Considering Possible Outcomes and the User's Environment in Designing User Interfaces for Data-Intensive Systems

Karen Renaud
University of South Africa
P O Box 392
UNISA, 0003
SOUTH AFRICA
renaukv@unisa.ac.za

Richard Cooper
University of Glasgow
17 Lilybank Gardens
Glasgow, G12 8RZ
UNITED KINGDOM
rich@dcs.gla.ac.uk

Abstract

Application programmers are often unrealistic about the end-user's working environment and seldom cater for the effects of events which will interfere with the use of the application. Such events can disrupt the straightforward execution of a task and interfere with a user's concentration. These events, which will be referred to in this paper as "quirks", could be system breakdowns, various types of interruptions to application use, or human errors. Applications often make no concession to the inevitability of quirks and seldom give assistance in rebuilding mental context afterwards or facilitate understanding of the cause in the case of an error.

In addition to the normal quirks caused merely by sharing office space or in working as part of a group of people, most data-intensive systems are distributed and this tends to precipitate a whole range of errors, hitherto unsuspected, which will probably be reported to the user in all their technical verbosity, reducing the user's understanding of, and confidence in, the system and perhaps necessitating intervention by specialists. The inherent distributed nature of data-intensive systems also increases the likelihood of breakdowns, since so many more computers are involved in the application than the computer being used by the end-user.

Few applications consider the effects of quirks while developing their systems, and the user is therefore unsupported in recovering from them. This paper discusses how applications may be designed to better support users in dealing with the effects of quirks in data-intensive systems.

1 Introduction

The trend towards the online use of data-intensive systems has led to a greater emphasis on the importance of the usability of applications. One often overlooked aspect of usability concerns the context in which the user is working. Take, for example, a designer using software which manages design data held in a database. The designer may well be working with a necessarily complex user interface in an open-plan office on a networked computer. This gives rise to a greater variety of error possibilities than will be evident with a single-user system or with a client application tailored for naïve users. As well as standard problems such as faulty software or user mistakes, the network might fail or the user might suffer an interruption.

In this paper we discuss the effect of the various kinds of problems that the user might suffer, grouping them all together under the term "quirk". It should clearly be one important aspect of support for the user that the application help the user cope with quirks. Unfortunately, we find that this is usually not the case. Problems are reported using technically verbose language which does not always let the user know what caused the problem. What is needed is support for recovery by giving assistance in rebuilding the user's mental context after a quirk and by facilitating understanding of the cause of the problem.

Few application developers consider the effects of quirks while designing their systems, and the user is therefore unsupported in recovering from them. This paper discusses how applications may be designed to support users when dealing with the effects of quirks in data-intensive systems.

We start by considering the method of use assumed by application designers, as being one from which the user regularly departs. This is shown in Figure 1, as the direct route which proceeds from the initial state I to the final state F, reaching this upon completion of the task. Using this direct

Proceedings of the Second International Workshop on User Interfaces to Data Intensive Systems (UIDIS '01)
0-7695-0834-0/01 $10.00 © 2001 IEEE

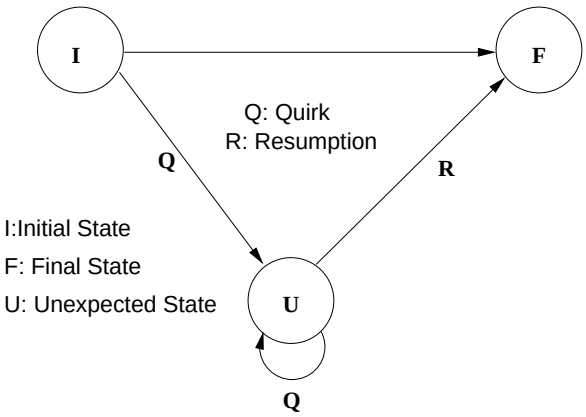


Figure 1. Initial and Final States in Task Execution

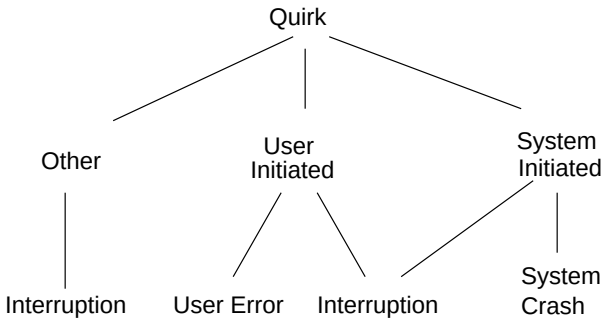


Figure 2. Classification of Quirks

path, with no detours on the way, is only *one* possible way of proceeding. In reality, this is a simplistic and unrealistic view of the way humans interact with computer applications.

The execution of a task can be disrupted by a *system breakdown*, a *user error* or an *interruption* — these will all be referred to as **quirks** — as indicated by the transition leading to an unexpected state as depicted by node U in Figure 1. The recursive arrow indicates the possibility of multiple quirks occurring between the start and end of application use. Humans are basically serial in their operation [38], so that they can process only a few symbols at a time. To compensate for this, these symbols must be held in complex structures in working memory - the *mental context* of the interaction - while they are being processed. Quirks cause the mental context to collapse while the user deals with something unexpected or unrelated. It is thus logical that quirks will tend to be troublesome.

Many researchers have worked on each of these different aspects — errors, interruptions and breakdowns — in isolation, but since there is often a commonality in the user's handling of each of these and in the effects on the user's emotions and task completion, it is useful to consider them as forming part of group of similar concepts. This paper will consider quirks with particular emphasis on distributed, data-intensive systems.

