Standards for Defending Systems against Interaction Faults

Avi Harel
Ergolight Ltd.
6 Givon Str., Haifa 34335, Israel
Tel: +972 54 453 4501, ergolight@gmail.com

Abstract. Interaction faults are operational failures attributed to improper interaction between system components. A special, very important case is when at least one of the system components is a human operator. For example, 60-80 percents of the accidents are due to user errors. This article presents guidelines for preventing and protecting from interaction faults.

Did you ever wonder why interactive systems are typically error-prone and why system development is typically behind schedule? Interaction faults are a main source for productivity and maintenance loss, for customer dissatisfaction, for accidents and eventually, project failure and shortening the system life cycles. Interaction faults are actually the results of design defects. The article shows that the effort required to handle exceptional events is in an order of magnitude more than that for conforming to the requirements for normal operation.

A special, very important type of interaction faults is of state mismatch. The first section presents examples of various types of interaction faults, and the design defects that caused these faults.

Standards are essential for defending against user errors, because developers often feel irresponsible for user errors and typically avoid fixing the design mistakes that enable these errors. Yet, many of the standards proposed here may also help for the interaction between non-human system elements.

An interaction fault is preceded by a trigger, which is a deviation of one of the interacting subsystems from the normal operational procedure. The article provides guidelines for preventing triggers attributed to user errors of two sources: usability mismatch and user slips. Few methods for preventing mode errors were tested using *ModeTester*, an experiment interaction specification tool by *ErgoLight*.

To detect interaction faults, the system may include an implementation of a protocol formalizing the operational procedures. The standards should provide guidelines for specifying the system behavior in cases of deviations from this protocol, in order to avoid the unpredictable results of operating in the unexpected circumstances.

From each incident, we can learn. The standards should include guidelines about enabling learning from incidents and from accidents.

Recently, the Standards Institute of Israel (SII) joined the international effort for usability standardization. Besides adopting international usability standards, which target usability practitioners, the Israeli chapter initiates standards that will target system engineers as well. In particular, the intention is to elaborate and formalize the guidelines and methods presented in this article, to help system engineers defend the system against interaction faults.

More about interaction faults:
http://www.usability-standards.com/English/Interaction-Fault/

Interaction Faults

Definition of interaction faults

Consider a system consisting of several units, each of them designed and tested according to the requirement specification and each of them working perfectly. Suppose that a first system unit requests an action from a second unit. Suppose that the second unit is in a state in which this function has not been defined, and consequently the system crashes. This is an example of an interaction fault.

> *An interaction fault is an incident of an undesired reaction to an action request, due to violation of a design assumption.*

Example: Machine damage. A production line was designed to operate with coolant valve open. Due to a transient power failure of the controller, the controller invoked an unusual command sequence, and the production line started with the coolant valve closed.

This is an example of an interaction fault due to **procedural disorder**. Both the controller and the unit worked according to the specifications. However, they were not coordinated. The designers wrongly assumed that the system will never deviate from the specified command sequence. The system was not designed to detect and protect from deviations from this sequence. This kind of interaction faults can be avoided by applying standards for scenario-based interaction protocols.

Interaction faults due to state mismatch

A very important special case of interaction faults is of state mismatch, namely, when a system designed and tested to work on specific state patterns happens to be in a state pattern not specified in the requirements documents.

Example: Therac-25 was a radiation therapy machine. It was involved with at least six known accidents between 1985 and 1987, in which patients were given massive overdoses of radiation. At least five patients died of the overdoses . The accidents were due to an interaction fault in a particular operational pattern, in which the system responded too slowly to the operator's commands (Casey, 1998 - Set Phasers on Stun). The accidents occurred when the system activated the radiation beam while in the exceptional state (Leveson, 1985).

This is an example of an interaction fault due to **state mismatch** between two simple system units. Each of the units could work perfectly, according to the specifications, but they were not synchronized. This kind of interaction faults can be avoided by implementing standards for assuring state synchronization.

The completeness problem: When the system state is unexpected, then the system response is unpredictable. The reason for this is the completeness problem, namely, the failure to consider the risks of all state combinations. Exceptional states are typically defined by combinations of elementary states. The number of all possible combinations is an exponential function of the number of elementary states. This is the well know completeness problem, which states that it is impractical to expect that the design will consider all possible states.

