Enabling Seamless Operation: Design Challenges

Avi Harel

August 29, 2024

Enabling Seamless Operation: Design Challenges

Abstract. The article examines methods for facilitating the operation of socio-technical systems. Operational failure is due to diversion from procedures and situations defining normal activity. Sources of operational failure include triggers, exceptions, and uncoordinated activity. The article presents a model comprising three layers of the visibility of operational failure: incidents, errors, and accidents. The principles and methods discussed here focus on the need to maintain coordinated activity, and to mitigate the risks of exceptions. The protection from failure should be based on generic rules describing normal situations and activity. A model of HSI proposed here consists of interaction cycles, each comprising a supervisor, controllers, and service processes. Human decisions should rely on forecast of the system behavior in response to optional decisions. The preview information may be obtained by behavioral twins.

Motivation

Operational waste

The primary goal of Human System Integration (HSI) design is to enable seamless operation. To achieve this goal, we need to understand and overcome the sources of operational waste, namely, barriers to seamless operation.

We should assume that the number of risky situations is huge, because we can see only those that are costly, and because we do not bother to detect and investigate low-cost events. In most of the systems of interest (SoI), most of these barriers are unknown, because neither the designers nor the customers can notice them, and therefore they are not aware of them. In case of an incident, when they notice operational waste, they often attribute it to an operator's error. They are obliged to pay attention to a barrier only in case of accident, namely, when the costs are extremely high. According to various statistics, accidents are commonly attributed to human errors (cf. Zonnenshain and Harel, 2015).

Systems of interest

The systems in scope of this discussion is any utility-critical system. The system utility may be defined as the value of performance minus the costs of operational waste. Operational waste is inversely correlated with the system usability, which is a key factor affecting functionality, safety, productivity, and consumer satisfaction. We may classify the systems of interest according to the rate and costs of incidents as follows:

- Low-rate, high-cost incidents are commonly called accidents. Traditionally, designers attribute accidents to force majeure or bad luck (Bloch, 1977; Taleb, 2007)
- High-rate, low-cost incidents are commonly called errors. Traditionally, system designers do not admit making the design mistakes, and the operators are likely to attribute the failure to their own errors (Norman, 1983)
- High-rate, high-cost, as in medical treatment, intensive wars, or any other disaster.

Understanding errors

All systems are always operated under the risk of unknown hazards, and these hazards are unexpected (Taleb, 2007). When the failure costs are high, we typically attribute them to operator's errors (Hollnagel, 1983). The term human error is just a name for operational failure that the human operator was not able to prevent, driven by the accountability bias (cf. Dekker, 2007).

To eliminate human errors, we need to understand how they develop. To be on the safe side, we should protect the system from all risky situations, because we cannot foresee which of them might become disastrous. Practically, this implies that error proofing ought to be a key topic of systems engineering. The challenge is to get enough evidence to understand how system operation fails.

Evidence base

To get the evidence about the sources of failure, we need to define when a situation may be considered as normal, and to embed special probes and tools in the system design, to sense, trace, and analyze the system activity (Harel et al., 2008), and to provide reports about exceptional activity (Harel, 1999, 2009). These extra means are costly, because the system activity is complex, and are affordable only in wealthy domains, such as aeronautics and astronautics, and high-risk industries such as nuclear power production.

Another barrier to capturing exceptional activity is the accountability bias, namely, people reluctance to gather evidence about the source of failure (Dekker, 2007). The designers' interest is to underestimate the costs of errors, and to overestimate the costs of exploring the sources of the errors. Typically, designers are likely to compromise operational risks, preferring to provide explanations such as force majeure (Harel & Weiss, 2011) and Murphy's Law, over doing in-depth root-cause analysis (RCA).

Reactive RCA

In traditional RCA of a mishap, we want to identify a single trigger of the chain of events. Often, we realize that besides the trigger, there are other risk factors, typically, special conditions that enable the undesired effect of the trigger. In reactive RCA, we often look for flaws in the development practices. For example, the RCA of the classical Therac-25 accidents indicated 12 engineering problems (Leveson, 1983, 2017). These findings are circumstantial, based on safety thinking, which is a description of how the system should have been developed, in hindsight. They might suggest to the development team how they could do better engineering, but they do not teach about how to prevent similar accidents in future systems, in other domains. In proactive engineering, the goal is to prevent such mishaps by design.