Figure 2 gives a classification of quirks, which are a superset of Jambon's singularities [16]. Quirks can be initiated either by the user, by the system or by some external entity (*Other*) — and can disrupt the user's task processing by demanding attention elsewhere (The telephone could ring, for example). The user could make an error, or interrupt the process voluntarily. Users often switch tasks when the computer is slow in completing a task [8]. The system could crash, or interrupt the process. The presence of a quirk could cause the system to end up in any of a number of different states, depending on the user's handling of the quirk. These different states will be explained in detail further on.

Section 2 discusses the importance of quirks. Sections 3, 4 and 5 will describe the nature of each of the three types of events which are grouped under the label of quirks. Section 6 makes recommendations about how systems can be designed to ease recovery from quirks. Section 7 introduces the HERCULE prototype, which assists users in recovering from quirks, and Section 8 concludes.

2 Do Quirks Merit Consideration?

Quirks are not merely an irritating fact of life. Research shows that they can raise worker stress and in some cases affect the health of workers [3, 18, 20, 41, 43, 45]. Interruptions have been shown to lead to lower quality decisions and reduced speed on intellective tasks [39]. However, while many people perceive quirks to be exclusively negative, there are also occasions when quirks have positive effects [17, 28, 35].

Quirks have a substantial effect on users' interaction with applications due to human information processing limitations. A user who is busy with some activity builds up a *context* [6] — a rich mental environment that stores all sorts of information built up during the time spent using that particular system to execute that particular task. Even a momentary interruption, such as a quirk, causes the mental context to collapse. The extent to which the system designer develops the system with possible disruptions in mind, to ease recovery of context after a quirk, will therefore contribute to the usability of the system.

Quirks are more common than one might expect. Perry *et al.* [30] found that a group of software developers spent 75 minutes *per day*, on average, in unplanned interpersonal interactions which interrupted their work. Surveys of computer use by expert users show that up to 10% of working

time can be spent handling errors [3]. Eldridge and Newman [10] found that the negative effect of a technological fault due to time lost in dealing with it was exacerbated by the damage done to the rest of the day's activities. Often one person's technological problem has an effect on other people's agendas too, so that a "simple" computer breakdown often has a bigger impact than meets the eye.

Users will have different attitudes towards quirks. The frequency of the quirks, together with user perceptions of the benefits or disadvantages of quirks, will play a part in determining users attitudes towards an application in particular, and towards their work satisfaction in general. Fogg and Nass [11] argue that the rule of reciprocity, which exists in all cultures, also applies to human-computer interactions. As a consequence of this, users will tend to "help" computers that have previously helped them and retaliate against computers that have performed poorly. The frequent occurrence of errors would therefore tend to have far more long-term effects than merely the time spent in repairing the error would suggest.

Breakdowns will be discussed in Section 3, human error will be discussed in Section 4 and interruptions will be described in Section 5.

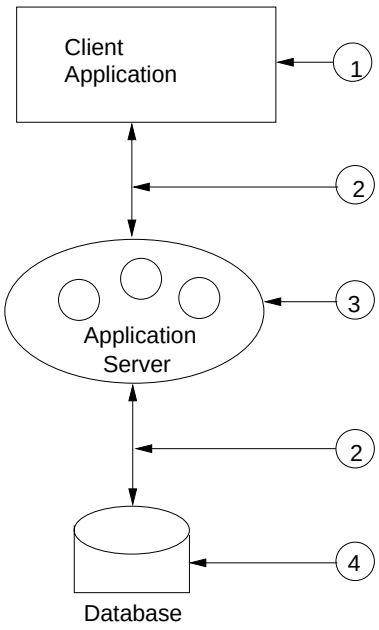


Figure 3. Breakdown Location

3 System Crashes and Breakdowns

Data-intensive systems are typically distributed, involving components at possibly more than one geographical location. The possibility of something breaking down is thus greatly increased, as is the possibility that one person's computer breakdown will affect other users [10]. The type of problems which can be classified as breakdowns are a failure of (shown in Figure 3)[1]:

1. *the user's computer,* a failure of either some application or of the whole computer.

2. *the network*, which can be affected by a crash, an omission, arbitrary or timing errors [23].

3. *an application server*, which could be the failure of the server host, or failure of the server housing the server component.

4. *the data store being used.* Since the application is completely separated from the data store by the middle tier, this type of failure will present as a failure of the previous type.

[1]The situation has been simplified for the purposes of this paper. Breakdowns are often very difficult to categorise in this fashion since their reporting is so difficult to do effectively and multiple network stages make errors difficult to interpret.

In the case of the end-user computer crashing, the user is generally left with little choice about how to handle the situation or doubt of its severity. After a crash, the user generally ends up in state IR shown in Figure 4 — the initial state reinstated after a recovery. This is not the same as the initial state I, since any application state built up before the crash will be lost and the user's context has been modified by the lost work. The work done before the crash might persist, depending on whether the application on the end-user computer had communicated the commands to the underlying data store or not. The user will have to check on whether their work was "saved" or not after the end-user computer starts up again.

In the case of a breakdown of the other computers involved in the distributed system or of the network, things become more difficult. The failure of some section of the system will mostly manifest itself by the reporting of an error by the end-user application. Sometimes the user will simply be faced with a lack of response from the computer, which could indicate a breakdown, but which could also conceivably simply be a symptom of an overloaded network. After a certain time period, the user will detect the problem and assume that the application has crashed. Whatever the source of the problem, the user is left to deal with the results thereof. The rest of this section will therefore address the effects of breakdowns on the user — whatever their source.