Human-machine interaction faults

A most challenging special case of interaction faults is when the operators are regarded as system components. For example, suppose that the user of an interactive

system went through an operational procedure, but missed one of the actions. Subsequently, the system entered an unexpected state and did not execute the proper command. This is an example of a human-system interaction fault.

> *A human-machine interaction fault is a user action to which the system does not respond as the user expects.*

Example: Production waste. A graphical user interface (GUI) enabled the operator of a manufacturing system to control various parameters of several production lines. The design was based on the service-oriented architecture (SOA) paradigm, in which machines and other components are represented by objects, the parameters are represented by properties and the machine operation is implemented by methods that define the system services. A dedicated screen was designed for each object, enabling the operator to set any of the object properties and to actuate any of the methods.

Many operators found this design logical. Sometimes, however, when they changed a production line, they forgot to change all the settings. To work around this problem they developed a verification test, to make sure that all the parameters were set correctly. Subsequently, they prepared check lists of parameters that should be set or reset for each production line. The check lists were implemented in the GUI next version, shortening the pre-production verification cycles significantly.

This is an example of a human-machine interaction fault due to operational complexity, resulting in **procedural disorder**. The designer wrongly assumed that the operator will be very precise with the data entry. The system was not designed to detect and protect from deviations from the precise operation sequence. This kind of interaction faults can be avoided by implementing standards for resetting and setting default values.

The operator as a system element. Traditional system engineering (SE) practices include measuring and considering the system performance and reliability, regardless of the system operators. This approach leads to wrong decisions about the system architecture and features, as the users typically impose significant limits on the system performance and reliability. For example, 60-80 percents of the accidents are attributed to human errors (Perrow, 1984). In order to model these limits, modern SE methodologies include the users in the system boundaries, regarding the users as intelligent system elements. With this approach, human-system interaction faults may be regarded as special cases of general interaction faults.

Human-machine state mismatch

An important special case of human-machine interaction faults is when the operators do not follow the system state. For example, suppose that the user of an interactive system activates a particular control, assuming that it will activate a particular function. Suppose that the user is not aware of the fact that the system is in a state for which this function has not been defined, and consequently the system crashes. This is an example of a human-system state mismatch.

> *Human-machine state mismatch happens when a correct user action turns out to be improper, because the system was in an exceptional state (Norman, 1983).*

A well known example of human-machine state mismatch is in document editing, when the user unintentionally enters capital letters instead of small letters.

Example: Air France Flight 296 was a chartered flight of a newly-delivered fly-by-wire Airbus A320 operated by Air France. On June 26, 1988, as part of an air show it was scheduled to fly over Mulhouse-Habsheim Airport at a low speed with landing gear down at an altitude of 100 feet, but instead slowly descended to 30 feet before crashing into the tops of trees beyond the runway. Three passengers on board were killed. The accident was due to an interaction fault, in which the captain unknonwingly set the airplane to an exceptional state, in which the airplane engines did not respond immediately to acceleration commands (Casey, 1998 – Leap of Faith).

This is an example of an human-machine interaction fault due to **state mismatch**, in which the user was not aware of system being in an exceptional state. This kind of interaction faults can be avoided by implementing standards for assuring the user awareness of changes in system states.

States and Modes

States. States are used extensively in system design to enable functional multiplexing. For example, the Caps keyboard mode is used in text editing to enable using the same letter keys for both small and capital letters. In the example of Therac-25, the radio therapy machine had two operational states: one of electronic beam and the other of X-ray. The states enable using the same machine for two types of radio therapy.

Definitions. Formally, states may be defined as members (options, possible values) of a state variable, which is of enumeration type. In the Therac-25 example, the state variable was Radiation-Type and the states were the Electronic-beam and X-ray.

Modes are a special kind of states, with an additional property: the user should be aware of them. The Therac-25 radiation types were actually operational modes, while the "target" positions were internal states, the existence of which was unknown to the operators.

Complex Interaction Faults

Consider the case of Therac-25 described above. By including the operator in the system boundaries, we actually extended the system to include three system elements: the two system units and the operator. Theoretically, to assure state compatibility, the operator might need to be aware of the state of each of the system units, and of the fact that they do not match. This approach leads to undesired operational complexity, increasing the chances for further state mismatches.

