THE INTERNATIONAL JOURNAL OF AVIATION PSYCHOLOGY, 22(4), 343–366

ISSN: 1050-8414 print / 1532-7108 online
DOI: 10.1080/10508414.2012.718240

Routledge
Taylor & Francis Group

Using Model Checking to Explore Checklist-Guided Pilot Behavior

Matthew L. Bolton[1] and Ellen J. Bass[2]
[1]*Department of Mechanical and Industrial Engineering, University of Illinois, Chicago, Illinois*
[2]*Department of Systems and Information Engineering, University of Virginia, Charlottesville, Virginia*

Pilot noncompliance with checklists has been associated with aviation accidents. This noncompliance can be influenced by complex interactions among the checklist, pilot behavior, aircraft automation, device interfaces, and policy, all within the dynamic flight environment. We present a method that uses model checking to evaluate checklist-guided pilot behavior while considering these interactions. We illustrate our approach with a case study of a pilot performing the "Before Landing" checklist. We use our method to explore how different design interventions could impact the safe arming and deployment of spoilers. Results and future research are discussed.

To support flight operations, Federal Aviation Regulations require that airlines supply checklists (step-by-step instructions for carrying out or verifying information). However, noncompliance with checklists continues to be associated with aviation accidents (Degani & Wiener, 1991; Graeber & Moodi, 1998; Lautman & Gallimore, 1987; National Transportation Safety Board [NTSB], 1994). Such noncompliance can be influenced by a number of factors. With respect to checklist design, for example, ordering (Burian, 2004; Degani & Wiener, 1993), wording (Burian, 2004; Degani & Wiener, 1993), and level of detail (Burian, 2004) can impact compliance. Limitations on pilot performance such as working memory and attention can also affect compliance (Burian, Barshi, & Dismukes, 2005; de Brito, 2002). The design of aircraft systems and displays can also play a role. For example, avionics might not provide enough information for pilots to keep

Correspondence should be sent to Matthew L. Bolton, Department of Mechanical and Industrial Engineering, University of Illinois, 842 W. Taylor, Chicago, IL 60607. E-mail: mbolton@uic.edu

track of the underlying automation's modes (Sarter & Woods, 1995). Alerting systems could produce multiple concurrent alarms and might not clearly indicate what procedure is appropriate (Bass, Ernst-Fortin, Small, & Hogans, 2004; Burian et al., 2005). In addition, the dynamics of the aircraft systems might not support the timing of checklist items during procedure execution (Degani, 2004).

Many methods have been employed to evaluate checklists, including questionnaires and structured interviews (de Brito, 2002; Degani & Wiener, 1991), part-task simulation studies (de Brito, 2002; Landry & Jacko, 2006), in-flight observations (Degani & Wiener, 1993), and reviews of accident reports (Degani & Wiener, 1991, 1993). To assist in the creation of checklists, design guidelines have been compiled (Burian, 2004; Degani & Wiener, 1997) and generation algorithms have been developed (Degani, Heymann, & Shafto, 1999). All of these efforts have provided valuable insights into how to improve the use of flight deck checklists. By considering all of the possible modeled interactions, this article suggests that formal methods, and specifically formal verification techniques, offer additional opportunities to support checklist development and evaluation.

Formal methods are a set of well-defined mathematical languages and techniques for the modeling, specification, and verification of systems (Wing, 1990). Systems are modeled using mathematically based languages, specifications are formulated to describe desirable system properties, and a verification process mathematically proves whether or not the model satisfies the specification. Model checking is a highly automated approach used to verify that a formal model of a system satisfies a set of desired properties (a specification; Clarke, Grumberg, & Peled, 1999). A formal model describes a system as a set of variables and transitions between variable states. Specification properties are usually represented in a temporal logic (see Emerson, 1990) using the formal system model variables to construct propositions. Verification is performed automatically by exhaustively searching a system's state space to determine if these propositions hold. If there is a violation, an execution trace called a counterexample is produced. This counterexample depicts a model state (the value of the model's variables) corresponding to a specification violation along with a list of the incremental model states leading up to the violation.

To use formal verification in the development and evaluation of checklists in a comprehensive analysis, one must consider the contribution of not only pilot behavior, but also aircraft system design, displays and controls, and the operational environment. We have developed tools and methods for modeling task behavior, human mission goals, human–device interfaces, device automation, and environmental conditions together in a formal framework (Bolton & Bass, 2009, 2010a; Bolton, Siminiceanu, & Bass, 2011). In this work, we show that this framework can be used to evaluate checklist-guided pilot behavior while also considering interactions with the aircraft, the design of its systems, and the operational environment.

INSTRUMENT APPROACH

An instrument approach procedure involves navigating the aircraft to the runway with the aid of two independent subsystems, one providing lateral guidance (localizer) and the other vertical guidance (glide slope). The vertical position of the aircraft relative to the glideslope is displayed with a moving diamond on the glideslope indicator (Figure 1). When the aircraft is within range of the instrument landing system, and nearing the glideslope, the diamond will become "alive," moving toward the center of the display. The diamond will first pass through the "two dot" and then the "one dot" positions. When the aircraft is on the glideslope, the diamond is at the capture position.

To land safely, the pilot performs the Before Landing checklist. Herein this means the following (Figure 2): (a) the ignition must be set to override, (b) the landing gear must be down, (c) the spoilers should be armed, (d) the flaps

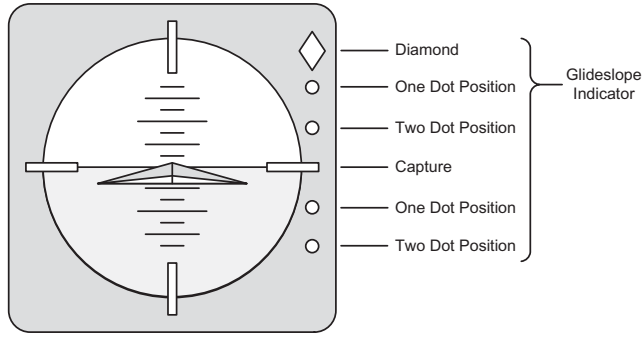

FIGURE 1 A simplified representation of an aircraft artificial horizon display with a glideslope indicator on the right.

BEFORE LANDING

Ignition……..……..Override

Landing Gear……..Down,Three Green

Spoilers……Armed

Flaps…….Extended, 40 Degrees

Annunciator Panel……..Checked

FIGURE 2 The Before Landing checklist.

should be extended to the appropriate flap setting (40° in this case), and (e) the annunciator panel should be checked.

The ignition should be set to override so that there will be enough ignition power to restart the engine should it quit. The ignition is controlled by a switch and a light illuminates when the switch is set to override.

Once the glideslope indicator diamond is alive, the pilot can deploy the landing gear. Pulling the landing gear lever opens the landing gear doors and deploys the gear. In a well-functioning aircraft, the doors can take approximately 10 sec to completely open. As such, the landing gear will fully deploy before the landing gear doors are completely open. Three lights on the flight deck (one for each of the landing gear) illuminate when the landing gear is fully deployed. Another light illuminates when the landing gear doors begin to open, and remains on until they are completely open.