The handling and effects of possible breakdowns can be classified on three axes — *extent, time taken to recover* and

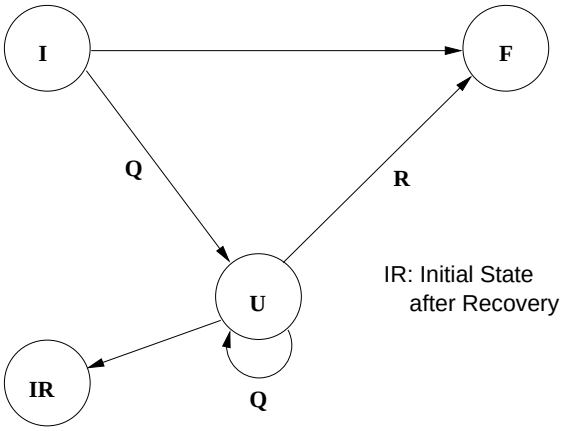


Figure 4. States in Task Execution, including state IR

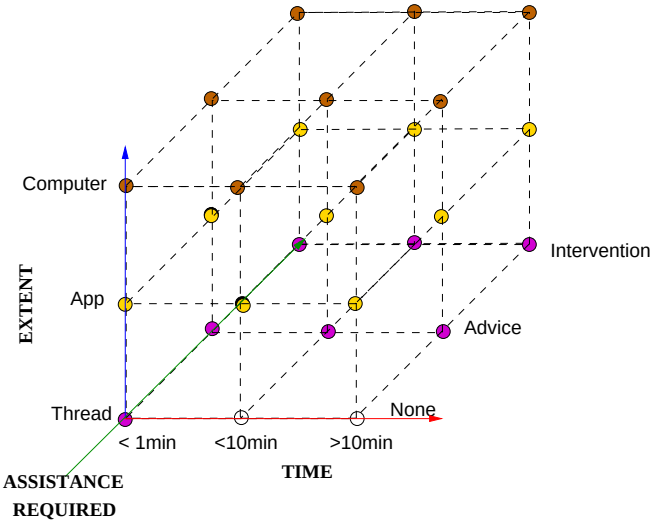


Figure 5. Classification of Breakdowns

assistance required [17]. The resulting graph is shown in Figure 5.

Each of the axes will be explained in turn. The planes of the Y axis (labeled *Extent*), refer to the severity of the breakdown which is one of:

1. *moderate* — where the user's immediate process is disrupted. This is typically the failure of an application thread.
2. *severe* — where the user's entire task is disrupted. This is the failure of the application.
3. *chronic* — where the entire end-user computer crashes and no work can be done.

A computer failure cannot realistically be resolved in less than 10 minutes and an application failure cannot be rectified in less than one minute. Intervention cannot realistically occur in less than 10 minutes, since presumably the user would have to summon assistance.

The X axis, labeled *Time*, refers to the time taken for the user to recover from the breakdown. This axis has three possible values, linked to the recovery from the disruption of the user's task. The values have been split up into the values of < 1 minute, < 10 minutes and > 10 minutes. This is due to the findings of Brodbeck *et al.* [3], which show a sharp increase in negative emotions when longer than 10 minutes is spent resolving an error. The Z axis, labeled *Assistance Required* has three possible values:

1. The user will sometimes be able to handle the recovery from a breakdown — linked to value *none*.
2. The user may telephone someone for advice, or consult a manual — linked to the value *advice*.
3. When all else fails, the user may have to request *intervention* from a specialist.

Once again the planes can be limited since it is simply not possible to get advice or assistance in less than a minute and intervention will probably take longer than 10 minutes to summon.

When all these restrictions are taken into account, the classification graph is reduced to the one shown in Figure 6. The obvious conclusion to be drawn from this graph is no surprise. The summoning of assistance from a specialist should be minimised so that the user's problem can be solved in the fastest possible time, thereby improving productivity and minimising stress. It is also obvious from the graph that breakdowns are almost certain to lead to negative emotions, something to which any computer user can attest.

4 Human Error

There is ample evidence in the literature [31] to lead to the conclusion that humans do indeed err, that they are unrealistic about their propensity for making errors and their ability to detect them, and thus, having erred, will convince themselves, in spite of clear evidence to the contrary, that they did not err [41, 44].

Errors will have to be handled in the course of a user's working day and can be considered to be part and parcel of the task execution — albeit an unpleasant or unexpected one. Error recovery can be likened to a "repair" effect often encountered in conversation. The end-user application will, as does the listener in a conversation, give negative feedback if it either does not understand, or is not satisfied with, the inputs the user is providing. The user (the speaker) will

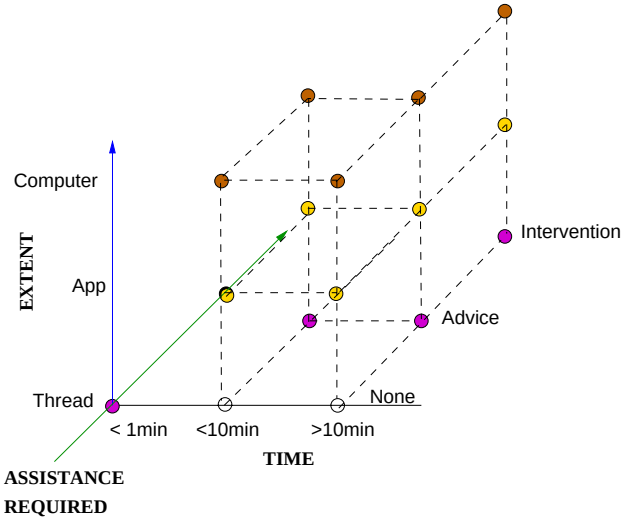


Figure 6. Classification of Probable Breakdowns

then attempt a repair and get the human-computer "conversation" back on course.