Example: Unintentional missile launch. Consider the case of a military control system exercising missile operation. Assume that at the beginning of the exercise all units are set to Exercise mode, in which Fire commands are interpreted as user request for launch simulation. Also assume that at startup and reset, all units are automatically set to the Operational mode. Consider the case of a transient power failure in one of the missile units while in Exercise mode. On recovery, the missile mode is automatically reset to Operational. A Fire command at this stage should result in the unintentional missile launch.

This is an example of an interaction fault due to state mismatch between two subsystems in a complex system. Each of the sub systems has its own operators. Assuming that each of the operators was aware of the local state, the subsystem state mismatch resulted in the operator's state mismatch. Theoretically, this kind of interaction faults could be avoided by implementing usability methodologies for

assuring the user awareness of changes in the states of all subsystems. However, this example also shows that sometimes it may be better if we keep the users outside the system boundaries, and let the subsystems synchronize automatically, without imposing extra workload on the user.

Additional examples

Additional incidents exemplifying the sources of interaction faults are available at:
http://www.usability-standards.com/English/Interaction-Fault/Motivation/

The examples include case studies of incidents and accidents in the various applications:

- Safety and disaster control.
- Security.
- Resource efficiency.
- Sustainable development.
- Quality of life.

It should be noted that most of these examples involve several failure modes. The complexity of interaction failures explains why their causes are typically disputed.

The need for standards

Standards can help to defend against interaction faults such as those described above. Many of the methods and guidelines proposed here may also help in the case of non-human elements. Nevertheless, standards are the only means for defining responsibilities.

It is commonly agreed that when the interaction is between non-human system elements, it is the developer's job to prevent faults and to protect the system when they occur. However, when one of the system elements is a human operator, people tend to accuse the interacting person for the fault, and thus avoid fixing the design mistake. Therefore, standards are critical for human-machine interaction validation.

Human-machine interaction faults and user errors

People often confuse human-machine interaction faults with user errors. We can argue that the user activated the wrong control, or we can argue that a design defect enabled the user to activate the wrong control. Both are correct.

The distinction between the two terms is in responsibilities, which impinges on our willingness to defend against them. The term 'interaction fault' is effectual; it suggests that the action could have been right, and that it is the system response that might have been wrong, implying that we could have avoided the fault. The term 'user error', on the other hand, is fatalistic, therefore ineffectual; it suggests that the user action was wrong, and since we cannot control the user behavior, the only thing we can do is punish the user. Standards can make it clear that user errors can be avoided and that it is the developers' job to prevent them.

Manufacturer responsibility

In case of an human-machine interaction fault, people are inclined to blame the user, the system operator. This natural, yet primitive approach is convenient for the manufacturer, but the result of blaming the user is that the real sources for the fault are not examined.

Example: leaking coolant fluid. Consider what happens to your car engine when driving with the coolant fluid leaking. After it runs out of liquid, the engine starts warming, but the engine thermometer does not indicate it, because it measures the fluid temperature, which is not there any more. Thousands of people experience this problem daily, but the car industry persistently disregards the problem, because nobody forces any standards about the usability of warning signals for car engines.

Example: cable TV remote control (clicker). People at home waste their time, and pay for customer support, because they unintentionally change the TV setup or turn off the cable converter. Presumably, millions of people experience this problem nightly, but the cable TV industry disregards the problem, because there is no standard yet about preventing state mismatch.

Standards can provide guidelines to manifest the manufacturer's responsibility on usability design deficiency.

Accident investigation

In case of interaction fault that ended up in an accident, the investigators typically seek to charge and execute the user, instead of examining the sources for the failure. The advantages of this approach are clear:

1. The focus of blame shifts away from the safety officers
2. In case of casualties, blaming the operators provides fast and easy relief from the anger and fear from similar accidents
3. It helps the operators confront their feelings of being guilty.

The reasons why this is ineffectual approach is commonly acceptable are:

1. In case of an accident, investigators can rarely retrieve the exceptional state and the critical event that triggered the accident.
2. Operators are not aware of the details of the system operation in emergency, and therefore cannot show that the source of the problem was in design mistakes.
3. Operators on duty typically feel responsible about ensuring safe operation.