Proactive RCA

A proactive version of Murphy's Law is that errors result from design mistakes, and therefore failure should be prevented by design. Error proofing should be based on a special model of root-cause analysis (RCA). To extend the findings to other domains, we need to employ system thinking, rather than safety thinking. A way to implement system thinking in the design of safety-critical systems is by employing the System Theoretic Accident Methods and Processes (STAMP) proposed by Leveson (2004), based on the theory of cybernetics (Wiener, 1948).

Model-based RCA

A meta-RCA indicates that system failure is often the outcome of a two-stage process: first, an unexpected trigger diverts the system situation from coordinated to exceptional, and the costs are due to a later action, which could be expected, should the system be in the original situation, as it was prior to the first trigger. The normal action becomes unexpected because the system situation is exceptional. Model-based RCA implies that the analysis assumes the two-stage model as above.

The principles of cybernetics apply not only to safety critical systems, as proposed originally, but also to all utility-critical systems. For example, model-based RCA is more effective than the traditional RCA for describing complex operational failures, such as mode errors following unintentional activation of shortcut keys (Harel, 2009). Therefore, the STAMP approach may be extended to a System Theoretic Utility Methods and Processes (STUMP) approach.

Operational risks

According to the failure model described above, the operational risks may be classified as triggers, situational exceptions, and activity risks. These categories are described here:

Triggers

A triggers may be actuated either by a human operator or by a technological unit. Human operators are error prone: they are likely to generate errors. A human factors version of Murphy's Law is: "if the design enables the operators to fail, eventually they will". Technological units do not generate triggers frequently, because they are identified and prevented following the system verification. A specific technological trigger is intermittent power failure, followed by automatic setting of a default mode, which does not comply with the system situation, as demonstrated in the GPS mode error causing the friendly fire accident in the war with the Taliban in December 2001 in Afghanistan.

A human trigger may be either unintentional or due to wrong decisions. Unintentional human triggers, such as in the B-17 accidents in WWII, or the lever setting to the maintenance-only Control position in the Torrey Canon accident, are called slips (Norman, 1983).

In the context of HSI, the system may support decision making by providing the human operators with clear and comprehensive information. This necessary information should include static prediction of the system situation, as well as exploratory prediction of operational options. This information should be provided gradually, according to the needs for decision making, employing Human-Centered Design (HCD) principles, and considering the mental capabilities of human operators.

Human triggers are challenging, because humans are embedded in the system to solve problems in emergencies, when operating in exceptional situations, which are unseen at design time. Unfortunately, it turns out that these expectations from the human operators are not realistic, because human operators are likely to follow their training, which was in normal conditions. According to the "irony of operation", in emergency, the operators are likely to react as trained in normal operation, instead as by calm, logical decision making (Bainbridge, 1983). For example, if the system frequently prompts the operator to confirm a particular risky action, then the human operator is likely to confirm the prompt inadvertently, before evaluating its applicability, as is typically expected by the designers.

Situational risks

The system situation may be defined as the aggregation of the active states of all state machines. A situational risk may be classified as either external or internal. In normal operation the system situation must reside in the performance envelope. A situational risk is classified as external when the system is approaching the performance boundaries, which are defined by limits of performance variables. Internal situational risks are those due to exceptions, namely, diversion from the situations defined as coordinated.

The number of possible situations grows exponentially with the number of state machines employed in the system operation; therefore, it is challenging to maintain situation coordination when the situation changes dynamically. Special coordination techniques, such as assigning the situation to scenarios, must be employed to maintain situation coordination.

Common significant situational risks are associated with the availability and accessibility of critical resources, such as features or processes. For example, if a utility critical feature, such as a backup facility, is disabled or inaccessible in functional operation, then the operation might fail due to an over-constrained (alpha type) design mistake. Typically, Loss Of Control Accidents (LOCA) such as in AF447 and IA605 airplane or the Torrey Canyon accidents, are due to improper disabling access to a control feature. On the other hand, if a risk critical feature, such as reset or restart, is enabled or is accessible in the wrong scenario, then the operation might fail due to under-constrained (beta type) design mistake. This kind of failure is typical of consumer product, in which the availability of sexy features is a prominent marketing requirement.

Activity risks

Activity risks are mode errors, namely, failures due to problems in coordinating a unit mode (operational state) with the operational scenario. Often, activity risks are due to enabling operation in either exceptional or fuzzy situations. A situation is regarded as fuzzy if the system design does not include means to identify the concrete situation, namely. For example, many mode errors, such as in friendly fire accidents, are due to assuming the wrong scenario, because was not defined explicitly.

