannot be modified by time. In such systems, the erroneous state can only be modified by an operator's action. As an example, word processor softwares are static systems: the user can correct his error whenever he wants – if no action is performed in the meantime.

Whereas the state of a dynamic system may be altered by time. In such systems, as aircraft, vessels, etc., the error state can be worsened by time. Consequently, the recovery functions may not be always available. As an example, a fire in an aircraft engine may be recovered by a correct use of the fire extinguisher. If the procedure is not correctly followed, the aircraft may suffer from an engine separation…

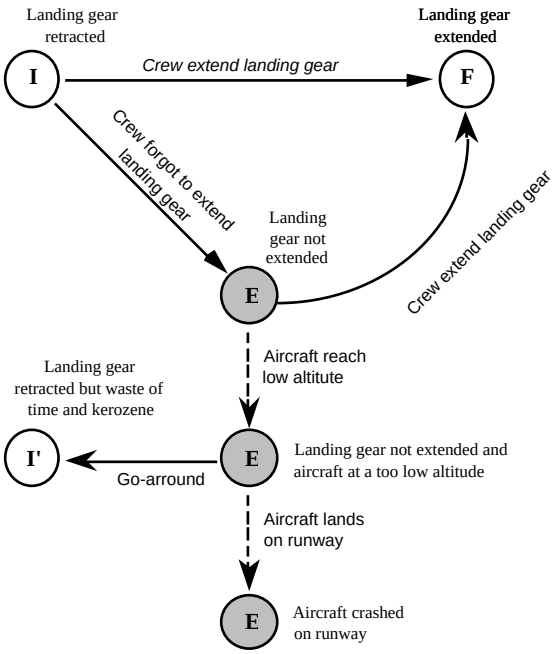


*Figure 6 : Forgotten landing gear extension example*

So, the notion of transient error state which represents a state that can be altered by time is here defined. In order

to illustrate this notion, the figure 6 uses back the example of the crew who forgot to extend the aircraft landing gear.

Do note that in dynamic systems, the error can be triggered by time. Indeed, a forgotten action – for example extending the landing gear – becomes an error after a few minutes – when the aircraft reaches low altitude.

### Recovery tasks

The operator's point of view makes out two types of recovery tasks:

- The **Generic recovery tasks** are undo, cancel, and stop functions. These tasks are well-known by the operator and are usually not difficult to perform. No planning is required and their cost, in terms of time, cognitive or conative [5] load, etc. is usually low. We would like to add to this category the forgotten actions performed by the operator after noticing the missing.
- Needless to say that **Planned recovery tasks** require planning for the operator. The cost of these tasks result usually high for the operator. However, the cost is not always higher than backward error recovery because the total cost of a backward error recovery is the cost of the recovery in addition to the cost of the correctly executed task [8].

This distinction between generic and planned recovery tasks is a engineering-driven distinction, i.e., it is related to the availability or not of atomic recovery functions – undo, cancel, and stop – in the man-machine interface. A more cognitive point of view can be taken here, considering that the recovery tasks are related to the three levels of control of human actions of Rasmussen's simplified human operator model [16]. Doing so makes out three types of recovery tasks: skill-based, rule-based, or knowledge-based recovery.

## DISCUSSION

The novel distinction between recovery alternatives can be put side-by-side to the former distinction between forward and backward recovery. This is the topic of the first part of this discussion section. Then, in a second part, some limits of our approach are going to be highlighted.

### Taxonomy mismatch

In our approach, we define a new taxonomy for error recovery. The two main dimensions – forward/backward & generic/planned – of our taxonomy are independent. So, from the operator's point of view, four types of recovery can be listed. Let us study the two recovery functions which are here below defined:

- Generic Backward Recovery in which we can find the well-known undo and cancel functions.
- Planned Forward Recovery generally used to recover from system failure or to correct some tiny errors.

As an example, let us study a typical task in a graphical editor like MacDraw®. The user's task is to create a perfect circle. In our example, the user forgets to press the shift key before drawing, so the circle is in fact an oval. On a one hand, the user can choose to manage the error by the undo function of MacDraw®, and re-draw the circle. This is a generic backward error recovery. On the other hand, the user can choose to turn his oval into an approximate circle by direct manipulation. This is a imperfect – planned forward error recovery. The system state is not optimal – the circle is not perfect – but enough satisfactory for the user's need.

We assume that these two error recovery functions are the more often recovery functions met in real world. They are shown on figure 7 on a gray tint background. We suppose that the former taxonomy of human error recovery [4, 9] merge the main dimensions of our novel taxonomy: forward/backward & generic/planned, i.e., it regards planned forward recovery as forward recovery and generic backward recovery as backward recovery. So, this former taxonomy is a subset of the proposed one. And yet, the latter is backward-compatible with the former.

However, the two other recovery functions listed below are critical issues in the design of real-world human-computer interfaces:

- Generic Forward Recovery gathering, for example, the performance of forgotten actions.
- Planned Backward Recovery often used to recover from sudden system failure.

In order to illustrate our position, the figure 7 reveals some typical examples of human error and system failure recovery in accordance with the main dimensions of our taxonomy.

