2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
17-20 October, 2021. Melbourne, Australia

2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC) | 978-1-6654-4207-7/21/$31.00 ©2021 IEEE | DOI: 10.1109/SMC52423.2021.9658721

A Taxonomy of Forcing Functions for Addressing Human Errors in Human-machine Interaction*

Pengyuan Wan[1] and Matthew L. Bolton[1]

***Abstract*— A forcing function is an intervention for constraining human behavior. However, the literature describing forcing functions provides little guidance for when and how to apply forcing functions or their associated trade-offs. In this paper, we address these shortcomings by introducing a novel taxonomy of forcing functions. This taxonomy extends the previous methods in four ways. First, it identifies two levels of forcing function solidity: hard forcing functions, which explicitly enforce constraints through the system, and soft forcing functions, which convey or communicate constraints. Second, each solidity level is decomposed into specific types. Third, the taxonomy hierarchically ranks forcing function solidities and types based on trade-offs of constraint and resilience. Fourth, for hard forcing functions, our taxonomy offers formal guidance for identifying the minimally constraining intervention that will prevent a specific error from occurring. We validated the ability of our method to identify effective error interventions by applying it to systems with known errors from the literature. We then compared the solutions offered by our method to known, effective interventions. We discuss our results and offer suggestions for further developments in future research.**

I. INTRODUCTION

One method for preventing human errors from causing problems is to use forcing functions. Forcing functions are constraints enforced on human inputs to a system that prevent errors from occurring [1]. For example, to ensure that a driver does not drive away from a fuel pump with the nozzle still in the car, some modern cars will not allow the engine to start unless the gas cap is in place.

While forcing functions are effective, they need to be used with caution. This is because, in limiting what behaviors humans can perform, system resiliency may be sacrificed [2]: the human ability to creatively respond to situations unanticipated by designers may be restricted. Additionally, forcing functions are loosely defined in the literature. This makes it difficult for engineers to identify what forcing function they should employ to address a given error and any associated effectiveness and resilience trade-offs.

Advances in human error taxonomies [3] have given engineers an unprecedented ability to formally understand how, why, and where (in a task model) erroneous behaviors occur [4]. In this work, we attempted to build off of this contribution by introducing a taxonomy of forcing functions that will allow analysts to use insights that can be gained from this modern taxonomy to select forcing function interventions with full knowledge of resilience trade-offs.

*This material is based upon work supported by the National Science Foundation under Grant Nos. 1918314.

[1]Pengyuan Wan and Matthew L. Bolton are both with the Department of Industrial and Systems Engineering at the University at Buffalo, the State University of New York, 342 Bell Hall, Buffalo, NY 14260, USA mbolton@buffalo.edu

II. BACKGROUND

Below we cover background on forcing functions and related barrier concepts. We also describe the task-based taxonomy of human error.

A. Forcing functions and Barriers

According to Norman [1], [5], forcing functions are physical constraints that can be implemented using: interlocks, lockins, and lockouts. An **interlock** forces operations to occur in a proper sequence. A **lockin** keeps an operation active, preventing someone from prematurely stopping it. A **lockout** prevents someone from entering a place that is dangerous or prevents a specific error from occurring. Beyond forcing functions, Norman also describes additional categories of constraints on human behavior: cultural, semantic, and logical. Cultural constraints use social situations and norms to limit behavior. Semantic constraints use human situational knowledge to regulate behavior (e.g., making computer file constructs and operations similar to those of real files). Logical constraints use logical relationships between behavior and the system to limit human error (e.g., having a part left over during equipment assembly indicates an error).

Barriers [6] also provide guidance for addressing human errors. Barriers are functions that protect against the uncontrolled transportation of mass, energy, or information. When applied to human-machine interaction, a barrier is used to prevent user operations. There are generally four types of barriers: physical or material barriers, functional barriers, symbolic barriers, and incorporeal barriers. Physical barriers physically prevent an error from occurring. Functional barriers create a precondition for an action, where the precondition must be met before the user can perform the action. Symbolic barriers provide information to users so that they know what actions they should or should not perform. Finally, incorporeal barriers rely on human knowledge for direct enforcement (e.g., training, rules, regulations, and laws).