Errors are generally split up into two distinct types — *slips* and *mistakes*. Slips generally result from unintended actions, where the action does not match the intention. Mistakes are intended actions, and occur because a user does not understand the system correctly and are thus far more difficult to recover from [31].

Surveys of computer use by expert users show that up to 10% of working time is spent handling errors [3]. Errors can therefore be expensive in both human and economic terms. The way error situations are handled is thus critical for usability.

4.1 Error Detection

Detecting an error is the first step towards recovery. It is often hard for a user to detect an error due to overconfidence, with the user using intelligence to explain away unusual occurrences thus failing to register the presence of an error.

Errors will typically be detected by some mismatch between what the user thinks the state of the system *should be* and what it seems to be. The user relies on some feedback mechanism — either by the computer or by some other means — which enables the user to compare what is expected with what has occurred. Data-intensive systems are generally structured in three tiers, which means that the state of the data store, which generally makes up the lowest tier, must be displayed by the end-user application. Thus the end-user application developer needs to portray, in the user interface, not only the state of the end-user application, but also that of the underlying data store — information which must be obtained remotely. It is difficult for the end-user application to portray enough information about the underlying data store based only on limited information gleaned from remote method invocations.

Most systems react to errors by generating error messages, but error messages are not necessarily the solution to the problem [19, 24]. Error messages generated at lower tiers of the system will generally not cater for the current state of the end-user application or the context from which the user needs to be relocated. Hammond [14] points out that interpretation of unfamiliar information makes heavy demands on working memory. An error message can be seen as an unfamiliar situation — since an error is always an unexpected occurrence. Thus it is to be expected that the user will be extremely likely to forget exactly what was being done prior to the error situation. This means that error recovery is not necessarily a simple process. The user needs to diagnose the source of the problem, and then correct it. Diagnosis implies an understanding of the state of the system, and recovery requires an understanding of the repair process.

4.2 System State after an Error

The occurrence of a user error can cause the system to enter a number of states, as illustrated by Figure 7. The discussion so far did not distinguish between user and system error detection. There is a need to distinguish between *system* detection of an error and *user* detection of an error because this typifies the so-called "gulf of evaluation" [25]. The width of this gulf is determined by the quality of the feedback in the user interface.

System Detection. If a user submits some input for a system to act upon, the system could detect an error and abort the action. The system then informs the user of the error with the success of the notification depending firstly on the quality of the feedback and secondly on whether the user is concentrating on the system at the time. If the user ignores or misses this notification and continues working, the gulf of evaluation has become wider and future actions will possibly be affected by this misunderstanding.

If the user does indeed realise that an error has occurred, either a decision can be made to abort the task — ending up at state IA (Initial State after an Abort) shown in Figure 8 — or to correct the input and continue working. Since the error was detected by the system, the effects of this error are not critical and the consistency of any underlying data store will not be compromised.

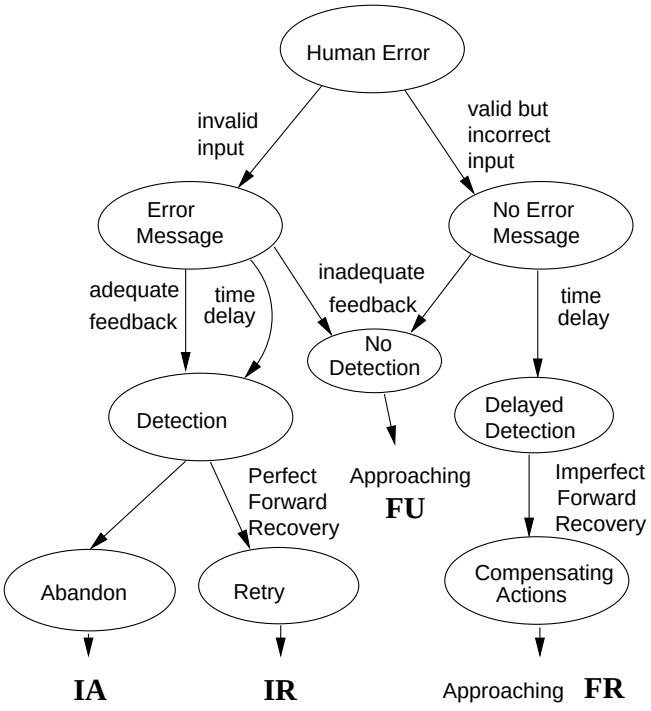
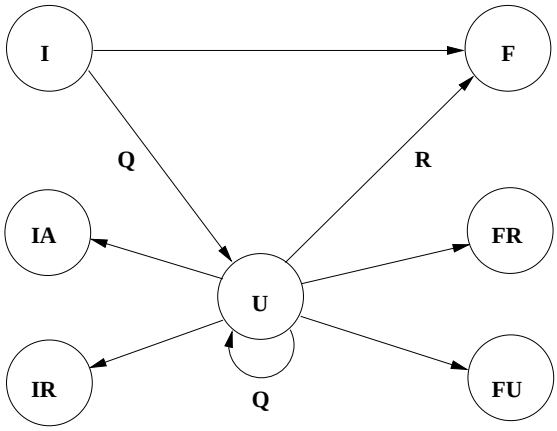