Besides the humanitarian problem of blaming the victims, this approach is also problematic in that it shifts the attention from the causes for the accident, and therefore disables investigating the design mistake, investigation that should prevent the next accident (Norman, 1990). The result is that accidents often repeat.

Example: A320 crashed in Bangalore, India. Because the committee that examined the Air France Flight 296 accident found the captain responsible and guilty, they did not examine the defects in the airplane design. Consequently, in 1990, another A320 crashed in Bangalore, India, for the same design mistake (Casey, 1998 - Leap of Faith).

Example: PCA infusion pump. Nurses' complaints about errors in using were disregarded, resulting in 65-650 deaths. The litigation resulting from an investigation of a particular accident pointed to the nurse and the hospital rather than the device manufacturer (Sheridan and Nadler, 2006), enabling increasing the death toll.

Standards can provide guidelines to regard the operators as victims rather than blaming them for the accident, and to investigate the design flaws that enabled the faulty situation.

Incident management

People investigate accidents instead of incidents. An incident is a sequence of events that could result in an accident. Most accidents were preceded by hundreds of

incidents that could have been analyzed and subsequently enable preventing the accidents. Standards can provide guidelines to capture incidents and investigate the design flaws that enabled the faulty situation, which fortunately did not end up in an accident.

International usability standards

Standards are a formal agreement on specific, detailed topics. They allow us to codify best practices or a set of requirements, and share them across industries, national boundaries and disciplines. (http://www.upassoc.org).

Current usability standards are intended for use by usability professionals. Consequently, they do not appeal to system engineers. Some standards focus on processes, describing principles and making recommendations for how to achieve a result. Others are detailed specifications, and contain requirements that must be met. For a complete list, see UsabilityNet (Bevan, 2001).

Limitations. Not only system engineers disregard usability standards. Most usability professional do not recommend to the project managers to use them. Possible reasons for this are:

- **Quantity**: there are too many standards, and it is difficult to find the one that you really need
- **Maturity**: usability engineering is a new disciplines,
- **Relevance**: the validity of the standards is limited to certain scenarios
- **Arbitrariness**: some of the standards are set arbitrarily, rather by well-established design principles (example, http://www.cja-jca.org/cgi/content/full/50/3/221)
- **Technology-dependence**: some of the standards are technology specific, and therefore become obsolete after few years.
- **Added value**: it is not clear how the standards practically contributed to the customers (possible reason: http://www.taskz.com/ucd_righi2_indepth.php).

Standardizing the Defenses

Goals

Standards about interaction faults are required to help system engineers with the following tasks:

- Define the defense strategy
- Prevent interaction faults
- Provide means to capture incidents of interaction faults
- Define the requirements for usable alerting about exceptional or risky states
- Define troubleshooting procedures
- Define how to recover and reset the system in case of interaction faults
- Prevent emergency operation errors
- Provide means for accident investigation.

Defining the defence strategy

A commonly accepted defense strategy is the one presented by Reason (1990), who recommended a Swiss cheese model of preventive and protective layers. The defense layers are presented in the following table:

Defense Layer	Guideline examples
Prevention	
1. Complexity reduction	• Reduce features, options, states • Resolve state dependency
2. Prevent exceptional states	• Scenario based control design
Protection	
3. Detect exceptional states	• Implement state transition model
4. Fault tolerance	• Safety net
5. Alerting	• Alert design methodology
6. Troubleshooting	• Troubleshooting methodology
7. Recovery	• Master-slave protocols • Assuring risk awareness • Resynchronize states
8. Emergency operation	• Instructional dialog
Investigation	
9. Incident management	• Incident logging and prompting
10. Accident investigation	• Investigation tools

Preventing interaction faults

People often confuse faults with their triggers. It is therefore important to specify the typical sources that trigger interaction faults. In order to prevent the interaction faults we need to identify and classify the sources that trigger them, and to provide methods and guidelines for disabling these triggers.

Triggers. An interaction fault may be the result of a unit fault or a user error.

A unit fault can trigger an interaction fault when another unit fails to react to the original fault. The Air France Flight 296 accident and occasional engine overheat incidents described above are examples of interaction fault triggered by unit faults.