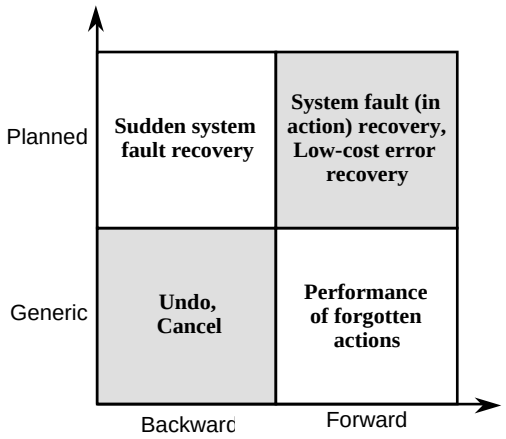


*Figure 7 : Example of the two main dimensions of our error recovery taxonomy*

### Limits

First and foremost, our approach is only an engineering view on human error and system fault recovery. We admit that this is an important limitation of the taxonomy. However, our approach is a bottom-up view of human error, i.e., error is only studied from the engineering perspective of the availability of recovery functions. No assumptions are done about cognitive aspects of detection, explanation of error, nor recovery planning. Our contribution deals with the execution of recovery. This view is directly related to the engineering process of

man-machine systems: it should help designer to explore recovery alternative through available recovery functions.

Another point limits our contribution. We consider that the operator goal is always to recover from the error. Sometime, due to lack of time in dynamic systems, the operator may want not to recover from an error in order to save time, i.e., to live with error.

## CONCLUSION & FUTURE WORK

As a conclusion, we hope we have provided designers a new way to understand the operators' point of view in error recovery. We assume that these dimensions can be used to evaluate system failure as well as human error recovery alternatives of safety-critical systems. By doing so, we hope that this taxonomy can ensure more safety in the conception of the Human Computer Interface of these systems. Obviously, these dimensions can also be used in the design process of non-critical desktop computer interfaces in order to evaluate the real benefits of recovery.

This article deals with a taxonomy of human error and system fault recovery from the engineering perspective. This taxonomy is a first attempt to enrich the classical human reliability analysis step of the critical man-machine interface engineering process. The next step of our approach, which is in progress, is to identify the basic patterns – in the dialogue controller of man-machine interfaces – that implements recovery functions. Then, we will make links between the taxonomy and these patterns.

## ACKNOWLEDGMENTS

The author would like to acknowledge the participants of the *"User Interfaces for All" ERCIM workshop* whose questions inspired this article. Many thanks to Sylvie Ferrés for the English review of the article and Jean-Marc Robert for his encouragement leading me to write it.

## REFERENCES

1. Airbus Industrie. *Flight Crew Operating Manual A320*. 1992.
2. Amalberti, R. *La conduite des systèmes critiques.* Presses Universitaires de France, Paris, 1996.
3. Dix, A., Mancini, R. and Levialdi, S. The cube - extending systems for undo, in *Proc. Eurographics Workshop on Design, Specification, Verification of Interactive Systems* (Granada, Spain, June 4-6, 1997), Eurographics, Springer-Verlag, 473-495.
4. Dix, A.J., Finlay, J., Abowd, G. and Beale, R. *Human-Computer Interaction*. Prentice Hall, 1993.
5. Dorwell, J. and Long, J. Towards a conception for an engineering discipline of human factors. *Ergonomics*. 32, 11 (November 1989), 1513-1535.
6. Gray, P.D. and Johnson, C.W. Supporting error-driven design, in *Proc. Eurographics Workshop on Design, Specification, and Verification of Interactive systems (DSV-IS'96)* (Namur, Belgium, 5-7 June, 1996), 207-228.
7. Hollnagel, E. *Human reliability analysis: context and control*. Academic Press, London, UK, 1993.
8. Jambon, F. Error Recovery Representations in Interactive System Development, in *Proc. Third Annual ERCIM Workshop on "User Interfaces for All"* (Obernai, France, 3-4 november, 1997), 177-182.
9. Lenman, S. and Robert, J.-M. A framework for error recovery, in *Proc. International Ergonomics Association (IEA'94)* (Toronto, Canada, August 15-19, 1994), International Ergonomics Association, 6, 374-376.
10. Lenman, S. and Robert, J.-M. Investigating the granularity of the Undo function in human-computer interfaces. *Applied Psychology: An international review*. 43, 4 (special issue on Human Errors) (1994), 543-564.
11. Leplat, J. *Erreur humaine, fiabilité humaine dans le travail*. Armand Colin, Paris, France, 1985.
12. Lewis, C. and Norman, D.A. Designing for Error in *User Centered System Design / New Perspectives on Human-Computer Interaction.* Lawrence Erlbaum Associates, 1986. pp. 411-432.
13. Ministère de l'équipement des transports et du tourisme. *Commission d'enquête sur l'accident survenu le 20 janvier 1992 à l'Airbus A320 F-GGED près du mont Sainte-Odile (Bas-Rhin)*. Journal officiel de la République française, édition des documents administratifs, n°31, Rapport final ISSN 0242-6773, 26 mars 1994.
14. Nicolet, J.-L., Carnino, A. and Wanner, J.-C. *Catastrophes ? Non merci ! La prévention des risques technologiques et humains*. Éditions Masson, Paris, France, 1990.
15. Norman, D.A. *The design of every day things*. Doubleday Currency, New York (NY), USA, 1990.
16. Rasmussen, J. *Information processing and Human-Machine Interaction : An approach to cognitive engineering*. Elsevier Science, Amsterdam, The Netherlands, 1986.
17. Reason, J. *Human error*. Cambridge University Press, New York (NY), USA, 1990.
18. Swain, A.D. and Guttmann, H.E. *Handbook of human reliability analysis with emphasis on nuclear power plant applications*. Nuclar Regulatory Commission (NUREG), Final report NUREG/CR-1278F, August 1983.
19. Thimbleby, H. *User Interface Design*. ACM Press, 1990.
20. Van der Schaaf, T.W. Prevention and Recovery of Errors in System Software, in *Proc. Workshop on Human Error and System Development* (Glasgow University, Scotland, 19-22 March, 1997), Glasgow Accident Analysis Group, GAAG TR-97-2, 49-57.
21. Zapf, D. and Reason, J.T. Introduction: Human Errors and Error Handling. *Applied Psychology: An International Review*. 43, 4 (1994), 427-432.