Almost all system units, and almost all system features, are prone to activity risks, due to mistakes in constraining the system situation (cf Harel, 2011). Mode errors are very frequent in the operation of consumer products, in which the design enables access to setup features while in functional operation. Enabling maintenance-only features in functional operation might also result in mode errors, such as the erroneous disabling of the TMI backup pump.

Mode errors are also the primary source of several friendly fire accidents, as well as accidents in transportation systems, such as several Takeoff/Go Around (TO/GA), the AF296, AeroPeru 603, and the Torrey Canyon Loss-Of-Control-Accident (LOCA). Mode errors are also involved in accidents due to operating in transient situations, such as in the Therac-25 accident. Mode errors are typical of design with multiple use cases, with conflicting demands, such as of mode assignment, as demonstrated in the Three Miles Island (TMI) backup pump case. Use cases may also be in conflict when competing on access to a shared resource. Such conflicts are called interlock problems.

Engineering

Forty years ago, the actual costs of software and time to market of software projects were 300% of the plans. Why? Because software development was an art, rather than a discipline. Therefore, according to Standish (1995)

> *"When a bridge falls down, it is investigated and a report is written on the cause of the failure. This is not so in the computer industry where failures are covered up, ignored, and/or rationalized. As a result, we keep making the same mistakes over and over again".*

In the forty years since then software development has matured and became an engineering discipline. HSI engineering should evolve similarly from art to engineering (Harel & Zonnenshain, 2019). Currently, HSI design is still an art, rather than a discipline. To turn utility oriented HSI design into an engineering discipline, we may learn from quality assurance practices, and understand that each system component might fail, and that it is impossible to predict the time when this should happen.

Affordability

A typical system is developed with limited resources of budget and schedule. A key requirement for enabling HSI design is affordability. The development should be based on predefined generic meta rules, which are common across many industries and domains. The generic rules should be based on models of system behavior (Harel, 2021). These generic rules may be customized for specific families of projects. To mitigate the typically limited resources, we should:

1. Develop and validate the meta rules for defining the coordinated situations

2. Develop software tools for customizing the coordination rules
3. Develop software tools and plugins for ongoing verification of the situation coordination in normal and in safe-mode operation: activity tracking, activity analysis, and investigation reporting
4. Develop software tools and plugins for supporting human decision making and troubleshooting.

Model-based HSI design

HSI should be designed hierarchically according to a model of a Socio-Technical System (STS). The top layer in the model is the big picture, with the context, comprising the STS, the stakeholders, and the interactions between them. Then, we should drill down from outside in (Boy, 2013). Each STS may include elementary units and sub-STSs. An elementary unit may be technical, human users and operators, or an AI unit. Typically, the technical and AI units include processes, which interact with each other.

Normal interaction is task driven, comprising a supervisor, one or more controllers, and one or more servers. In normal operation the supervisor processes define tasks for the controller processes, as well as operational scenarios. The controller processes issue commands or requests to the service processes, and the services provide situational and activity reports. The service processes may employ behavioral twins, intended to provide static and exploratory preview information, based on simulation (Luqi, 1989).

The interactions among processes are associated with use case descriptions, which are bottom-up. The supervisor processes may be associated as use cases of the controller processes, and the latter as use cases of the service processes. Processes may be duplicated, to ensure that we have a single use-case per process. This may facilitate the prevention of conflicting mode setting. An interlock mechanism should be employed for elementary processes competing on managing shared resources.

In utility-oriented engineering, as in quality assurance, we assume that we cannot predict the failure of elementary units. Rather, we focus on the interactions, and we assume an OEM model of the elementary units. According to this model, we do not need to know anything about the unit processing to integrate it in the system: we only need to verify that the functions, performance, and situational reports comply with the requirements.

Model-based coordination

A method used to design the coordination between processes is based on the principle of multiple layer defense, as demonstrated by the Swiss Cheese illustration (Reason, 1997). The protection model defines stages of risk implementation: a hazard is associated with the need to become alerted, and a threat is when an action should be taken. The generic rules are arranged in layers according to the generic model of system failure as follows (Harel, 2021):

1. The primary protection layer comprises methods for preventing hazards, by enforcing operation by the rules, by supporting decision making, and by notifying on approaching the protection boundaries.
2. Not all hazards may be detected directly. A second protection layer is about threat detection, such as by tracing state transitions. A method for detecting unexpected situations is by risk indicator, based on segmentation of continuous system variables, such as performance variables, or time measurement of state transitions and of process execution.