There are tradeoffs between barrier types [6]. In general, physical and functional barriers are more restrictive than symbolic or incorporeal ones, which can often make them more effective, robust, and reliable. However, this often comes with the disadvantage of implementation being more expensive and time consuming.

Multiple barriers and redundancy can improve effectiveness [7], [6]. Physical or functional barriers can back up symbolic or incorporeal barrier systems. Alternatively, symbolic or

incorporeal forcing functions can be used to discourage users from employing workarounds for functional or physical barriers [6].

There were three main limitations of these approaches. First, none provide clear guidance for how their concepts can be used to fix specific human errors. Forcing functions [1] do not describe how each forcing function constraint should be applied to most effectively address specific errors. The barrier literature does provide guidance for how to implement the different constraint concepts. However, these recommendations are very general and not specific to human error. Second, neither method provides a sense of trade-off in terms of solutions' impact on resilience: the ability of a system to maintain or regain stable operation after or during a failure [8]. This is important because the ability of humans to deploy creative solutions to problems during unexpected or unusual circumstances is often a source of system resilience. Thus, strict constraints on human behavior can negatively affect resilience. Third, forcing functions and barriers contain similar concepts, but both contain information that is not in the other. This can make it difficult for analysts to know what concept is appropriate in a given situation.

B. The Task-Based Taxonomy of Human Error

In practice, engineers will likely want to understand both the phenotype [9] and genotype [10] of an error. They will also want specific information about the human's goals and what they were (or were not) attending to when the error occurred. This synthesis was the basis for the task-based taxonomy of human human error [3]: a formal system that unifies the genomenological and phenomenological by classifying errors based on where in a hierarchical task model the divergent behavior occurred. Specifically, by knowing exactly where in a task a human's behavior diverged, an analyst will know how the erroneous behavior manifested (its phenotype), what information (task or strategic knowledge contained in the structure) the human had to incorrectly attend to to perform the error (its genotype), and the context of the error (what goals the human was trying to achieve) based on the location of the error in the task hierarchy.

The specifics of the task-based taxonomy are formulated using the Enhanced Operator Function Model (EOFM) [11], [12]. This is because EOFM uses standard hierarchical task model concepts while also having a formal semantics (a mathematical description of the behavior encompassed by the task model). The formal semantics are useful in error classification because any divergence from the normative task will inherently violate the semantics. The nature of the violation specifically indicates what information was improperly attended to (which can include information about task execution itself, in the person's memory, or perceived from the environment). The semantics are also useful because knowing where an error occurs enables an analyst to to determine what set of behaviors will follow the violation. For example, if a person skips an activity that is in a sequence, then the following activity in that sequence will occur immediately after the error.

While there are multiple levels of error classification in the task-based taxonomy, this research will only focus on the error mode level. Specifically an activity can execute when it should not (an intrusion), transition to done when it should not (an omission), restart when it should not (a restart), and not transition when it should (a delay). The taxonomy further decomposes all of these concepts to understand why and how these errors occur based on which semantic information (strategic or task knowledge) was violated.

III. Objective

Given the disparate concepts and lack of guidance for how to apply barrier and forcing functions to address human errors, engineers could find it difficult to determine how to best improve system safety without compromising resilience. In this research, we sought to address these issue. To accomplish this, we created a unified taxonomy of forcing functions that combines the two concepts [1], [6]. Additionally, we addressed the major limitation of both by using the formalism offered by the task-based taxonomy of human error to reason about the behaviors associated with a given, unwanted error. We then use this to determine how to minimally constrain human behavior to prevent the error and thus minimally impact resilience. In what follows, we describe our taxonomy and show how interventions can be identified using it.