The pilot progressively extends the flaps to reduce the aircraft's stalling speed and allow for safe flying at slower speeds. Extending flaps also increases drag, which helps to slow the aircraft. In general, when the aircraft is between the one dot and capture positions, the pilot has slowed to a speed where the flaps should be set to 25°. When the aircraft has reached the capture position, the pilot should set the flaps to 40°. Pilots can also progressively set the flaps to intermediate degrees before or in between these two settings. The position of the flaps is indicated by a gauge on the flight deck.

The annunciator panel is checked to ensure that the rudder is unrestricted so that it can be used to help control aircraft yaw during landing.

Spoilers are retractable plates on the wings that, when deployed, slow the aircraft and decrease lift. A pilot can arm the spoilers for automatic deployment using a lever. Alternatively, a pilot can manually deploy the spoilers after touchdown. If spoilers are not used, the aircraft can overrun the runway (see, e.g., American Airlines Flight 1420; NTSB, 2001). If spoilers are deployed too early, the aircraft loses lift and could have a hard landing. A pilot who forgets to arm the spoilers might attempt to deploy them manually, but do so prematurely (as with Air Canada Flight 621; Flight Safety Foundation, 1974a). Premature deployment can also occur due to mechanical issues. Arming the spoilers before the landing gear has been lowered or the landing gear doors have fully opened can result in automatic premature deployment (Degani, 2004). Further, if the landing gear configuration interferes with spoiler arming, and pilots move on to subsequent checklist items, they might forget to return to the arming step (Degani, 2004).

OBJECTIVES

Problems related to spoiler deployment could involve the Before Landing checklist, pilot behavior, the flight deck avionics interfaces, and the automation,

all interacting in a dynamically changing environment. Variations in pilot behavior (what actions are performed and when) and automation behavior (the timing of automation controlled procedures) have been associated with spoiler-related failures. Herein, we use our formal modeling framework and formal verification with model checking to explore these issues. We first discuss our methods. This includes a description of our framework, the associated models, and the specifications to be verified. We then describe two phases of analysis. In the first, we manipulate human task behavior and mission goals to explore how they affect safe spoiler deployment. We then explore how modified models of the device automation, human–device interface, human mission, and human task behavior could eliminate the discovered problems. In the second analysis phase, the device automation is manipulated to investigate how landing gear deployment timing can impact the safe deployment of spoilers. As with the first set of analyses, we perform additional evaluations to explore potential solutions to discovered problems. We conclude with a discussion of our results and areas of future research.

METHODS

A Method for Model Checking Checklist-Guided Pilot Behavior

To analyze the performance of checklist-guided pilot behavior formally, one could write model checking code for each checklist to be tested. The code could then be integrated with formal models of the rest of the system (the mission, human–device interface, device automation, and environment; Figure 3; see Bolton & Bass, 2010a). Coupled with the safety specification (the system qualities to be

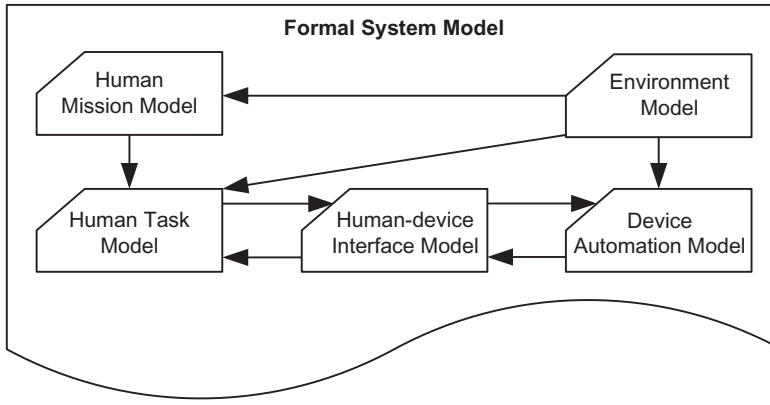

FIGURE 3 A formal modeling framework. An arrow from a submodel represents output variables and to a model, inputs variables.

verified), the analyst could then determine if the modeled system violated the specification using model checking. To ease the burden of representing checklists in model checking code, we have developed a task analytic modeling language called Enhanced Operator Function Model (EOFM; Bolton et al., 2011). When checklists are represented using EOFM, our tools automatically translate represented checklists into model checking code (Bolton et al., 2011). In this way, an analyst can define checklists with different steps, ordering of steps, and so on, and can more easily formally verify them.

EOFM is an Extensible Markup Language (XML)-based human task modeling language that can be used to define procedures (Bolton et al., 2011). EOFMs are generally more expressive than procedures as they are hierarchical and heterarchical representations of goal-driven activities that decompose into lower level activities, and finally, atomic actions (typically observable human actions but cognitive and perceptual actions are also possible). In addition, EOFMs express task knowledge not always listed in checklists. These are expressed as conditions indicating when activities can be undertaken: what must be true before they can execute (preconditions), when they can repeat (repeat conditions), and when they have completed (completion conditions). Every activity can decompose into one or more other activities or one or more actions. Many checklists do not specify the relationship between steps in a procedure, but a decomposition operator in EOFM can specify the temporal relationships between and the cardinality of the decomposed activities or actions (when they can execute relative to each other and how many can execute).

EOFMs can be represented visually as a tree-like graph (Bolton & Bass, 2010c). Actions are rectangles and activities are rounded rectangles. An activity's decomposition is presented as an arrow, labeled with the decomposition operator, that points to a large rounded rectangle containing the decomposed activities or actions. Herein, two of the nine decomposition operators (Bolton et al., 2011) are used:

- *ord*—All activities or actions in the decomposition must execute in the order in which they appear.
- *and_par*—All of the activities or actions in the decomposition must execute, where the execution of activities or actions can overlap.

Conditions on activities are represented as shapes or arrows (annotated with the logic) connected to the activity that they constrain. The form, position, and color of the shape are determined by the type of condition. A precondition is a yellow, downward-pointing triangle; a completion condition is a magenta, upward-pointing triangle; and a repeat condition (not used herein) is an arrow recursively pointing to the top of the activity.

EOFM has formal semantics that specify how an instantiated EOFM model executes (Bolton et al., 2011). Specifically, each activity or action can have one of three execution states: waiting to execute (*Ready*), executing (*Executing*), and done (*Done*). An activity or action transitions between each of these states based on its current state; the state of its immediate parent, its siblings (activities or actions contained in the same decomposition), and its immediate children in the hierarchy; and the decomposition operators that connect the activity to its parent and its children. Instantiated EOFM task models can be automatically translated (Bolton et al., 2011) into the language of the Symbolic Analysis Laboratory (SAL; de Moura, Owre, & Shankar, 2003) using the language's formal semantics. This allows the task models to be integrated into a larger formal system model using a defined architecture and coordination protocol (Bolton & Bass, 2010a; Bolton et al., 2011). Formal verifications are performed on this complete system model using SAL's Symbolic Model Checker (SAL–SMC).