User errors are user actions that do not match the designer's intention. The Torrey Canyon accident, the Therac-25 accident, cable TV setup mishaps and production waste described above are all examples of interaction faults triggered by user errors.

Sources of user errors include:

- **Usability mismatch**: when the user's intention does not match the designer's intention. Historically, the term was **user mistakes** (Norman, 1983), but this has changed in support for the methodology of incident investigation.
- **User slips**: when the user's action is not as the user intended (More about this in: http://www.usability-standards.com/English/Interaction-Fault/Triggers/User-slips.htm)

Preventing usability mismatch. Usability mismatch should be defined and examined in the context of operational procedure. Main guidelines for usability assurance:

- Keep it Simple Stupid (KISS):
  - Simplicity: Removing unnecessary features and options reduces the chances for subsystem faults and user errors.
  - Stupidity: Removing unnecessary dependencies on states reduces the chances for activating the wrong function.
- State independence. Defining user commands that are independent of the system state. Methods for this include:
  - Event threading by control duplication. Resolve state dependency by transforming conditional events to unconditional events.
  - Virtual controls. Make use of auxiliary 'Shift' controls.
- Eliminating redundant UI states (subsequent to removing state dependencies)
- Automatic dependency termination, by time out or following a normal event.

Preventing user slips. Guidelines include:

- Scenario-based control. Avoid exceptional state combinations by operational procedures
- State independence, as above
- Avoid shortcut controls. Shortcut controls typically used for debugging are error-prone, therefore they should be disabled on the system release.

Protecting from interaction faults

Interaction faults cannot always be prevented. For example, we cannot prevent exceptional states resulting from a fault of a system unit. Since the system behavior is well-defined for scenarios only, the system response to the next event might be unpredictable. For example, an accident of friendly air strike in December 2001 in Afghanistan was due to automatic coordination reset after battery replacement (Sheridan and Nadler, 2006). Also, complex systems such as flight control systems require that the operation is state dependent. In these cases, incidents of state mismatch are inevitable, because sooner or later the user will forget to regard the current state

The standards for protecting from interaction faults should include guidelines for detecting and identifying interaction faults, for tolerating and recovering, for emergency operation and for investigating the incidents.

Detecting interaction faults. How can the system detect interaction faults?

By definition, an interaction fault is an incident of an undesired reaction to an action request, due to violation of a design assumption. Therefore, to detect interaction faults the system needs to:

- Compare its state and behavior to the design assumptions, and
- Decide that a particular reaction is undesirable.

This means that the system needs to store a model of its own behavior, which is actually a high-level protocol. This model will define the normal behavior. At run-time the system needs to continuously monitor its own behavior, by comparing to the protocol. Also, this means that the system needs to store rules describing design assumptions about the existence and range of various setup parameters at each stage along the protocol.

Guidelines

- Apply scenario-based design to define operational procedures

- Formalize the operational procedures as a protocol, a model describing the system interactions, and the accompanying state transitions
- Provide a means to record the actual interaction
- Provide a means to verify that the actual interaction and states comply with the protocol.

Tolerating interaction faults. Scenario-based design enables solving the completeness problem by taking the positive route; instead of specifying all exceptional state combinations (the negative route) we just specify the operational procedures for normal operation and for troubleshooting, corresponding to the scenarios. This type of specification is of low (linear) complexity, compared to the exponential complexity when taking the negative route.

To specify the system behavior for all possible state combinations, we need to classify the states according to the desired system reaction (e.g., reject request, prompt for confirmation, beep, etc.). The design may include several layers of safety nets. At the top there is the protocol. At the bottom there is the default response to unknown situations. The default response may be to halt the system, when in testing, or to revert to a previous state, if it does not make sense to halt the system (such as in a transportation system).

Alerting. This is an important feature in safety-critical systems. Consider the Torrey Canyon supertanker accident. The state mismatch occurred after the navigation system was unintentionally set to the Control state, a state intended for use in maintenance only. However, the last straw was the fact that the captain did not notice the exceptional state. Similarly, were the "engine idle" mode of the Airbus A320 plane highlighted, the captain of Air France Flight 296 could have noticed it and reaccelerated before it was too late.