Figure 7. Analysis of an Error Occurrence



IA: Initial State after Abort
FR: Final State after Forward Recovery
FU: Final State after an Undetected Error

Figure 8. All Possible States in Task Execution

User Detection. If the user provides input to the system which is valid, but not what they intended, the system has no way of realising that this is a mistake on the part of the user and accepts the input. The input will thus be processed and changes will possibly be made in one or more underlying data stores as a result.[2] If the user were to discover the error, as a result of its effect, a decision could be made to supply inputs to the application which compensate for the error. The user could continue to work on the task in hand, but the final state will not be state F, but rather state FR, since another user could have made use of the incorrect information between the erroneous action and the compensation. If the user does not realise that an error has been made, then the gulf of evaluation, which has just become wider, needs to be bridged in order for the user to realise that an error has been made. The system is now in state FU, since the state of the system is not what the user intends and the consistency of the underlying data store has possibly been compromised.

The effects of user errors could accumulate, affecting the eventual recovery process and the error handling time, and exacerbating long-term effects of the error. The more unresolved errors in the system, the more time and effort will be taken to restore the data store to the correct state.

4.3 Error Recovery

Many applications today provide an *undo* feature so that a user can backtrack and undo the effects of an error [1]. In non-transactional systems the undo function will work admirably but is probably not an option in transactional systems. If the system detects the error, undo is not really necessary since the database will not be affected by the error. If the system does not detect the error, undo is also not an option, unless the application is "intelligent" enough to generate a compensating transaction automatically. Thus in a transactional system, slips, which are traditionally easy to recover from, become far more difficult to manage.

Recovering from mistakes requires complex actions compelling the user to go back through some actions to recover [3]. Users will often realise that something is amiss with their reasoning, or method of achieving the goal, but are at a loss as to how to go about recovering. Rizzo *et al.*[34] argue that most mistakes depend on the misactivation, conscious or unconscious, of knowledge. Rizzo *et al.* propose the following guidelines for supporting the handling of human errors [34]:

[2]Most distributed data-intensive systems are structured as an n-tier and the nature of these systems implies that every remote method invocation originating from the end-user application constitutes a complete transaction. Thus each method invocation will potentially make immediate and durable changes to the underlying data store.

1. *Make the action perceptible* — by this is meant that designers should make the match between action and outcome more obvious.

2. *Display the error message at a high level* — messages should be displayed at the user's level of understanding, with the possibility of getting more detailed messages should they be required.

3. *Provide an activity log* — thus supplying people with an external memory aid.

4. *Allow comparisons* — the user must be assisted in comparing the state with other, perhaps intended, states.

5. *Make the action result available to user evaluation* — this needs to be achieved as soon as possible. This aspect coincides with the discussion on feedback in the following chapter, which stresses that the feedback should provide aspects relevant to the task just performed.

6. *Provide result explanations* — the best way to provide error diagnosis is to give specific answers to the user. The user should not be overwhelmed by reams of explanations. The user should only be given a high-level message, with further details available upon request.

5 Interruptions

Interruptions pervade our 21st Century lives. Telephones ring, people pop into the office and email continuously demands to be read and answered. For example, studies by van Solingen *et al.* [40] into the effects of interrupts in software development found that the subjects of the study spent 20% of their time servicing interrupts. A study by Rouncefield *et al.* [35] found that the staff in one particular organisations actually prefered handling interruptions to doing more mundane tasks — so that interruptions are clearly not universally considered to be negative.

Disruptions have been shown to inhibit performance in the execution of complex tasks [39]. Attention is broken if the same sensory channel is used by the disruption as is being used by the current task. Computer application users are using their eyes, ears and touch senses (via their fingertips). They are also making heavy use of short-term memory. Thus computer users are less tolerant of interruptions than traditional workers because it disrupts their short-term memory and makes it hard for them to continue their task easily [2]. This intense use of the person's cognitive abilities is in stark contrast to the traditional nature of the workplace where social interaction plays an important part in making up the person's working day and often makes it more enjoyable.

Interruptions tend to break what Dix *et al.* [8] refer to as *the loop of interaction*. This means that there could be a delay between user actions and the feedback on these actions — so that the action and the observable effect can no longer be linked in the user's mind. Users tend to operate in terms of an action-evaluation of effect-action paradigm and once the time delay between action and observable effect is longer than the short-term memory span the evaluation becomes difficult and decisions about the following action take longer.

Interruptions can occur concurrently or consecutively. Humans routinely handle up to five activities simultaneously by interleaving them. Cypher [6] maintains that they do this by *linearising* — organising the parallel activities into a single linear stream of actions.

This interleaving of activities could be voluntary — such as when we decide that we do not want to wait for something to finish, and switch to another activity — or involuntary when, for example, the phone rings and has to be answered. In Section 2, the context which a user builds up during an activity was mentioned. Waern [41] notes that working memory is only able to retain information for a couple of seconds at a time and that unexpected interruptions can thus be fatal to an entire problem solving process. Studies have shown the process of switching tasks to be costly. Cutrell *et al.* [5] cite a study by Gopher *et al.* [13] which has shown that the cost is related to the nature of the current and pending activity, as well as the user's proficiency in the task at hand. The context switch cost can affect a person's general performance, stress levels and job satisfaction. One way to measure the cost of task switching is to gauge the amount of time it takes to recover from interruptions. A study by van Solingen found the recovery time after an interrupt to be a minimum of 15 minutes [40].