Formal Modeling Approach

For the case study presented here, we consulted the Before Landing checklist (Figure 2), accident reports (Flight Safety Foundation, 1974a, 1974b; NTSB, 2001), and a related account (Degani, 2004).

A base formal system model including the operational environment, device automation, human–device interface, and human (pilot) mission must be defined for integration with the translated checklist (represented as a human task behavior model). This base model is implemented in the language of SAL.

For our aircraft model, the SAL input file was configured with three modules to represent the entire system (Figure 4): one representing the human task behavior model (*HumanTask*), one representing the human operator's mission goals (*Mission*), and one (*AIE*) representing the other elements of the system model (the device automation model, the human–device interface model, and the environment model). The three modules are ultimately composed together (using SAL's asynchronous composition operator '[]'; see de Moura et al., 2003) into the final system model (*System*).

Each module is an input–output model defined by variables (input, output, or local), and transition logic that determines how changes occur in local and output variables. Our modules can generally be viewed as having three distinct parts: variable declarations, where variables are defined in terms of their name, data type, and whether they are input, output, or local; initialization, where output and local variables are assigned their initial value; and transition logic (discussed later).

There are very distinct interactions between the modules. They communicate information to each other via the input–output relationships of their shared variables. Mission goals are communicated from the Mission module to the

```
aircraft : CONTEXT =
BEGIN
    % Constant and Type definitions
    ...

    % Module containing the mission model
    Mission: Module =
    Begin
         OUTPUT PreferToArmSpoilers: BOOLEAN
         INITIALIZATION
              PreferToArmSpoilers IN {TRUE, FALSE};
    END;

    % Module containing the task behavior model
    HumanTask: MODULE =
    BEGIN
         INPUT PreferToArmSpoilers: BOOLEAN
         INPUT Ready: BOOLEAN
         OUTPUT Submitted: BOOLEAN
         % Human actions as Output variables
         ...
         % Human-device interface information as INPUT variables
         ...
         % Task behavior execution state as LOCAL variables
         INITIALIZATION
              ...
              Submitted = FALSE;
              ...
         TRANSITION
         [
              ...
         ];
    END;

    % Module containing the automation, human-device interface, and environment models
    AIE: MODULE =
    BEGIN
         INPUT Submitted: BOOLEAN
         OUTPUT Ready: BOOLEAN
         % Human actions as INPUT variables
         ...
         % Human-device interface information as OUTPUT Variables
         ...
         % Device automation and environment variables as LOCAL Variables
         ...
         INITIALIZATION
              Ready = True;
              ...
         TRANSITION
         [
              NOT (Ready OR Submitted) -->
                   Ready' = TRUE;
              []... Submitted AND Ready -->
                   Ready' = FALSE;
                   % Transition assignment for the environment model
                   ...
                   % Transition assignment for the automation model
                   ...
                   % Transition assignment for the human-device interface model
                   ...
         ];
    END;

    % System module
    System: MODULE = HumanTask [] Mission [] AIE;

    % Specification properties
    ...
END
```

FIGURE 4 General format of the base Symbolic Analysis Laboratory (SAL) input file used for representing formal models in our example. The file starts with the definition of constants and types. This is followed by the definition of four separate modules: *Mission* represents the human operator's mission goals; *HumanTask* represents the human task behavior the human operator uses to interact with the system to fulfill the mission goals; *AIE* represents the behavior of the automation, human–device interface, and environment; and *System* represents the composition of all three of the other modules to form the larger system. The file ends with a definition of the specification properties that will be checked against the model. The actual SAL files can be found at http://www.fmhfe.com/IJAP2012

HumanTask module; the HumanTask module communicates human actions (each represented as a single Boolean variable that is true when the action is being performed and false otherwise) to the AIE module; and the AIE module communicates environment and human–device interface information to the HumanTask module.

To allow the HumanTask behavior module to have time to respond to changes in the human–device interface and, conversely, to allow the AIE module to respond to human actions, a coordination protocol controls when each module is allowed to transition (Bolton & Bass, 2010a). This "handshake" protocol allows the two modules to take turns transitioning between states according to their internal transition logic. The module allowed to transition is determined by the values of two Boolean variables: *Submitted,* an output variable from the HumanTask module; and *Ready,* an output variable from the AIE module.

For the HumanTask module, the coordination logic is automatically generated when an instantiated EOFM's XML code is translated into SAL (Bolton et al., 2011). However, the AIE half of the protocol must be implemented manually. The remainder of this section describes the variables and transition logic of the formal model.

Operational environment. The operational environment of an actual aircraft includes many components such as the weather and air traffic. Herein, the environment is modeled as the relative distance (*Position*) of the aircraft from the capture position on the glideslope. The aircraft starts at a position where the glideslope diamond is not alive. The aircraft proceeds up to the capture position and begins to descend on the glideslope, a process that will take 18 sec. Thus, the relative position of the aircraft from the initial position is discretized into intervals (0–18) where the aircraft passes from one interval to the next in one second (Figure 5). This means the aircraft speed and altitude are abstracted into the position.

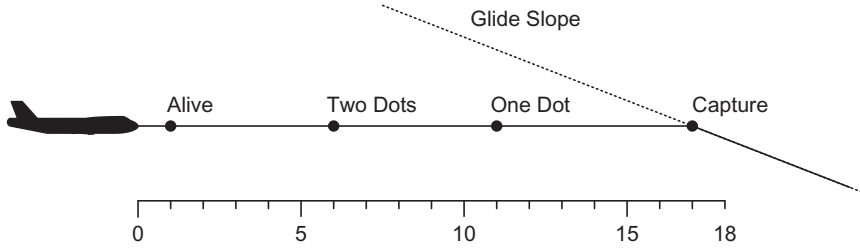

FIGURE 5 The position of the aircraft relative to the glideslope.

In the SAL input file, Position is represented as a local variable in the AIE module. It is initialized to 0 and, under "Transition assignment for the environment model" (Figure 4), a next state assignment advances the position: "Position' = Position + 1." Further, an additional clause on the second guard in the AIE module ("Position < 18") limits what positions are considered.

Device automation. The formal model of the device automation (Figure 6) represented the functionality of the aircraft's landing gear, spoilers, and landing gear doors. The ignition was not explicitly modeled in the device automation, although it was modeled as part of the human–device interface.

The landing gear (Figure 6a) starts in the up position as it would before the pilot performs the Before Landing checklist. When the pilot pulls the landing gear lever (*PullGearLever*) the landing gear transitions (is deployed) to the down position.