Guidelines on alerting should include (Harel, 2006):
- Provide default feedback about exceptional states, assuming that the operator is not attending the state change
- Indication about exceptional states should be active, as opposed to passive, as was the case with the Torrey Canyon accident.
- Two stage alerting: sound for drawing attention, visual for exploring the situation
- When to sound: balancing the risks of missing a sound vs. false alarms
- Adapt the scenario model to accommodate disturbing sounds.
- Audibility assurance: considering variant levels of environmental noise
- Choosing the proper type of sound: speech, tunes, etc.
- Choosing the proper tones: pitch, volume, etc.
- Information encoding in sound about hazard proximity and severity
- Sound reliability: the case of alarm turned off …

Fault verification. Suppose that the system has detected an instance of deviation from normal interaction. The reason for this may be an interaction fault, or a need to deviate from the normal procedure in order to solve a problem. How can the system tell which is the case?

To decide how the system should react to a detected deviation from the procedure, we need to consider two cases about the subsystem that triggered the unexpected event:
- subsystem is dumb and cannot tell if the event was due to a fault
- The subsystem is intelligent and can inform the recipient about the intention.

The case of a dumb sub system. Suppose that system cannot decide if the exception was due to a fault. How should the system react? There is no single answer to this question, but there is a guideline: The system design should include a definition of a default behavior for such cases. Example of default behavior can be: halt the system, beep, log the event, generate a report, alert, ignore, etc.

The case of an intelligent sub system. Suppose that the system has a way to either reject or confirm the response to the exceptional event. How should the system react in case of rejection and in case of confirmation? In case of rejection, can the system roll back to the situation before the reception of the exceptional event? If the system cannot roll back, how should it proceed? In case of confirming the exceptional event, how should the unfinished procedure terminate? Again, there is no single answer to these questions, but there are several guidelines.

Guidelines:

- The system design should include a definition of the default behavior in case of rejection of the exceptional event.
- The system specification should include a definition of rolling back from undesired states
- A default rolling back rule should be specified for cases of unexpected states
- The system specification should include a specification of the data that should be reset when rolling back
- In case of confirming the exceptional event, the system should always start a new procedure
- The specification should include definition of all the data that should be reset. and their default values

The case of a human operator. We are not allowed to just ignore all events when in the exceptional state. Users, just like all other system components, are error generators. Even after doing everything to prevent user slips, they still slip. The only way to stop them from slipping is by keeping them away from the system, but unfortunately, we need the users to control the system, to monitor, to find problems, to report about them and to fix them. We need the operator's skills to find the proper solution, which may involve means beyond the system implementation. On the other hand, when in an exceptional state, the system response to the next action might be unpredictable. The design challenge is to define how to respond to the operator commands in such cases. The challenge is to find out which of the user actions is erroneous and which is intentional. The guideline is that the system should provide means for the decision maker to decide whether to proceed with the procedure execution or to cancel the unexpected event

User error detection. In a service-oriented architecture, such as in the examples of cable TV or production waste above, the user can access any service, any time. Nothing in the user interface suggests that a particular action might be erroneous. However, if the user interface is designed according to scenarios, then at each particular point of each procedure, we expect only certain actions from the user. Any unexpected action may be due to either a slip or an intentional deviation from the procedure.

To conclude that the user action is by mistake, we need to ask the users about their intention. Typically, we use a dialog box to confirm that the suspected action was intentional. If they confirm that the action was intentional, we need to first make sure that the system can obey to their command, and then to make sure that the users

indeed wish to terminate the current interaction.

Troubleshooting. Users of software programs are typically being frustrated from error messages such as "System error", raised by exception handlers, because they do not know what to do in order to solve the problem and how to avoid it in the future.

Example: the Three Miles Island (TMI). The Three Mile Island nuclear power station accident (http://en.wikipedia.org/wiki/Three_Mile_Island_accident) was the most significant in the history of the American commercial nuclear power generating industry. The accident was exacerbated by wrong decisions made because the operators were overwhelmed with information, much of it irrelevant, misleading or incorrect. The TMI accident revealed the need for helping the users find the source of warning messages.