When people are doing paper work it is relatively simple to mark their current position so that they can return later [35]. In order for a computer system to support the user in linearising of multiple activities, it is essential that the user be provided with some sort of memory aid. This should keep the activity visible and provide a way for the user to "pick up the threads" as quickly as possible upon resuming an activity. It is hard for applications to provide this facility effectively. Czerwinski *et al.* [7, 5] experimented with the provision of a marker to assist users returning to previous on-screen tasks but found that this did not assist users as much as expected.

Care should be taken that any provided memory aid should itself not be distracting or clutter up the display. There is a continuous trade-off between providing the user with external memory aids and the limitations of working space [22].

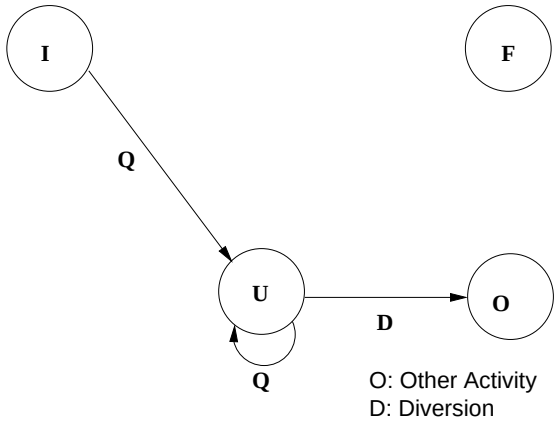


Figure 9. Non-Resumption of the Primary Task

5.1 Handling Interruptions

It may appear that a user will simply handle an interruption and then continue with the task. This might indeed be the case for isolated interruptions, but repeated and numerous interruptions may require more sophisticated handling. The user in the process of handling one interruption could be interrupted by yet another. The interruptions could be handled immediately so that the first interruption is suspended to deal with the most recent one, or the most recent one is queued and forced to wait until the handling of the first one has been completed [42]. The user may also choose to interleave the handling of the interruptions. After the interruption, the user may resume the original task, but in 45% of cases, according to a study done by O'Conaill and Frohlich [28], the user will not resume the disrupted task, but will be diverted to another task. This is illustrated in the diagram in Figure 9, by the transition to node O (Other activity state), instead of node F (Final state).

After the interruption has been dealt with, the user needs to change context again and decide which task to proceed with. Miyata and Norman [22] suggest that a system of reminders might be a good idea in ensuring that the user does indeed resume a suspended activity. Human memory limitations require these prompts, if a potentially critical activity is not to be forgotten.

5.2 Recovery from Interruptions

In order to assist the user in recovering from interruptions, it would thus be helpful to have the following features provided by the application:

- *mental aids*, to help the user remember past actions;
- *graphical features* to allow the user to take a couple of steps back to rebuild the mental context.
- *user assistance* in building an awareness of the history of interaction with the application, by linking past inputs to the results — or outputs – thereof.

Since each user has different "remembering" needs, the principle of giving the user an overview and then allowing zooming-in [37] to get required detailed information, applies here.

6 Designing for Quirks

Jambon [16] urges system developers to design with interruptions and errors in mind. He argues that this would decrease the possibility of operators forgetting something critical after handling a quirk, thereby causing a serious accident. The focus of Jambon's research was interfaces for pilots. Errors made by users using other systems may not have such serious repercussions as those made by pilots, but that does not make them any less annoying. The discussion of the different types of quirks has made the need for two distinct different types of feedback obvious. The first type is feedback with respect to the latest user action, or error, being reported to the user — this can be referred to as *immediate feedback.* The second type is feedback with respect to past actions, to support the user in rebuilding context about what they were busy doing before the occurrence of the quirk — this can be referred to as *archival feedback* [32]. The contribution made by both types of feedback in alleviating the effects of each of the quirk categories will be discussed in the following sections.

6.1 Breakdowns

Immediate feedback is not much use if the end-user computer breaks down. Archival feedback can only be useful if it persists. If another part of the distributed system breaks down, it will depend on the forethought of the application designer whether the system will respond in a helpful way or not. If the breakdown was not anticipated by the designer during system development, the user is sure to receive an unintelligible response. Archival feedback could very well be helpful to the specialist summoned to track the events leading to the breakdown. What *will* be useful is some way of understanding exactly what the problem is together with some indication of the course of action to be taken.

It is notable that, whereas a data-intensive application is expected to recover the consistency of the data store after a breakdown, there is often no equivalent attempt to recover the user's mental context at the same time. With archival information being available, there is no reason why the user

interface might not be similarly rebuilt, given a modern user interface toolkit, such as Java's Swing [21].

6.2 Human Error

The recommendations given for error recovery by Rizzo *et al.* [34] for supporting the handling of human errors were discussed in Section 4.3. The first, second, fifth and sixth recommendations are satisfied by immediate feedback, while the third and fourth are satisfied by archival feedback. Zakay [44] has shown that immediate computerised feedback reduces overconfidence which means that the error is more likely to be detected, and archival feedback is likely to assist the user in understanding an error, and in comprehending the current state of the system. This should lead to speedier error recovery.