3. Not all expected hazards can possibly be prevented. A third protection layer comprises methods for rebounding from exceptional back to coordinated situations, typically, by notifying the operators about a potential hazard.
4. Occasionally, the operators might fail to rebound from the exceptional situation. The fourth protection layer comprises methods for preventing the situation escalation, namely, of the hazard transforming to threats. These methods are about troubleshooting and recovery procedures, and about enforcing collaboration between the operators and the system, while in safe-mode operation.
5. Sometimes, the system design does not include sufficient means for troubleshooting, and consequently the coordination fails. For these cases, the system should apply a last protection layer, which is by employing resilience procedures.

Rule definition

Brute-force definition of situational and activity rules may be too tedious, due to the complexity of native situations. The rule definition may be facilitated when the situation definition is based on scenarios. Scenarios are used as situation vectors, namely, pointers to the set of state machines, thus reducing the situational complexity from exponential to linear.

The situational rules should define the mapping from scenarios to situation vectors. For example, in the Therac-25 accident, there were two operational scenarios: X-ray and E-beam. The situations underlying these scenarios were tray position: in or out, and beam intensity: low or high. The corresponding vectors were

X-ray ➔ (in, high), and
E-beam ➔ (out, low)

This accident was due to operating in an exceptional situation (out, high) during the transition from X-ray to E-beam. The accident could have been prevented, should the system postpone the risky activity until the end of the synchronization, when the transition is complete.

The rules should also define the scenario transition, and the system behavior in the synchronization of the transient scenario, during the transition. Other generic rules should support reacting to risky activity and to diversion. The reaction to risky activity may be by rebounding. The reaction to diversion should be troubleshooting in special safe-mode operation, in which risky activity should be disabled. Yet another group of generic rules is testing support. This should be required to cope with the unexpected. A special test mode should enable faking triggers, uncoordinated situations, and risky activity.

Transdisciplinary engineering

Error proofing is a transdisciplinary activity, coordinating between four disciplines:

- Systems engineering: in charge of defining functions, architecture and performance
- Human Centered Design: in charge of User Interface design, considering human factors
- HSI engineering: in charge of defining the rules for normal operation, and for reacting to exceptions
- SW engineering: in charge of employing the rules in object classes and instances

Conclusions

The article presents three layers of operational failure: exceptions, errors, and accidents. The design goal proposed here is to facilitate the system operation. The principles and methods discussed here focus on mitigating the risks of exceptions. To implement the ideas described here, we should:

1. Develop and validate the meta rules proposed here
2. Develop tools for rule customization, activity tracking, activity analysis, investigation reporting, embedding behavioral twins in the system design, including test support
3. Define software object classes for the processing of supervision, control, and services, in normal and in safe-mode operation, with testability attributes
4. Develop software plugins for scenario editing, rule-based detecting, alerting, rebounding, and troubleshooting.

Censored References

Bainbridge, L 1983, Ironies of automation. *Automatica. 19 (6)*: 775–779. doi:10.1016/0005-1098(83)90046-8. ISSN 0005-1098

Bloch, A 1977, *Murphy's Law, and Other Reasons Why Things Go WRONG*

Boy, GA, 2013, *Orchestrating Human-Centered Design.* New York: Springer. ISBN 978-1-4471-4338-3

Hollnagel, E 1983, Human Error. Position Paper for *NATO Conference on Human Error*. Bellagio, Italy.

Leveson, N Turner, C 1993, "An Investigation of the Therac-25 Accidents," In *Ethics and Computing: Living Responsibly in a Computerized World*, by KW Bowyer. Los Alamitos, CA: IEEE Computer Society Press, 1996. First Published in Computer, Vol. 26. No. 7, July 1993, pp. 18-41.

Leveson, N 2004. A New Accident Model for Engineering Safer Systems. *Safety Science* 42(4):237-270

Leveson, NG 2017. "The Therac-25: 30 Years Later," in *Computer*, vol. 50, no. 11, pp. 8-11, November

Luqi 1989, Software Evolution through Rapid Prototyping. *IEEE Computer. 22 (5):* 13–25. doi:10.1109/2.27953. hdl:10945/43610

Norman, DA 1983, Design Rules Based on Analyses of Human Error. *Communications of the ACM 26(4)*:254-258

Norman, DA 2013, *The design of everyday things*. MIT Press.

Reason, J, 1997. *Managing the Risks of Organizational Accidents*, Ashgate.

Standish Group, 1995, The COMPASS report, *Forbes*.

Taleb, NN 2007, *The Black Swan: The Impact of the Highly Improbable*. Random House Trade Paperbacks.

Wiener, N 1948, *Cybernetics; or, Control and communication in the animal and the machine*. Technology Press, Cambridge.