Example: chemical processing. Consider the example of a tank used for some chemical processing, with sensors for temperature, pressure and PH. The result of hazard analysis may indicate that leakage from one of the valves should raise the temperature and pressure and lower the PH. If the same valve is stuck closed, the temperature should raise and the pressure and PH should get lower values.

Hazard road map. Similar data, with different results, may be obtained about other valves. If the tank leaks, the temperature and pressure may decrease and the PH would remain unchanged. What we get is a map of trends in sensor data due to hazards. We can use this map at run time to direct the operator to the sources of warning messages.

Guidelines:

- The system should guide the operators in finding the trigger for the exceptional activity
- The system should indicate the source of the interaction fault
- Avoid sensor-level warnings. Apply a hazard road map to provide a single warning, indicating the source for the interaction fault
- The warning message should include all the details required for fixing the problem.

Recovery. Sometimes, the feedback message may include instructions for recovery from the exceptional state, such as for restarting or rolling back to an earlier normal state. The instructions may depend on the particular action, but also on particular values of some of the state variables.

Guidelines:

- Provide a master-slave model to handle state resynchronization
- Define a behavioral architecture to handle multiple instructions
- Provide risk information for options in the recovery procedure
- Arranged the recovery instructions in order of priority
- Inform the operators about the risks of resuming normal operation.

Emergency operation. How will the operators find the features they need to handle unexpected situations?

Typically, because the situation is rare, the operators may not have experienced the part of the UI used to handle the risky situation. This means that this part of the UI should guide the operators about the options they can take. Because at this stage the operators might be in stress, they might not see the forest through the trees, therefore, information overload should be avoided. The conclusion is that this part of the UI should compose of wizards, enabling the operators' access to a minimal set of

options. Wizard forms allow the user to specify parameters gradually, and to go back to previous forms before submitting the recovery command.

On the other hand, it may be the case that the operators know how to recover from the emergency situation. This means that beside the wizards, the UI should provide visibility and direct access to critical features.

Additional Guidelines:

- Provide wizards to drill-down to the problem solving
- Enable operation in training mode, so that the operators have the chance to experience the system behavior in unexpected situations
- The system design shall not set default values for data that the human operator needs to enter manually.

Investigation

From each incident, we can learn. The basic means is by logging the system activity and states. The logger may be the same as the one used for identifying the incidents, by comparing the system activity with the model.

It is very easy to include a logger in the system, but special means should be added to extract information valuable for the investigation.

Guidelines for incident investigation:

- Provide an incident reporter, with statistics about repeating incidents, classified by hazards
- Provide a search engine enabling filtering the log files by key features and selecting incidents for investigation
- Provide a back tracking visualization means showing the concurrency of state transitions (e.g., in form of GANTT)
- Provide an incident manager, enabling to record and back track aggregates of incidents in form of design problems.

Standards Development

Proof of concept

The methods and guidelines described here are generic, and their validity limits are well recognized, which means that they may be formulated as industry standards. Few of the methods proposed for avoiding state mismatch were tested using *ModeTester*, an experimental interaction specification tool, developed by ErgoLight Ltd. This tool enables specification of the system structure, UI layout, control usage and operational procedures and to report on violation of the guidelines.

ModeTester represents the system structure by a component tree, where each node can hold a simple state machine. State hierarchy is defined as links from the states to components that they enable, and state charts are projected from this structure by ignoring the components.

The operational procedures are elaborated based on scenarios. The procedures are defined by links between UI controls. The links have a Logic-Type property, enabling specifying serial order, parallel sequencing and options. Actions may depend on conditions, defined by the states.

Finally, *ModeTester* generates reports with use cases describing the ways the functions are used, and with layout usage, describing the ways the UI elements are used. The reports provide warning about UI controls that are overloaded in ways that

might result in state mismatch.

To check the feasibility of using tools to prevent state mismatch, we used *ModeTester* to specify the operation of a cable TV remote control set (http://www.hot.net.il/HOT.aspx?docID=1686&FolderID=631). The results indicate that basic methods and guidelines described in this article can be formulated as standards.

Interaction validation

New initiative. Today, there are no standards for either preventing or protecting from interaction faults. Recently, the Standards Institute of Israel (SII) joined the international effort for usability standardization. Besides adopting international usability standards, which target usability practitioners, the Israeli chapter initiates standards that will target system engineers as well. In particular, the intention is to elaborate and formalize the guidelines mentioned in this article, to help system engineers defend the system from interaction faults.

