Essentials of Integration Engineering

Avi Harel, https://avi.har-el.com/

Ergolight consulting, ergolight@gmail.com

Foundations

The systems of interest

The primary goal of systems engineering is to maximize system utility. The system utility may be defined as the value of performance minus the operational risks. Therefore, the primary goals of systems engineering are to maximize system performance and to minimize the operational risks.

The operational risks may be defined in terms of operational waste. The operational waste may be defined as the maximal performance minus the actual performance. Operational waste is inversely correlated with seamless operation. According to prior case studies, the sources of operational waste include operating in hidden incidents. These are exceptional situations, of which the operators are not aware (Harel & Weiss, 2011).

Exceptional situations hamper seamless operation. To minimize the operational waste, the system design should enable seamless operation. This feature, commonly labeled as system usability, 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.

- Low-rate, high-cost incidents are commonly called accidents. Traditionally, designers attribute accidents to bad luck (Bloch, 1977; Taleb, 2007)
- High-rate, low-cost incidents are commonly called errors. Traditionally, the designers do not admit the design mistakes, as the customers agree to attribute the failure to themselves (Norman, 1983).
- High-rate, high-cost, as in medical treatment, intensive wars, or other disaster.

Barriers to seamless operation

To enable seamless operation, we need to understand the sources of incidents, and find ways to eliminate them. Practically, we need to understand the ways incidents are generated, and the ways to recover from them.

Typically, system designers, the customers, and the operators, cannot notice most of the incidents, and therefore are not aware of them. In case when they notice an incident, they often attribute it to an operator's error. They are obliged to pay attention to an incident only in case of an accident, namely, when the costs are extremely high. Still, most accidents are commonly attributed to human errors (cf. Zonnenshain and Harel, 2015). Practically, we should assume that the number of incidents is huge, because we can see only those that are costly, and because we do not bother to detect and investigate low-cost incidents.

Understanding errors

All systems are always operated under risk of unexpected failure, because these risks are unknown (Taleb, 2007). When the failure costs are high, we typically attribute them to operator's errors (Hollnagel, 1983). Analysis of many accidents has shown that the term human error is just a name for operational failure that the human operator was not able to prevent (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 tell when one of them might be 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 of the sources of failure, we need 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 special domains, such as aeronautics and high-risk process industries.

Another barrier to capturing exceptional activity is the accountability bias, namely, 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 capturing exceptions. Typically, designers are likely to compromise operational risks, preferring explanations such as force majeure and Murphy's Law, over comprehensive root-cause analysis (RCA).

Reactive RCA

In traditional RCA we look for a unique trigger for the unfortunate event. 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 engineering, we often look for flaws in the

development practices. For example, the RCA of the classical Therac-25 accidents indicated nearly 12 engineering problems (Leveson, 1983, 2017). These findings are based on safety thinking, which is ad hoc. These findings are circumstantial. They might suggest to the development team how they could do better engineering, but they did 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 are 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 system 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 to exceptional, and then, another activity results in the undesired costly situation. Typically, the second activity could be regarded as expected, should the system be in the original situation, prior to the first trigger. However, because the system situation is exceptional, the second activity is unexpected.

The two-stage model applies not only to safety critical systems, as proposed originally, but also to all utility-critical systems. It is often 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, and activity risks.

Triggers

The triggers may be classified according to their actuators: human or technological. Human operators are error prone. A human factors version of Murphy's Law is: if the design enables the operators to fail, eventually they will. Technological triggers are much rarer because they are captured during the system verification process. A specific technological trigger is intermittent power failure, followed by automatic setting of a default mode, which does not comply with the system situation.

Human triggers may be 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).

Human triggers are challenging, because humans are included in the system as flexible operators in emergencies, to enable coping with exceptional situations unseen at design time. However, they can rarely do it properly, due to their virtue of training-based reaction. 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. For example, if the system design includes a frequently used prompting to confirm risky operation, the human operator is likely to confirm the prompt automatically, before considering its applicability, as expected by the designers.

Situational risks

The situational risks may be classified as external or internal. Externally, normal operation must be in the performance envelope. Internally, the situations must be coordinated. External risks are due to approaching the performance boundaries, defined as limits of performance variables. Internal risks are due to diversion from the situations defined as normal. The number of possible situations grows exponentially with the number of state machines employed in the system operation; therefore, careless design of the situation coordination is error prone. Special coordination techniques, such as scenario-based situation assignment, must be employed to maintain situation coordination.

In normal design, almost all nontrivial system units are prone to situational errors, in terms of accessibility or availability. 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. On the other hand, if a risk critical feature, such as reset or restart, is enabled or accessible in the wrong scenario, then the

operation might fail due to under-constrained (beta type) design mistake. Very common examples of

Activity risks

Activity risks are mode errors, namely, failures due problems of coordinating the mode (operational state) of a system unit with the operational scenario. Often, they are due to enabling operation in exceptional or in fuzzy situations. A situation is regarded as fuzzy if the system design does not include means to identify the concrete situation. A special mode of fuzzy situation, implemented in many accidents, is when the operational scenario is not defined explicitly. In these cases, the activity intended for a particular scenario might be risky in other scenarios.