As would happen in an actual approach, the landing gear doors (Figure 6b) start in the closed position. When the pilot pulls the landing gear lever, the doors

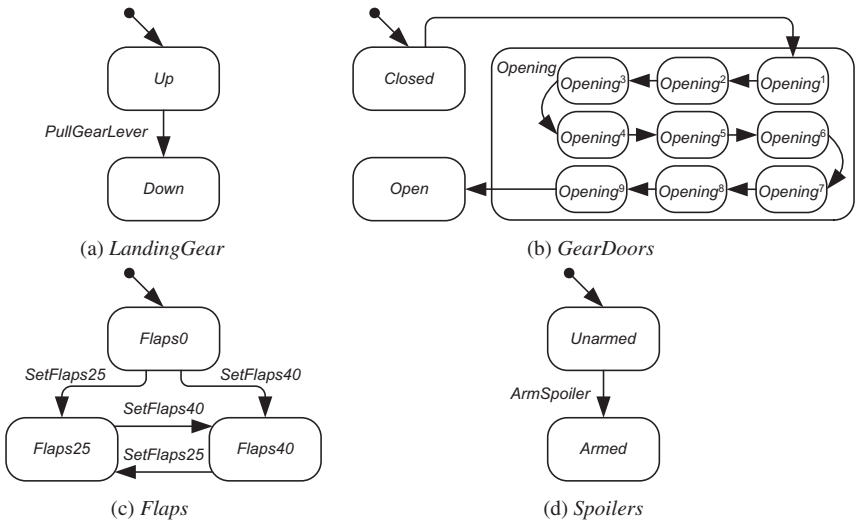

FIGURE 6 State transition representation of the system automation formal model. (a) Landing gear. (b) Landing gear doors, where the number of *Opening* state transitions are set to a constant (nine *Opening* states) and the total door opening time is 10 sec. (c) Flaps. (d) Spoilers. In the Symbolic Analysis Laboratory (SAL) input file, each of these is represented as a local variable. Each is initialized to the state (value) indicated by the arrow with a dot. Next state assignments with condition logic are used under "Transition assignment for the automation model" in the AIE module to control transitions between states (values). For example, the next state assignment for *Flaps* (c) would take the form: "Flaps' = IF SetFlaps25 THEN Flaps25 ELSIF SetFlaps40 THEN Flaps40 ELSE Flaps ENDIF."

begin to open. In the model it takes a constant amount of time (10 sec) for the doors to completely open, where the amount of time corresponds to the number of distance positions (Figure 5) the aircraft has passed (1 sec per interval).

The flaps (Figure 6c) start in the clean configuration (0°), as would be the case before the Before Landing checklist. Herein, the flaps can be set to 25° and 40°.

The spoilers (Figure 6d) are unarmed, as they would be at the beginning of an approach. They are armed when the pilot pulls the lever to arm the spoilers (*ArmSpoilers*).

Human–device interface. The formal model of the human–device interface (Figure 7) represents the state of the flight deck controls and indicator lights associated with arming the spoilers, the landing gear doors, the landing gear, the flaps, the glideslope indicator, and the ignition, all of which change state in response to changes in the automation, the environment, and human actions.

The state of the glideslope indicator (Figure 7a) is dependent on the position of the aircraft (Figure 5). Initially the glideslope indicator's diamond is inactive. At position 1 it becomes alive. It indicates "two dots" at position 6 and "one dot" at position 11. At position 17, it indicates capture.

The state of the ignition is indicated by the ignition switch and an indicator light. The ignition switch (Figure 7b) starts in the unflipped state. It transitions when the pilot flips the switch. The ignition light (Figure 7c) is on when the switch is flipped and off when it is un-flipped.

The state of the landing gear and landing gear doors are indicated by the human–device interface's gear lever (Figure 7d), three landing gear lights (Figure 7e), and the gear doors light (Figure 7f). The gear lever starts in the unpulled position. It becomes pulled when the pilot pulls the gear lever. The three landing gear lights are off whenever the landing gear is up and on when it is down. The gear doors light is dependent on the landing gear doors' state. When the doors are either open or closed, the light is off. Otherwise it is on.

The state of the flaps is indicated by the gauge the pilot uses to set the angle of the flaps (Figure 7g). It reflects the state of the flaps as determined by the automation.

The state of the spoilers (armed or unarmed) is indicated by the arming lever (Figure 7h) and the spoiler indicator light (Figure 7i). The lever starts out in the unpulled position and transitions to pulled when the pilot performs the action for arming the spoilers. The indicator light is dependent on the state of the spoilers from the automation. If the spoilers are armed then the light is on. Otherwise the light is off.

Human mission. Airline policy might dictate whether or not pilots should arm the spoilers or manually deploy them during landing. Pilots might also prefer to use a particular spoiler option (see Flight Safety Foundation, 1974b). A pilot

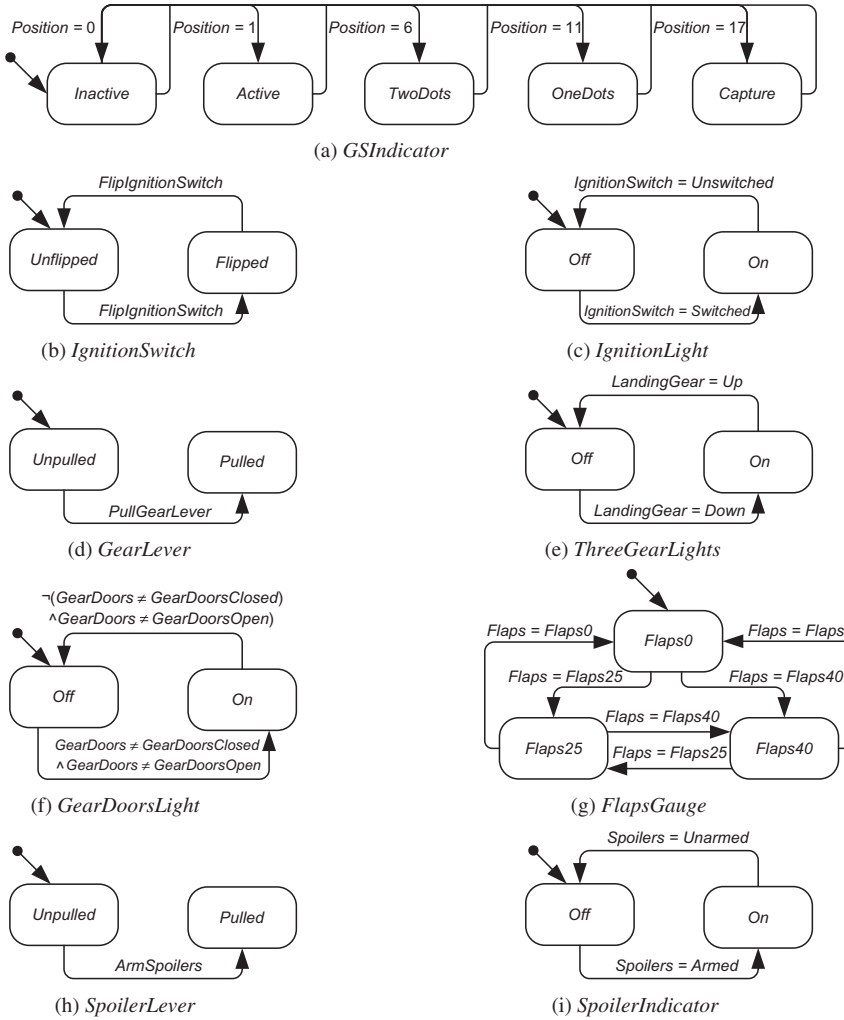

FIGURE 7 State transition representation of the formal model of the human device interface. (a) Glideslope indicator. (b) Ignition switch. (c) Ignition light. (d) Landing gear lever. (e) Three landing gear lights. (f) Gear doors light. (g) Flaps gauge. (h) Spoiler arming lever. (i) Spoiler indicator light. In the Symbolic Analysis Laboratory (SAL) input file (Figure 4), each of these is represented as an output variable. Each is initialized to the state (value) indicated by the arrow with a dot. Next state assignments with condition logic are used under "Transition assignment for the human-device interface model" in the AIE module to control transitions between states (values).

who prefers to manually deploy the spoilers will do so once the aircraft touches down. This preference constitutes the mission model as a Boolean variable *PreferToArmSpoilers,* which can be either true or false. In the SAL input file, this is represented in the Mission module (Figure 4) where *PreferToArmSpoilers* is an output variable that is initialized to be either true or false: "PreferToArmSpoilers IN {TRUE, FALSE}."