IV. A Taxonomy of Forcing Function

Figure 1 shows our forcing function taxonomy. The taxonomy is hierarchical in that it classifies interventions based on how restrictive they are (and thus how many potential errors/actions they prevent). This has an inversely proportional relationship with resilience, here grossly indicated by the number of behaviors/actions allowed. The taxonomy first breaks forcing functions down based on their "solidity," where forcing functions can be "hard" or "soft".

Hard forcing functions prevent an error by having the human-machine interface specifically prevent the associated erroneous actions from occurring. They are thus compatible with Norman's [1] original concept of the the forcing function as well as Hollnagel's physical and function barriers [6].

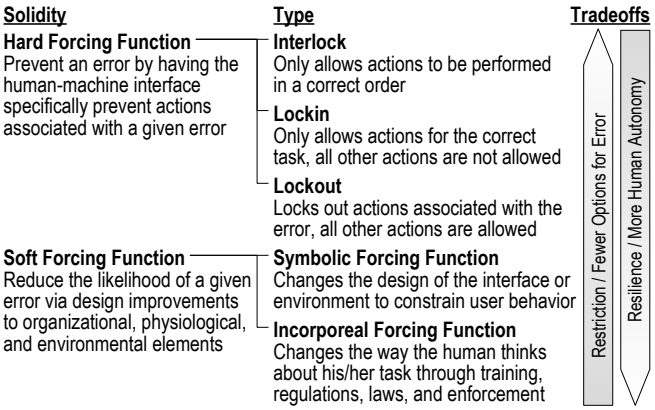

Fig. 1. The taxonomy of forcing functions.

Forcing functions with soft solidity are those that do not explicitly restrict human actions. Rather, they attempt to prevent errors by removing the organizational, psychological, and environmental conditions that cause the error. This makes them compatible with symbolic and incorporeal barrier systems [6] as well as Norman's extended discussion on cultural, semantic, and logical constraints.

The solidity levels are further broken down into specific forcing function types, each of which also falls along the restrictive/resilience continuum (see Fig. 1).

Compatible with Norman's categories of forcing functions, hard forcing functions have three levels: interlock, lockin, and lockout. An interlock is a forcing function where the human-machine interface only allows actions to be performed in a specific or set of specific correct orders (presumably as dictated by the operator's normative task). It is the most restrictive of the hard forcing functions. A lockin is a forcing function where the human-machine interface only allows actions associated with the correct task that is being performed. It is thus less restrictive than an interlock because it does not enforce an action order. Finally, a lockout is a forcing function that locks out or prevents the performance of all actions associated with an erroneous behavior. Given that it only restricts behavior associated with a given error, a lockout is the least restrictive of the hard forcing functions.

Soft forcing functions are broken down into two types: symbolic and incorporeal. Symbolic forcing functions prevent an error by changing the design of the interface or environment so that the human will be able to more readily identify correct behaviors. These are consistent with symbolic barriers [6] or logical constraints [1]. There can be many different types of symbolic forcing functions which could include things like improved information displays, warning signs, and decision support systems. Incorporeal forcing functions prevent an error by changing the way the human thinks about the task. These are consistent with incorporeal barriers [6] and semantic and cultural constraints [1]. These forcing functions are meant to encompass things like training, regulations, laws, cultural norms, and enforcement. Generally, incorporeal functions are less restrictive than symbolic ones because they strictly rely on changing information in a human's mind rather than something in the physical world.

Hard forcing function types are mutually exclusive solutions: more restrictive forcing functions inherently encompass the behavior of less restrictive ones. However, no such restriction exists for soft forcing functions. In fact, multiple symbolic functions, incorporeal functions, or combinations of them could be used to address a given error. Additionally, although potentially redundant, these can be used alongside any selected hard forcing function. For example, you could provide a user with (symbolic) information that makes it clear why a hard forcing function is being enforced.