6.3 Interruptions

Users have severe limits with respect to memory, often forgetting what they have done, and they often experience difficulty in holding recently experienced information until needed [29]. Users can be supported in handling interruptions by archival feedback, which assists them by providing a mental aid to help them remember things [26, 37]. A log of recent actions can help by reminding the user of the activity that was interrupted.

Archival feedback can also be used to provide interreferential feedback. Draper [9] argues the importance of a mutual reference between user input and application reaction so that the previous parts of the user-machine dialogue can be referred to. A mere list of the user's inputs to the system is of limited use because there is no way for the user to remember what the application state was at the time that particular command was entered or menu choice made. In order to support users adequately archival feedback should provide such a link between user actions and application state and allow browsing of such information.

7 HERCULE

The HERCULE[3] prototype was developed, using Java, to be a generic facility for the visualisation of application activity. HERCULE provides both context sensitive immediate feedback, archival feedback, and overview functions [33]. This supports the user in understanding the nature of breakdowns and errors, and in recovering from the quirks mentioned in this paper.

HERCULE's approach is that feedback be provided in a generic fashion, produced *independently* of the application implementation. This approach necessitates treating the provision of feedback as a *separate concern.* This well-established technique has been successfully applied in separating several non-functional characteristics from the main concern of application programs, but has hitherto not been applied to the provision of feedback. Separating feedback provision from the application makes things easier for the programmer and provides a mechanism for augmenting the feedback provided by the application itself.

There are many approaches to achieving separation of concerns [15]. One approach, application tracking, requires the least effort from the programmer and was thus the approach applied in the development of HERCULE. It is also the least invasive way of achieving the required separation of concerns. Application tracking is widely used for many purposes, but once again has not hitherto been used to augment application feedback.

The success of the HERCULE prototype has shown that this means of augmenting application feedback can indeed be used and that it enriches the concept of feedback in such a way that it can enhance the recovery process in the presence of quirks.

8 Conclusion

There is a commonality in the user's handling of errors, breakdowns and interruptions. In the case of error, the user has to *understand* the cause of the error and *understand* how to recover from it. In the case of breakdowns, the user needs to *understand* what caused the breakdown and what, if any, action should be taken to recover. In the case of interruptions, the user attempting to resume context must *correctly perceive* the state of the application in order to take up their task at the point of interruption.

We can conclude that feedback which enhances the user's comprehension of the application state, and the events that led to that state, is a valuable tool in ensuring that users are able to handle quirks easily. Furthermore, this will comprise a judicious mixture of immediate and archival feedback.

The development of the HERCULE prototype is just a first step to providing software which is usable after unanticipated events during application use. The techniques of monitoring user actions, system responses and the relationship between them should prove a fruitful method of advancing the usability of software under modern conditions.

Acknowledgements

The authors thank Phil Gray and Francis Jambon for their contributions towards this work. The authors are grateful to the referees of this paper for their very helpful comments.

[3] Named after *Hercule Poirot*, Agatha Christie's legendary detective.

References

[1] G. D. Abowd and A. J. Dix. Giving Undo Attention. *Interacting with Computers*, 4(3):317–342, 1992.

[2] C. Brod. *Technostress*. Addison-Wesley, Reading, Mass, 1984.

[3] F. C. Brodbeck, D. Zapf, J. Prümper, and M. Frese. Error handling in office work with computers: A field study. *Journal of Occupational and Organizational Psychology*, 66:303–317, 1993.

[4] J. M. Carroll, editor. *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*. MIT Press, Cambridge, MA, 1987.

[5] E. Cutrell, M. Czerwinski, and E. Horvitz. Notification, disruption and memory: Effects of messaging interruptions on memory and performance. In *INTERACT 2001. Eighth IFIP TC.13 Conference on Human-Computer Interaction. 9–13 July*, Tokyo, Japan, 2001. To appear.

[6] A. Cypher. The structure of users' activities. In D. A. Norman and S. W. Draper, editors, *[27]*, chapter 12, pages 243–264. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.

[7] M. Czerwinski, E. Cutrell, and E. Horvitz. Instant messaging and interruption : Influence of task on performance. In *OZCHI 2000. Interfacing Reality in the New Millennium. December 4 - 8*, Univ. of Technology Sydney, Sydney, Australia, 2000.

[8] A. Dix, D. Ramduny, and J. Wilkinson. Interaction in the Large. *Interacting with Computers*, 11(1):9–32, 1998.

[9] S. Draper. Display managers as the basis for user-machine communication. In D. A. Norman and S. W. Draper, editors, *[27]*, chapter 16, pages 339–352. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.

[10] M. Eldridge and W. Newman. Agenda Benders: Modelling the Disruptions Caused by Technology Failures in the Workplace. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, volume 2 of *SHORT PAPERS: Models of Work Practice (Short Papers Suite)*, pages 219–220, 1996.

[11] B. Fogg and C. Nass. How Users Reciprocate to Computers: An Experiment that Demonstrates Behavior Change. In *Proceedings of ACM CHI 97 Conference on Human Factors in Computing Systems*, volume 2 of *SHORT TALKS: A Melange*, pages 331–332, 1997.

[12] M. M. Gardiner and B. Christie, editors. *Applying Cognitive Psychology to User Interface Design*, Chichester, 1987. John Wiley & Sons.

[13] D. Gopher, Y. Greenshpan, and L. Armony. Switching attention between tasks: Exploration of the components of executive control and their development with training. In *Proceedings of the 40th Annual Meeting of the Human Factors and Ergonomics Society*, Santa Monica. 2-6 September, 1996.