Human task behavior modeling. An EOFM was instantiated to represent the pilot task behavior for performing the Before Landing checklist (Degani, 2004). An EOFM is instantiated as an XML file using the EOFM notation (Bolton et al., 2011). In this particular model, a pilot human operator has access to input variables from the human–device interface (*GSIndicator, IgnitionLight, IgnitionSwitch, GearLever, GearDoorsLight, ThreeGearLights, FlapsGauge, SpoilerLever,* and *SpoilerIndicator*) and the human mission (*PreferToArmSpoilers*). The pilot model generates human action outputs representing actions performed through the human–device interface: flipping the ignition switch (*FlipIgnitionSwitch*), pulling the landing gear lever (*PullGearLever*), setting the flaps to either 25° or 40° (*SetFlaps25* and *SetFlaps40,* respectively), and pulling the lever to arm the spoilers (*ArmSpoilers*).

The visualization of this model is shown in Figure 8. The pilot can override the ignition (*aOverrideIgnition*) before the glideslope indicator diamond is alive if

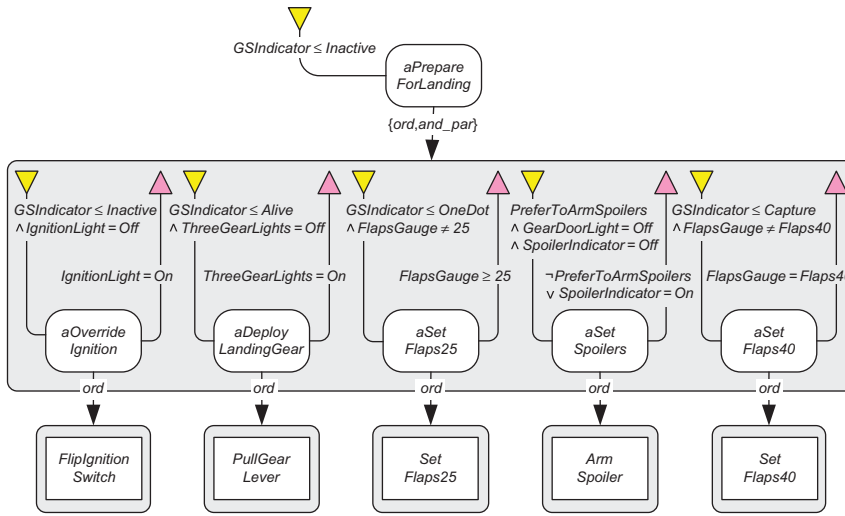

FIGURE 8 Visualization of the Enhanced Operator Function Model (EOFM) task model for the Before Landing checklist. Note that the decomposition of *aPrepareForLanding* can have either the *ord* or *and_par* decomposition. The actual EOFM files can be found at http://www.fmhfe.com/IJAP2012 (color figure available online).

the ignition light is off, thus fulfilling the first item on the Before Landing checklist. For the second item on the checklist, the pilot can deploy the landing gear (*aDeployLandingGear*) if the glideslope indicator is alive and the three landing gear lights are off. When the glideslope indicator reads one dot, the pilot can set the flaps to the intermediate 25°. The pilot can arm the spoilers, the third item on the Before Landing checklist, through the *aSetSpoilers* activity if he or she prefers to arm the spoilers, and both the spoiler indicator and landing gear doors lights are off. If the pilot does not prefer to arm the spoilers or the spoiler indicator is on, the activity will complete without the pilot pulling the lever to arm the spoilers (*ArmSpoiler*). Once the glideslope indicator reaches the capture position, the pilot can set the flaps to 40° (*aSetFlaps40*), the fourth item on the Before Landing checklist. Note that because the aircraft rudder and annunciator panel are not included in the system model, the step for checking the annunciator panel is not represented in the task model.

Pilots use the Before Landing checklist to guide them (Degani, 2004). Although pilots generally follow checklist items in order, they can complete them out of sequence. To model both of these conditions, two task models were created: one where the pilot will always perform the task in order (enforced by an *ord* decomposition for *aPrepareForLanding*) and one where he or she can perform them in any order (an *and_par* decomposition for *aPrepareForLanding*).

The EOFM task model instances were translated into SAL code and incorporated into the larger formal system model. In its original XML form, the human task behavior models were represented in 86 lines of code. The translated SAL versions were represented in 155 lines of code.

Specification

We created specifications that asserted properties about the system that we want to be true using linear temporal logic (LTL; Emerson, 1990). An LTL specification property is represented using variables from the formal model, common logic operators, and temporal operators. For the purposes of this article, we only use the temporal operator **G,** which asserts that a condition must be true for all paths through a model.

For safety related to spoiler arming, we use LTL to specify that if the aircraft is landing (at position 18), the spoilers should be armed if that is the pilot's preference:

$$\mathbf{G}\left(\begin{array}{l}(Position = 18)\\ \Rightarrow \left(\begin{array}{l}(PreferToArmSpoilers \wedge Spoilers = Armed)\\ \vee (\neg PreferToArmSpoilers \wedge Spoilers \neq Armed)\end{array}\right)\end{array}\right) \qquad (1)$$

For safety related to premature spoiler deployment, we use LTL to specify that we never want it to be true that the spoilers are armed when the gear doors are in the process of opening (not closed and not open):

$$\mathbf{G} \neg \begin{pmatrix} Spoilers = Armed \\ \wedge\ GearDoors \neq Closed \\ \wedge\ GearDoors \neq Open \end{pmatrix} \tag{2}$$

Specification properties are incorporated into the SAL input file under Specification properties. Each specification is given a specific name and asserts which module it is referring to. Both Equations 1 and 2 refer to the System module. For the purpose of this example, Equation 2 is represented in the SAL input file as: "prematurespoiler : THEOREM System -|G(NOT(Spoilers = Armed AND GearDoors /= GearDoorsClosed AND GearDoors /= cGearDoorsOpen))."

Apparatus

All verifications were completed using SAL–SMC 3.0, the SAL symbolic model checker. Verifications were conducted on a workstation with a 3.0 gigahertz dual-core Intel Xeon processor and 16 gigabytes of RAM running the Ubuntu 9.04 desktop.