V. Hard Forcing Function Selection

Based on the hierarchical arrangement of forcing function concepts from above (see Fig. 1), we know what the tradeoffs are between interlocks, lockins, and lockouts. However, this representation does not give us specific guidance about which actions need to be locked out, locked in, or interlocked. In what follows, we describe how we can use an EOFM's formal interpretation of task behavior and its task-based human erroneous behavior taxonomy to reason about what and how errors occur. We then use this information to identify how to minimally constrain a human's task with lockout, lockin, and interlock interventions to prevent specific errors from occurring. Below we breakdown how hard forcing function interventions can be identified for each of the three error modes from the task-based taxonomy.

A. Intrusion

In the following discussion, let A be the set of actions associated with performing an activity normatively and let B be the set of actions associated with an erroneous, intruding activity. If the intruding actions are a subset of the normative actions, or they are equal, ($B \subseteq A$), then **interlock** is the only option. This is because there are no actions in B that distinguish it from A, thus execution order and context is the only way to distinguish the execution of B from A. This relationship is illustrated in Fig. 2(a).

If the actions of A are a subset of B's ($A \subset B$; Fig. 2(b)) or neither are subsets of each other but have overlap ($A \cap B \neq \emptyset \wedge A \not\subseteq B \wedge B \not\subseteq A$; Fig. 2(c)), then there are actions in B that are different from A. In this situation, all three hard forcing functions are options. Thus, only a lockout of B is strictly required to prevent the error. In this case, analysts will need to select the solution that best suits their desired tradeoffs between resilience, options for error, and ease of

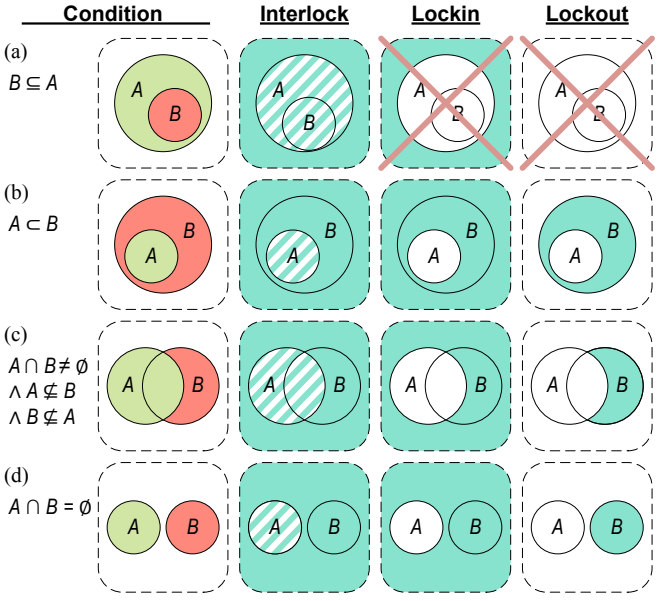

Fig. 2. The left column shows the possible set relationships between normative activity actions A (green) and those associated with the erroneous activity B (red). The three additional columns show how actions are constrained for each of the hard forcing function options for the associated set relationships. In these, aqua areas show which actions are constrained by the associated forcing function. Aqua areas with diagonal lines indicate interlock constraints. Red Xs show when a function is not effective.

implementation. Given the overlap between A and B, it may be conceptually simpler to lockin to A instead of selectively locking out the actions of B that are not in A or interlocking.

If A and B have no shared actions ($A \cap B = \emptyset$; Fig. 2(d)), all three hard forcing functions are also an option and thus a lockout is sufficient to completely prevent the error. Because there is no overlap between A and B, a lockout should be no harder to implement than an interlock or lockin. Thus, a lockout should generally be preferred because it is less restrictive / more resilient than the other options.

B. Omission

The forcing functions for addressing omission error modes use the formulation for intrusions. Specifically, the omission of any activity or action will ultimately be followed by the performance of some activity or action that would have come after the omitted behaviors (which should be easily determined from accident reports, examination of task models, or formal analysis of said task models [11], [12]). Thus, omission errors can be treated as intrusion errors, where A is the activity that you are attempting to prevent from being omitted and B is the activity (or activities) that could occur following A's omission. With this formulation, the guidance provided for intrusions will address omission errors.