The plan. System engineers should benefit from the following features:

- Proof of added value: the standards for interaction validation are examined by benchmarks, based on the examples described in this article
- Validity assessment: guidelines will be stated together with their validity limitations
- User-centered design: the standards will guide system engineers about the relevant human factors and how to apply them in scenario-based design
- Test-time features: the standards will guide the engineers about the way to include features that are useful for the testing stage, so that they will not hamper usability of the released version.
- Interactive standards: to facilitate using the standards, they will be arranged as html pages, with links that will enable navigation to explanations about guidelines, human factors, validity limitations etc.
- Demonstration: a demo program should be developed and released with the standard, enabling the customer to experience the meaning of the instructions.
- Design tools: based on the experience obtained by using *ErgoLight ModeTester* it is clear that key methods and guidelines presented here can be integrated in tools for system specifications, to eliminate interaction faults by design.

Pilot project. A committee of experts is working these days on new standards for ensuring the usability of sound alarms by medical equipments. An interactive version of the commonly accepted standard 60601-1-8 revealed the need for rewriting it. A demo program has been developed, demonstrating better ways for alerting about medical emergencies.

Conclusion

Did you ever wonder why interactive systems are typically error-prone and why system development is typically behind schedule? This article shows that the effort required to handle exceptional events is in an order of magnitude more than that for conforming to the requirements for normal operation.

Interaction faults are a main source for productivity and maintenance loss, for customer dissatisfaction, for accidents and eventually for shortening the system life cycles. The article describes and analyses several examples of interaction faults. The article provides methods and guidelines for preventing these failure sources, and for

protecting from those that were not eliminated. Few of these methods for preventing interaction faults caused by state mismatch were tested using an experimental tool. New standards are being developed in the SII, intended to formulize these methods and guidelines for the benefit of system engineers, to improve system safety, productivity, reliability and ease of use.

References

Bevan N., 2001, International standards for HCI and usability, *International Journal of Human-Computer Studies*, Volume 55, Number 4, pp. 533-552(20), Academic Press (available at: http://www.usabilitynet.org/tools/r_international.htm)

Casey, S. "Set Phasers on Stun", Aegean Publishing: Santa Barbara, 1998

Dumas, J.S. and Redish, J.C., "A Practical Guide to Usability Testing", Exeter, England; Portland, Or.: Intellect Books, 1999

Harel, A., "Optimizing the user interface for intensive usage", People and Computers, Aug. 1987 (Hebrew)

Harel, A., "Alarm Reliability", User Experience Magazine, Vol 5., Issue 3., 2006

Harel, A., "Defending systems against human errors", The fourth Incose-Israel conference, Herzelia, Israel 2007 (Hebrew)

Leveson, N., Safeware: System Safety and Computers, Addison-Wesley Prof., 1995.

Norman, D.A., Design Rules Based on Analyses of Human Error. *Communication of the ACM.* 1983;26(4):254–258.

Norman, D.A., Commentary: Human Error and the Design of Computer Systems, Editorial published in Communications of the ACM, 1990, 33, 4-7.

Perrow, C., Normal Accidents: Living with High Risk Technologies. NY: Basic Books. 1984.

Robert, D., Berry, D., Isensee, S. and Mullaly, J., Designing for the User with OVID: Bridging User Interface Design and Software Engineering, Software Engineering Series, Macmillan Technical Publication, 1998.

Sheridan T.B., and Nadler, E.D., Review of Human-Automation Interaction Failures and Lessons Learned. October 2006 (DOT-VNTSC-NASA-06-01) http://www.volpe.dot.gov/hf/docs/ha-failures-sheridan.doc

Avi Harel. A mathematician, founder and active manager of Ergolight Ltd. Formerly, a software engineer, a system engineer and a human-factors engineer of main projects of Rafael, the Armament Development Authority of Israel. The inventor and chief developer of Ergolight award-winning tools for usability diagnostics based on logs of the users' activity. Currently, a board member of the Israeli chapter of the Usability Professional Association (UPA), and the chair of the Technical Committee for Usability of the Standards of Institute of Israel (SII).