SAL–SMC is a command-line program. Thus, to perform a model checking analysis, an analyst must run SAL–SMC while pointing to the appropriate SAL input file and specification property. For example, to check Equation 2 the analyst would run the command "sal-smc aircraft prematurespoiler." More information on configuring and running SAL can be found in the tutorial by de Moura (2004).

Model checker output was redirected from the command line into files (see Cooper, 2011). Standard output contained basic model checker output and verification statistics (including verification time and number of visited states) that were stored in one file. The actual verification results either indicated that the theorem was "proved" or listed a counterexample showing what states led to the safety specification violation. This was redirected from standard error output to a separate file. Any counterexamples that were produced were evaluated using our visualizer (Bolton & Bass, 2010c).

ANALYSIS PHASE 1: EXPLORING DIFFERENT MISSIONS AND CHECKLIST-GUIDED BEHAVIOR

In the first analysis phase, we evaluated how different pilot behavior for performing the Before Landing checklist would impact spoiler safety. This was accomplished by performing formal verifications for both Equation 1 and Equation 2 on two versions of the formal system model: one in which the pilot model

incorporated the *ord* decomposition operator (activities normatively performed in order) and one in which the *and_par* decomposition operator was used (activities to be performed in any order). The nature of the mission model allowed pilots to prefer arming or not arming the spoiler for both human behavior models.

Verification Results

For the model employing the human task behavior with the *ord* decomposition operator, both Equations 1 and 2 verified to true (239 visited states in 1.14 sec and 239 visited states in 1.12 sec, respectively). For the system model using the human task behavior with the *and_par* decomposition operator, Equation 1 verified to true (725 visited states in 1.18 sec). However, the verification of Equation 2 returned a counterexample after 0.58 sec. The counterexample revealed that the pilot's first action was to arm the spoilers. The pilot followed this by deploying the aircraft landing gear, an action that resulted in the landing gear doors opening. Thus a violation of Equation 2 occurred with the landing gear doors opening while the spoilers were armed.

Exploring Potential Solutions

There are a number of different ways in which this problem could be addressed. With respect to the automation, the spoiler system could be redesigned so that the spoilers could be armed while the landing gear doors are opening without risk of premature deployment, making Equation 2 irrelevant. However, such a solution might require an expensive retrofit. Another possible solution would be to implement a forcing function (Norman, 1988) to prevent the pilot from being able to arm the spoilers before the landing gear doors are open. However, this solution would also have associated expense and could artificially limit the procedures pilots could use to mitigate emergencies.

Another solution could change the pilot's mission through policy changes, where pilots would only be allowed to deploy spoilers manually. This modification eliminates the violation without introducing a new one: Equations 1 and 2 verified to true in just over 1 sec having visited 118 states for the *ord* model and in 204 states for the *and_par* model. However, this might also not be a desirable solution, as it could lead to an increase in pilots manually deploying spoilers prematurely, a problem known to occur with manual spoiler deployment (Degani, 2004; Flight Safety Foundation, 1974a, 1974b).

An additional option is to alter the pilot's task through additional policy or training (Basnyat, Palanque, Bernhaupt, & Poupart, 2008). In this situation, the *and_par* task model could be made irrelevant if pilots always performed the Before Landing checklist activities in order. Training could also address the criteria pilots use for arming the spoilers. For example, if the precondition has the

additional constraint that the three landing gear lights must be on before a pilot arms the spoilers, the discovered violation is eliminated without adding any additional violations: Equation 1 and Equation 2 both verified to true in just over 1 sec having visited 239 states for the *ord* model and 431 states for the *and_par* model.

ANALYSIS PHASE 2: EXPLORING VARIATIONS IN THE BEHAVIOR OF THE AUTOMATION

There can be anomalous conditions in device automation or the environment that can impact system performance and thus contribute to checklist noncompliance. In this second analysis phase, we explored variations in the behavior of automated systems with respect to their impact on system safety properties. Herein, we are concerned with how the time delays associated with aging hydraulic systems might influence the pilot's ability to perform the Before Landing checklist. We assume that the aging hydraulic systems for opening the landing gear doors can delay the door opening process by up to 7 sec beyond the nominal 10 sec. We can exploit the modularization of our architecture to replace the previous device automation model with one in which the landing gear opening can take between 0 and 7 additional sec (in 1-sec increments), making the range of potential landing gear opening times between 10 and 17. The formal system model incorporated the normative task behavior models with the *ord* decomposition and the *and_par* decomposition, both using the corrected precondition discussed at the end of the previous section.

Verification Results

When Equation 2 was checked with the *and_par* decomposition operator model, it verified in 1.23 sec having visited 1,931 states. However, when Equation 1 was checked with the same model, a counterexample was returned after 1.43 sec:

1. Initially, the pilot (preferring to arm the spoilers) correctly flipped the ignition switch to override the ignition.
2. At position 1, the pilot correctly initiated landing gear deployment and the opening of the gear doors, which would take 16 sec to open.
3. At position 11, having reached the one dot glideslope position, the pilot correctly set the flaps to 25°.
4. At position 17, the landing gear doors light turned off because the doors finished opening and, because glideslope was captured, the pilot correctly set the flaps to 40°.
5. The aircraft proceeded to position 18 with the landing gear doors light turned off and without the spoilers being armed.

When Equation 2 was checked with the *ord* decomposition operator model, it verified in 1.18 sec having visited 1,403 states. However, Equation 1 returned a counterexample after 1.45 sec:

1. Same as before.
2. Same as before except it would take 17 sec for the gear doors to open.
3. Same as before.
4. The pilot then waited to arm the spoilers because the landing gear doors were still opening and thus the landing gear doors light was on.
5. The aircraft proceeded to position 18 when the landing gear doors light finally turned off. However, the spoilers were not armed.

Exploring Potential Solutions

This example illustrates how anomalous or degraded behavior in device automation can impact the pilot's ability to effectively perform procedures, as delays in the hydraulic systems prevented the pilot from performing the activities necessary for preparing the aircraft for landing.

There are a number of ways to address this. As previously mentioned, the aircraft could be modified to allow spoilers to be armed while the aircraft landing gear doors are opening without the risk of premature deployment. This would allow spoilers to be armed much earlier in the process, and thus there would be less risk of this process being impacted by time delays.

Alternatively, airlines could require that pilots manually deploy the spoilers. When the mission model was updated to reflect this change, Equation 1 verified to true for both the *and_par* (1,002 visited states in 0.82 sec) and *ord* (676 visited states in 0.81 sec) formal system models. However, as was previously noted, this might increase the risk of premature manual spoiler deployment during actual landing.

Another solution could be to establish a maintenance policy on the aircraft to ensure that the landing gear door hydraulics open within a given time. For example, if maintenance can ensure that landing gear doors can open in under 15 sec, Equation 1 verifies to true for both the *and_par* (1,439 visited states in 1.39 sec) and *ord* (1,021 visited states in 1.35 sec) formal system models.