Note that the only exception to the above procedure occurs when a human omits the last set of actions in a task without there being any additional behaviors following it. Such a situation could arise due to a postcompletion error (when primary goals are satisfied before actions for performing subsidiary goals can occur [13]). In this situation, the task would need to be reordered to ensure that the task only completes with the satisfaction of its primary goal (see [14] for a formal description of this intervention approach). Alternatively, soft forcing functions (such as alarms, additional information displays, or training) may be required to ensure humans are properly attending to task completion.

C. Restart

The forcing functions for restarts are handled almost identically to omission errors. For errors that occur with the restart error mode, A is the activity at which an erroneous restart was initiated and B is the activity that initiates the restart. Using this formulation, an analyst would then follow the guidance for intrusions.

VI. Applying the Taxonomy

When using the taxonomy to address human errors with hard forcing functions, we assume that the analyst has identified the normative activity within a task that was disrupted. He or she should also also identify the intruding erroneous activity. As the discussion above makes clear, these designations manifest for intrusion, omission, and restart error modes. Then, based on the parameters outlined under intrusions, the analyst should be able to determine if an interlock, lockin, or lockout is a viable solution, and the minimal restrictions for implementing them. At this point, it is at the analysts discretion which intervention to use.

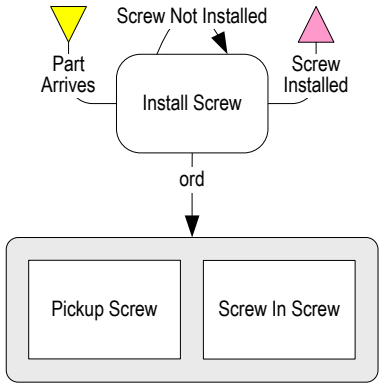

Fig. 3. Screw install task model rendered in the visual notation of EOFM ([11], [12]). Activities are rounded rectangles, actions are pointed ones. Preconditions (yellow triangles), completions conditions (magenta triangles), and repeated conditions (recursive arrows) are attached to activities and labeled with condition information. Arrows below activities point to decompositions that define sub-activities or actions. Operators on decomposition arrows indicate execution of subtasks in order from left to right (ord) or that one or more can execute in any order (or_par).

While it is true that there could be mitigating, contextual, or implementation circumstances that might suggest analysts choose a more restrictive solution (such as an interlock), the taxonomy makes it clear that doing so will come at a potential loss of resilience. Thus, it is our general recommendation that analysts choose the effective intervention that puts the least restriction on human actions. Given the non-mutual-exclusivity of soft forcing functions, any number of these could be applied with or without hard forcing function interventions. In the next section we provide examples that demonstrate how our taxonomy could be used to identify effective interventions.

VII. Illustrative Examples and Validation

In this section, we present two illustrative examples to show how our forcing function taxonomy can be effectively used when attempting to eliminate a human error in a system. These examples were specifically drawn from the literature on poka-yoke [15] (a Japanese design philosophy which means "mistake-proofing"). This allowed us to provide some validation of our approach by comparing the solutions produced by our method to ones that have been successful in the field.

A. Example 1: A Screw Installation Task

Consider a system where a sequence of devices are delivered to the human on a conveyor belt. It is the human's job to perform an assembly task where a screw is installed that holds each device together.

Figure 3 shows the associated task model (rendered with EOFM [11], [12]). When the device arrives to the human on a conveyor, he or she will install a screw by first picking up a screw out of a box and then screwing it into the device.