[14] N. Hammond. Principles from the psychology of skill acquisition. In *[12]*, chapter 6, pages 163–188. John Wiley & Sons, 1987.

[15] W. Hürsch and C. V. Lopes. Separation of Concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, Massachusetts, Feb.24 1995.

[16] F. Jambon. *Erreurs et interruptions du point de vue de l'ingénierie de l'interaction homme-machine*. Phd thesis, Université Joseph Fourier, 1996.

[17] F. Jambon. Personal communication, May 2000.

[18] G. Johansson and G. Aronsson. Stress reactions in computerized administrative work. *Journal of Occupational Behaviour*, 5:159–181, 1984.

[19] C. Lewis and D. A. Norman. Designing for Error. In D. A. Norman and S. W. Draper, editors, *User Centred System Design. New Perspectives on Human-Computer Interaction*, chapter 20, pages 411–432. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.

[20] K. Lindstrom. Breakdowns and other interruptions in VDT work as a source of stress in customer service and banking. In *Proceedings of the Fourth International Conference on Human-Computer Interaction*, volume 1 of *Congress I: Work with Terminals: HEALTH ASPECTS: WORKLOAD, STRESS AND STRAIN AND IRREGULAR WORKING HOURS; Causes and Measures of Stress*, pages 185–189, 1991.

[21] S. Microsystems. Java foundation classes. Web Document, 1 November 2000. http://java.sun.com/products/jfc.

[22] Y. Miyata and D. A. Norman. Psychological issues in support of multiple activities. In D. A. Norman and S. W. Draper, editors, *[27]*, chapter 13, pages 171–186. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.

[23] S. Mullender. *Distributed Systems*. Addison Wesley, Wokingham, second edition, 1993.

[24] J. Nielsen. *Usability Engineering*. AP Professional, Boston, 1993.

[25] D. Norman. Cognitive engineering. In D. A. Norman and S. W. Draper, editors, *[27]*, chapter 3, pages 31–62. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.

[26] D. A. Norman. *The design of everyday things*. MIT Press, London, England, 98.

[27] D. A. Norman and S. W. Draper, editors. *User Centred System Design. New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, Publishers, Hilldale, New Jersey, 1986.

[28] B. O'Conaill and D. Frohlich. Timespace in the workplace: Dealing with interruptions. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 2 of *Short Papers: Workplaces and Classrooms*, pages 262–263, 1995.

[29] J. R. Olsen. Cognitive Analysis of People's Use of Software. In *[4]*, chapter 10, pages 260–293. MIT Press, 1987.

[30] D. E. Perry, N. A. Staudenmeyer, and L. G. Votta. People, Organizations, and Process Improvement: Two experiments to discover how developers spend their time. *IEEE Software*, 11(4):36–45, July 1994.

[31] J. Reason. *Human Error*. Cambridge University Press, 1990.

[32] K. Renaud and R. Cooper. Feedback in human-computer interaction - characteristics and recommendations. *South African Computing Journal*, (26), 2000.

[33] K. V. Renaud. HERCULE: Non-invasively Tracking Java Component-Based Application Activity. In *14th European Conference on Object-Oriented Programming. ECOOP 2000.*, Sophia Antipolis and Cannes, France., 12 – 16 June 2000.

[34] A. Rizzo, O. Parlangeli, E. Marchigiani, and S. Bagnara. The management of human errors in user-centered design. *ACM SIGCHI Bulletin*, 28(3):114–119, 1996.

[35] M. Rouncefield, J. A. Hughes, T. Rodden, and S. Viller. Working with "Constant Interruption": CSCW and the Small Office. In *Proceedings of CSCW '94*, pages 275–86, Chapel Hill, North Carolina, October 22–26 1994.

[36] G. Salvendy and M. J. Smith, editors. *Advances in Human Factors/Ergonomics. Proceedings of the Fifth International Conference on Human-Computer Interaction, (HCI International '93)*, Orlando, Florida, August 8-13 1993. Elsevier, Amsterdam.

[37] B. Shneiderman. *Designing the User Interface*. Addison-Wesley, Reading, Massachusetts, 1998.

[38] H. A. Simon. *The Sciences of the Artificial*. The M.I.T Press, Cambridge, Massachusetts, 1969.

[39] C. Speier, J. S. Valacich, and I. Vessey. The Effects of Task Interruption and Information Presentation on individual decision making. In *Proceedings of the 18th International Conference on Information Systems*, pages 21–26, Atlanta, GA, USA, December 14-17 1997.

[40] R. van Solingen, E. Berghout, and F. van Latum. Interrupts: Just a Minute Never Is. *IEEE Software*, 15(5):97–103, September/October 1998.

[41] Y. Waern. *Cognitive Aspects of Computer Supported Tasks*. John Wiley & Sons, Chichester, 1989.

[42] W. Walker and H. G. Cragon. Interrupt Processing in Concurrent Processors. *Computer*, 28(6):36–46, June 1995.

[43] C. Yang and P. Carayon. Effects of computer system performance and job support on stress among office workers. In *[36]*, 1993. volume I.

[44] D. Zakay. The influence of computerized feedback on overconfidence in knowledge. *Behaviour & Information Technology*, 11(6):329–33, 1992.

[45] D. Zapf, F. C. Brodbeck, M. Frese, H. Peters, and J. Prümper. Errors in working with office computers: A first validation of a taxonomy for observed errors in a field setting. *International Journal of Human-Computer Interaction*, 4(4):311–339, 1992.