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 in 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. Another source of mode errors is process design having multiple use cases of mode setting. This might result in conflicting mode setting, as demonstrated in the TMI backup pump case.

Mode errors are also the primary source of several friendly fire accidents, as well as accidents in transportation systems, such as several TO/GA, the AF296, AeroPeru 603, and Torrey Canyon LOCA. Mode errors are also involved in accidents due to operating in transient situations, such as in the Therac-25 accident. A special form of activity risk, known as an interlock problem, is when enabling mode transition by conflicting controllers' scenarios.

Engineering

Affordability

A primary requirement for enabling integration design is affordability. The development should be based on predefined generic meta rules, which are common across many industries and domains. These generic rules may be customized for specific families of projects.

Learning from SW engineering

Integration design should develop from art to engineering (Harel & Zonnenshain, 2019). A model of this transition was proposed for SW engineering. Traditionally, the actual costs and time to market of software projects were 300% of the plans. Why?

> *"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".*

(Standish, 1995).

Model-based integration design

The generic rules are based on models of system behavior (Harel, 2021). A top-level model is of the big picture, comprising the STS, the stakeholders, and the interactions between them. Then, we drill down from outside in (Boy, 2013). Each STS may include elementary units: technical units, human users and operators, and AI units. The technical and AI units include processes, which interact with each other.

In utility-oriented engineering we assume that we cannot predict the failure of elementary units. We focus on the interactions, and we assume an OEM models of the elementary units. According to these models, we need to specify the functional and performance requirements, and the unit messages about both success and failure.

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 situation and activity reports. The service processes may employ behavioral twins, intended to provide static and exploratory preview information, based on simulation (Luqi, 1989).

The supervisor processes are use cases of the controller processes, and the latter are use cases of the service processes. Process duplication may support single use-case per process, enabling to prevent conflicting mode setting.

Model-based coordination

A method used to design the coordination between processes is based on the principle of multiple layer defense, as demonstrated using the Swiss Cheese illustration: the preferred layer is by risk elimination, rebounding from hazards, and finally resilience.

1. The primary protection layer comprises methods for preventing hazards, such as by constraining the operation and by notifying on approaching the protection boundaries.

2. Not all hazards are expected. A second protection layer is about threat detection. 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 process execution or state transition.
3. Not all expected hazards can possibly be prevented. A third protection layer comprises methods for rebounding from exceptional situations, such as by alerting the operators about the increase of the risk level.
4. Occasionally, the operators might fail to rebound from the exceptional situation. The fourth protection layer comprises methods for preventing the situation escalation, of the hazard transforming to threats. The methods are by applying troubleshooting and recovery procedures, by 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 the coordination practically fails. For these cases, the system should apply a last protection layer, which is by employing resilience procedures.

Rule definition

Key generic meta rules are intended for scenario definition and scenario-based situation and activity design. Scenarios are used as situation vectors, namely, pointers to the set of state machines, thus reducing the situational complexity from exponential to linear. The rules should define the mapping from scenarios to the 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 mapping from scenarios to vectors were:

X-ray ➔ (in, high), and

E-beam ➔ (out, low)

The accident was due to an exceptional situation (out, high).

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 activity risks.

Transdisciplinary engineering

Error proofing is a transdisciplinary activity:

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

In the context of HSI, the system should provide the human operators with clear and comprehensive information required for decision making. This information should include static prediction of the system situation, as well as exploratory prediction of operational options. The information should be provided gradually, according to the needs for decision making, employing HCD principles, considering the mental capabilities of the human operators.

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.

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

Harel, A 1999, Automatic Operation Logging and Usability Validation, *Proceedings of HCI International '99*, Munich, Germany, Vol. 1, pp. 1128-1133

Harel, A 2009, Statistical Analysis of the User Experience, Invited talk - *2nd Meeting of isENBIS*, Hertzelia, Israel

Harel, A 2011, Comments on IEC 60601-1-8. *Letter submitted to IEC/TC 62 working group*.

Harel, A 2021. Towards Model-based HSI Engineering: A Universal HSI Model for Utility Optimization, to be published in *Proceeding of the second HSI conference,* San Diego, US.

Harel, A, Kenett, R & Ruggeri, F 2008, - Modeling Web Usability Diagnostics on the basis of Usage Statistics. in: *Statistical Methods in eCommerce Research,* W. Jank and G. Shmueli editors, Wiley.

Harel, A & Weiss, M, 2011, Mitigating the Risks of Unexpected Events by Systems Engineering, The Sixth Conference of INCOSE-IL, Hertzelia, Israel

Harel, A & Zonnenshain, A 2019, Engineering the HSI. *Proceedings of the first HSI conference*, Biarritz, France

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.

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.

Zonnenshain, A & Harel, A 2015, A practical guide to assuring the system resilience to operational errors, *INCOSE. Annual International Symposium, Seattle.*