In this system, workers will sometimes fail to attach a screw to the device by missing a device on the conveyor. This is an omission, where the entire normative task (Fig. 3; A) is the omitted activity. Thus, the intruding activity is the

performance of that same task on the next device on the conveyor (Fig. 3; B). This constitutes a situation where $A = B$ and thus, according to our approach (Fig. 2(a)), only an interlock will be effective. This implies that steps must be taken to ensure that no device can be allowed to pass by on the conveyor without the screw being installed.

If we look at the poka-yoke solution to this problem [15], mechanisms were installed to ensure that the convey system can detect whether the worker has picked up a screw for a device as it is being conveyed past. If the worker fails to pick up a screw, then the conveyor is stopped. Thus, the neglected device is the one immediately available to the worker, ensuring that it is not skipped.

Thus, our approach dictated an interlock solution to address the omission error and this is consistent with the presented poka-yoke solution, which specifically forces an order of actions on human workers to avoid an omission.

B. Example 2: A Part Collection Task

For the the second example, consider a situation where a worker first reads model identifiers off of a specification chart and then goes to a shelf of boxes and selects the parts with the given identifiers out of the appropriate boxes (one box per identifier). The worker would then install the parts into a larger system. For this procedure, it was observed that workers occasionally installed the wrong parts. A deeper analysis revealed that this error occurred during when selecting parts from the shelf (not during chart consultation).

The model for performing this task is shown in Fig. 4. The error of a worker selecting the wrong part would constitute an intrusion. In this situation, the correct activity A is the activity for picking up a part from a correct box. The intruding activity B is the activity for picking up a part from any incorrect box. In this situation, there should be no overlap between the correct and incorrect activities (the condition described in Fig. 2(d)), thus a lockout will provide the minimally constraining solution. For the example, this means that the system must prevent the worker from picking up parts from boxes other than the ones indicated on the chart.

In the poka-yoke solution to the problem [15], all of the part boxes are given automated lids which unlock and open when a new specification chart is loaded that requires the given part. The lids would automatically close and lock when a different, incompatible chart was loaded. Thus, the poke-yoke solution to the incorrect part selection problem is compatible with the lockout solution identified by our approach.

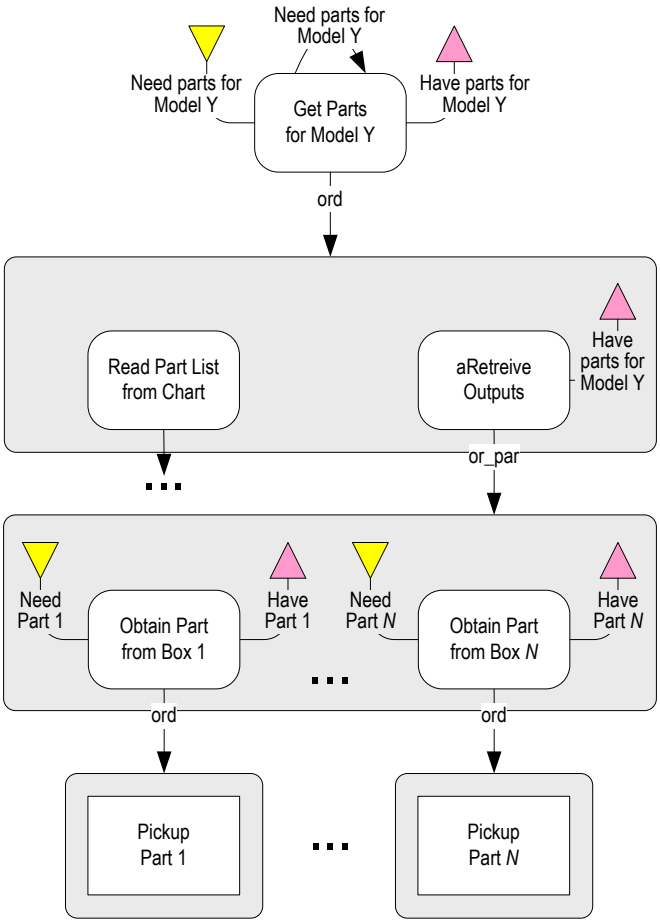

Fig. 4. The original task for selecting parts for a given system model. The worker first reads the part specification from a chart and them picks up parts from the appropriate boxes in the shelf.