DISCUSSION

The modern flight deck is very complex due to the interactions among the pilot, airline policies, regulations, human–device interfaces, and automation in a dynamic environment. Thus, even though pilot checklist noncompliance is often cited as a cause of failures in this environment, failures often result from the

interaction of these elements. We have presented an instrument landing approach application to illustrate how analyses offered by model checking can be used to evaluate the safety of flight deck checklists while considering these interactions. We exploited the flexibility of our formal modeling architectural framework (Figure 3) to explore how different human task behavior and device automation models could contribute to safety concerns and to investigate potential interventions.

Our approach has a unique utility in that it will consider all possible interactions that are supported by the model. However, this is not to say that this method will replace other methods for evaluating flight deck checklists: simulations scale better than model checking analyses and offer the ability to model and measure quantities that might not be easily discretized (Hu, 2008); surveys, interviews, observations, and other forms of human subject testing (de Brito, 2002; Degani & Wiener, 1991, 1993; Landry & Jacko, 2006) allow for the collection of data that is currently not possible with formal methods; accident report investigations (Degani & Wiener, 1991, 1993) give designers and analysts data with face validity; improved training (Burian & Barshi, 2003) helps ensure that pilots will properly follow checklists; and improved design elements (fonts, wording, etc.) constructed around consistent design and corporate philosophies (Burian, 2004; Degani & Wiener, 1997) help create more usable checklists. Thus, the method discussed here is not a replacement for any of these other techniques, but rather a complementary analysis that allows the interactions between system elements to be considered more exhaustively than they would be otherwise. Despite its utility, there are still extensions of our method suitable for future investigation.

Higher Fidelity Applications

The presented application showed how problems with flight deck checklists could be discovered with our method. The model was kept simple and was used to find known problems with spoilers related to the Before Landing checklist. However, the real power of this method would reside in its ability to find previously unknown problems. Thus, future work should go toward using this method to investigate new flight deck procedure designs using higher fidelity models.

Erroneous Human Behavior

Phase 1 analyses demonstrated how the impact of variations in human task behavior might affect system safety and showed how different solutions could also be evaluated. Whereas the task model with the *and_par* decomposition could be viewed as noncompliant behavior, other forms of noncompliant erroneous human behavior are often associated with system failures (Hollnagel, 1993; Jones, 1999; Reason, 1990; Sarter & Alexander, 2000; Shappell & Wiegmann, 1997).

Several researchers have shown that it is possible to manually incorporate patterns of erroneous human behavior into task analytic behavior models (Bastide & Basnyat, 2007; Fields, 2001; Paternò & Santoro, 2002). These techniques should be capable of being supported by our method. Future work should investigate if this is indeed the case and determine whether these techniques would be appropriate for evaluating checklist noncompliance. Further, we have investigated ways of automatically generating erroneous human behavior within previously normative task analytic behavior models (Bolton & Bass, 2010b, 2011). Future work should investigate whether our method is capable of supporting multiple erroneous human behavior generation methods, and when each is appropriate when evaluating noncompliance with checklists.

Degraded Performance of the Automation and the Environment

The second phase of analyses demonstrated how degraded performance (landing gear deployment time) of the automation could impact system safety. This analysis was based on a known issue (Degani, 2004). There are, of course, many additional ways in which degraded performance of the automation could impact the performance of an approach procedure. Future work should investigate how such factors could be included.

In our aircraft instrument approach procedure, the environment model was particularly simple. Although it is theoretically possible to do so, no variation in the environment model was considered in the analyses. In reality, there are a number of environment conditions that could impact the instrument-landing approach procedure such as weather, visibility conditions, and the relative locations of other aircraft. Future work should investigate how these could be incorporated into the formal model of the environment so that their impact could be evaluated. Further, disciplines such as error injection (Voas, 1997), reliability engineering (Department of Defense, 1987), and resilience engineering (Hollnagel, Woods, & Leveson, 2006) offer theories about how errors, degraded performance, and uncertainty can be modeled as part of automation and environment behavior in complex systems. Future work should investigate how theories and practices from these fields could potentially be used to extend our method so that it could systematically incorporate degraded or erroneous automation and environmental behavior.

Multiple Human Operators

To date, limited multioperator systems have been evaluated using our approach (Bass et al., 2011). This is a distinct limitation as modern, complex systems

have multiple human operators that interact with them. In fact, an instrument landing approach in a commercial aircraft would be performed by a pilot team. Multioperator systems would require extensions to the architectural framework (Figure 3) to address the following:

1. Multioperator systems could have human–device interfaces that are shared across human operators.
2. Device automation might be associated with each human–device interface, remotely located, or both.
3. Human operators could have the ability to communicate directly or through the human–device interface.

Future work should investigate how our architecture should be extended to accommodate these and other issues related to modeling systems with multiple human operators.

Comparison of Analysis Results

In a more complex example, the analyst might have many more specifications he or she would want to verify against each design variation. In such a circumstance, it is likely that the analyst would want to compare the results of different analyses to diagnose similar problems that exist between designs, and to compare and contrast the performance of the designs. We have developed a visualization tool that helps to interpret counterexamples (Bolton & Bass, 2010c), but such evaluations would require counterexample visualization enhancements. Future work should investigate tools and analyses that might be incorporated to facilitate these sorts of comparative assessments.

ACKNOWLEDGMENTS

The work documented here was completed while Matthew L. Bolton was a Senior Research Associate for San José State University Research Foundation at NASA Ames Research Center. The work has been supported in part by NASA Cooperative Agreement NCC1002043 to the National Institute of Aerospace (NIA; UVA-03-01, Sub-Awards 2623-VA and 2723-VA), Grant Number T15LM009462 from the National Library of Medicine (NLM), and NASA contract NNA10DE79C (NextGenAA: Integrated model checking and simulation of NextGen authority and autonomy). The content is solely the responsibility of the authors and does not necessarily represent the official views of the NIA, NASA, NLM, or the National Institutes of Health.

REFERENCES

Basnyat, S., Palanque, P. A., Bernhaupt, R., & Poupart, E. (2008). Formal modelling of incidents and accidents as a means for enriching training material for satellite control operations. In *Proceedings of the Joint ESREL 2008 and 17th SRA-Europe Conference* [CD-ROM]. London, UK: Taylor & Francis.

Bass, E. J., Bolton, M. L., Feigh, K., Griffith, D., Gunter, E., Mansky, W., & Rushby, J. (2011). Toward a multi-method approach to formalizing human–automation interaction and human–human communications. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics* (pp. 1817–1824). Piscataway, NJ: IEEE.

Bass, E. J., Ernst-Fortin, S. T., Small, R. L., & Hogans, J. (2004). Architecture and development environment of a knowledge-based monitor that facilitate incremental knowledge-base development. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, *34*, 441–449.

Bastide, R., & Basnyat, S. (2007). Error patterns: Systematic investigation of deviations in task models. In Coninx, K., Luyten, K., & Schneider, K. A. (Eds.), *5th International Workshop on Task Models and Diagrams for Users Interface Design* (pp. 109–121). Berlin, Germany: Springer.

Bolton, M. L., & Bass, E. J. (2009). A method for the formal verification of human interactive systems. In *Proceedings of the 53rd annual meeting of the Human Factors and Ergonomics Society* (pp. 764–768). Santa Monica, CA: Human Factors and Ergonomics Society.