VIII. Discussion and Conclusion

We have made a number of significant contributions in this research. (1) We presented a new forcing function hierarchy that describes the tradeoffs between the restriction and resilience offered by different forcing function concepts. This explicitly extended concepts from the larger forcing function and barrier literature. (2) Our taxonomy classifies forcing functions into two broad categories: hard forcing functions (those that prevent certain actions from occurring) and soft forcing functions (those that discourage actions without hard constraints). (3) The taxonomy identified tradeoffs between resilience and constraint effectiveness between hard and soft forcing functions, as well as sub-categories of forcing functions within each designation. (4) For hard forcing functions, we presented a novel formal approach based on set theory (enabled by formal interpretations of task models and the task-based error taxonomy) that provides guidance for choosing the most appropriate hard forcing function. (5) Through examples, we showed how method application could be used to identify specific interventions based on the activities (and associated actions) that need to be constrained. These applications also provided evidence of face validity by showing that our method recommends interventions that are consistent with known solutions.

Below we further discuss our results based on the method limitations, other potential applications, and future research.

A. Limitations

One limitation of our method is the generality of produced hard forcing functions solutions. The taxonomy and formal method do not recommend specific design implementations of the recommended forcing function behavior. This means that designers will still need to develop a detailed approach

that prevents the identified erroneous actions and/or activities from executing. Future research should investigate methods for making recommendations more application specific.

Another limitation arises because of the assumptions of the hard forcing function identification process. First, the analysts must have an accurate system hierarchical task model. Second, the analysts must know the exact erroneous behaviors they are eliminating. Practically, this may not be common. Thus, future work should investigate how standard task analytic methods can be better integrated into industrial practice.

Finally, the current formulation of our taxonomy and method does not provide recommendations for deploying soft forcing functions. Future work should investigate how information contained in hierarchical task models can be used to recommend soft function interventions as additions or alternatives to hard forcing function options. In particular, hard forcing function implementation must be done carefully. This is because the ability of users to understand what actions are being constrained (and why that constraint is occurring) is critical to system safety. Specifically, this helps avoid mode confusion, automation surprise, and associated additional human errors [16]. Soft forcing functions have the potential to offer alternatives to hard functions without such concerns, or could be used with hard forcing functions to help humans understand their operational context. Such uses of soft forcing functions should be the subject of future research.

B. Future Works

1) Automatic Interface Repair: This research is part of a larger project to improve the reliability of human-machine interaction through automatic, formal methods for assessing reliability and repairing human-machine interfaces. The formal method for identifying forcing function interventions should serve as the basis for identifying automatic repairs for errors that reliability analysis [17], [18] will identify. Future work should investigate how forcing functions could be automatically applied to a human-machine interface design in automated repair, similar to the way that interlock designs could be generated by transforming task models [19].

2) Human Behavior Tracking and Error Detection: There are different ways to implement all of the hard forcing function concepts from our hierarchy. In a situation with available computational resources, all hard forcing functions could be generically implemented through the use of task behavior tracking. In such a situation, the machine/interface would be able to follow what task (and activities and actions within the task) the person is performing based on environmental or system conditions and what human actions have been performed. Such a system could allow or disallow human actions based on what task is being performed and the hard forcing function concept designers wish to enforce. EOFM, as a task behavior formalism, should be capable of being used in such a system. The OFM (operator function model; EOFM's predecessor), was used for tracking human behavior in intelligent tutoring systems [20]. Thus, such a system should be possible with EOFM and should be investigated in future efforts.

It is conceivable that soft forcing functions could also benefit from task tracking. For example, tracking could be used to enhance or change display concepts based on operational context; make the endogenous task information (task information that should presumably exist in the person's mind) exogenous (explicitly represented in the interface) in specific situations; provide humans with feedback when they appear to go off task either during actual system operations or during training; or help suggest error recovery actions.

REFERENCES

[1] D. Norman, *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[2] R. Amalberti, "Optimum system safety and optimum system resilience: Agonistic or antagonistic concepts?" in *Relience Engineering*, E. Hollnagel, D. D. Woods, and N. Leveson, Eds. Farnham: Ashgate Publishing, Ltd., 2006, ch. 16, pp. 253–271.

[3] M. L. Bolton, "A task-based taxonomy of erroneous human behavior," *International Journal of Human-Computer Studies*, vol. 108, pp. 105–121, 2017.

[4] M. L. Bolton, K. A. Molinaro, and A. M. Houser, "A formal method for assessing the impact of task-based erroneous human behavior on system safety," *Reliability Engineering & System Safety*, vol. 188, pp. 168–180, 2019.

[5] C. Lewis and D. A. Norman, "Designing for error," in *Readings in Human–Computer Interaction*. Elsevier, 1995, pp. 686–697.

[6] E. Hollnagel, "Risk+ barriers= safety?" *Safety science*, vol. 46, no. 2, pp. 221–229, 2008.

[7] F. Vanderhaegen, "Human-error-based design of barriers and analysis of their uses," *Cognition, Technology & Work*, vol. 12, no. 2, pp. 133–142, 2010.

[8] E. Hollnagel, "Resilience – the challenge of the unstable," in *Resilience engineering: Concepts and precepts*, E. Hollnagel, D. D. Woods, and N. Leveson, Eds. Farnham: Ashgate Publishing, Ltd., 2006, ch. 1, pp. 9–19.

[9] ——, "The phenotype of erroneous actions," *International Journal of Man-Machine Studies*, vol. 39, no. 1, pp. 1–32, 1993.

[10] J. Reason, *Human error*. Cambridge university press, 1990.

[11] M. L. Bolton, R. I. Siminiceanu, and E. J. Bass, "A systematic approach to model checking human-automation interaction using task-analytic models," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 41, no. 5, pp. 961–976, 2011.

[12] M. L. Bolton and E. J. Bass, "Enhanced operator function model (EOFM): A task analytic modeling formalism for including human behavior in the verification of complex systems," in *The Handbook of Formal Methods in Human-Computer Interaction*, B. Weyers, J. Bowen, A. Dix, and P. Palanque, Eds. Cham: Springer, 2017, pp. 343–377.

[13] M. D. Byrne and S. Bovair, "A working memory model of a common procedural error," *Cognitive Science*, vol. 21, no. 1, pp. 31–61, 1997.

[14] P. Curzon and A. Blandford, "Using a verification system to reason about post-completion errors," in *Design, Specification and Verification of Interactive Systems*, 2000.

[15] S. Shingo, *Zero quality control: source inspection and the poka-yoke system*. CRC Press, 1986.

[16] A. Degani and M. Heymann, "Formal verification of human-automation interaction," *Human Factors*, vol. 44, no. 1, pp. 28–43, 2002.

[17] X. Zheng, M. L. Bolton, and C. Daly, "Extended SAFPHR (systems analysis for formal pharmaceutical human reliability): Two approaches based on extended cream and a comparative analysis," *Safety Science*, vol. 132, 2020.

[18] M. L. Bolton, X. Zheng, and E. Kang, "A formal method for including the probabilityof erroneous human task behavior in system analyses," *Reliability Engineering and System Safety*, 2021, in Press.

[19] M. Li, J. Wei, X. Zheng, and M. L. Bolton, "A formal machine–learning approach to generating human–machine interfaces from task models," *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 6, pp. 822–833, 2017.

[20] R. W. Chu, C. M. Mitchell, and P. M. Jones, "Using the operator function model and ofmspert as the basis for an intelligent tutoring system: Towards a tutor/aid paradigm for operators of supervisory control systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 7, pp. 1054–1075, 1995.