Bolton, M. L., & Bass, E. J. (2010a). Formally verifying human–automation interaction as part of a system model: Limitations and tradeoffs. *Innovations in Systems and Software Engineering: A NASA Journal*, *6*(3), 219–231.

Bolton, M. L., & Bass, E. J. (2010b). Using task analytic models and phenotypes of erroneous human behavior to discover system failures using model checking. In *Proceedings of the 54th annual meeting of the Human Factors and Ergonomics Society* (pp. 992–996). Santa Monica, CA: Human Factors and Ergonomics Society.

Bolton, M. L., & Bass, E. J. (2010c). Using task analytic models to visualize model checker counterexamples. In *Proceedings of the International Conference on Systems Man and Cybernetics* (pp. 2069–2074). Piscataway, NJ: IEEE.

Bolton, M. L., & Bass, E. J. (2011). Using task analytic behavior models, strategic knowledge-based erroneous human behavior generation, and model checking to evaluate human–automation interaction. In *Proceedings of the IEEE International Conference on Systems Man and Cybernetics* (pp. 1788–1794). Piscataway, NJ: IEEE.

Bolton, M. L., Siminiceanu, R. I., & Bass, E. J. (2011). A systematic approach to model checking human–automation interaction using task-analytic models. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, *41*, 961–976.

Burian, B. K. (2004). Emergency and abnormal checklist design factors influencing flight crew response: A case study. In *Proceedings of the International Conference on Human–Computer Interaction in Aeronautics* [CD-ROM]. Menlo Park, CA: AIAA.

Burian, B. K., & Barshi, I. (2003). Emergency and abnormal situations: A review of ASRS reports. In *Proceedings of the 12th International Symposium on Aviation Psychology* [CD-ROM]. Dayton, OH: Wright State University.

Burian, B. K., Barshi, I., & Dismukes, K. (2005). *The challenge of aviation emergency and abnormal situations* (Tech. Rep.). Moffett Field, CA: NASA Ames Research Center.

Clarke, E. M., Grumberg, O., & Peled, D. A. (1999). *Model checking*. Cambridge, MA: MIT Press.

Cooper, M. (2011). I/o redirection. In *Advanced bash-scripting guide: An in-depth exploration of the art of shell scripting*. Retrieved from http://tldp.org/LDP/abs/html/io-redirection.html

de Brito, G. (2002). Towards a model for the study of written procedure following in dynamic environments. *Reliability Engineering and System Safety*, *75*, 233–244.

Degani, A. (2004). *Taming HAL: Designing interfaces beyond 2001*. New York, NY: Macmillan.

Degani, A., Heymann, M., & Shafto, M. (1999). Formal aspects of procedures: The problem of sequential correctness. In *Proceedings of the 43rd Annual Meeting of the Human Factors and Ergonomics Society* (pp. 1113–1117). Santa Monica, CA: Human Factors and Ergonomics Society.

Degani, A., & Wiener, E. L. (1991). *Human factors of flight-deck checklists: The normal checklist* (Tech. Rep.). Moffett Field, CA: NASA Ames Research Center.

Degani, A., & Wiener, E. L. (1993). Cockpit checklists: Concepts, design, and use. *Human Factors*, *35*, 345–359.

Degani, A., & Wiener, E. (1997). Procedures in complex systems: the airline cockpit. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, *27*, 302–312.

de Moura, L. (2004). *SAL: Tutorial* (Tech. Rep.). Menlo Park, CA: Computer Science Laboratory, SRI International.

de Moura, L., Owre, S., & Shankar, N. (2003). *The SAL language manual* (Tech. Rep. No. CSL-01-01). Menlo Park, CA: Computer Science Laboratory, SRI International.

Department of Defense. (1987). *Reliability test methods, plans, and environments for engineering development, qualification and production* (Tech. Rep. No. MIL-HDBK-781). Washington, DC: Author. Retrieved from http://src.alionscience.com/pdf/MIL-HDBK-781.pdf

Emerson, E. A. (1990). Temporal and modal logic. In J. Leeuwen, A. R. Meyer, M. Paterson, & D. Perrin (Eds.), *Handbook of theoretical computer science* (pp. 995–1072). Cambridge, MA: MIT Press.

Fields, R. E. (2001). *Analysis of erroneous actions in the design of critical systems* (Unpublished doctoral dissertation). University of York, York, UK.

Flight Safety Foundation. (1974a). *Accident description: Air Canada flight 621.* Retrieved from http://aviation-safety.net/database/record.php?id=19700705-0

Flight Safety Foundation. (1974b). *Accident description: Loftleidir flight 509.* Retrieved from http://aviation-safety.net/database/record.php?id=19730623-0

Graeber, R. C., & Moodi, M. M. (1998). Understanding flight crew adherence to procedures: The procedural event analysis tool (PEAT). In *Annual international air safety seminar* (pp. 415–424). Alexandria, VA: Flight Safety Foundation.

Hollnagel, E. (1993). The phenotype of erroneous actions. *International Journal of Man-Machine Studies*, *39*(1), 1–32.

Hollnagel, E., Woods, D. D., & Leveson, N. (2006). *Resilience engineering: Concepts and precepts*. Surrey, UK: Ashgate.

Hu, A. (2008). Simulation vs. formal: Absorb what is useful; reject what is useless. In Yorav, K. (Ed.), *Proceedings of the Third International Haifa Verification Conference on Hardware and Software: Verification and Testing* (pp. 1–7). Berlin, Germany: Springer.

Jones, P. M. (1999). Human error and its amelioration. In Sage, A. P., & Rouse, W. B. (Eds.), *Handbook of systems engineering and management* (pp. 687–702). Malden, MA: Wiley.

Landry, S., & Jacko, J. (2006). Improving pilot procedure following using displays of procedure context. *International Journal of Applied Aviation Studies*, *6*(1), 47–70.

Lautman, L., & Gallimore, P. (1987). Control of the crew caused accident: Results of a 12-operator survey. *Boeing Airliner*, 1–6.

National Transportation Safety Board. (1994). *A review of flightcrew-involved major accidents of U.S. air carriers, 1978 through 1990* (Tech. Rep. No. NTSB/SS-94/01). Washington, DC: NASA Ames Research Center.

National Transportation Safety Board. (2001). *Runway overrun during landing, American Airlines flight 1420, McDonnell Douglas MD-82, N215AA, Little Rock, Arkansas, June 1, 1999* (Tech. Rep. No. NTSB/AAR-01/02). Washington, DC: Author.

Norman, D. A. (1988). *The psychology of everyday things*. New York, NY: Basic Books.

Paternò, F., & Santoro, C. (2002). Preventing user errors by systematic analysis of deviations from the system task model. *International Journal of Human-Computer Studies*, *56*, 225–245.

Reason, J. (1990). *Human error*. New York, NY: Cambridge University Press.
Sarter, N. B., & Alexander, H. M. (2000). Error types and related error detection mechanisms in the aviation domain: An analysis of aviation safety reporting system incident reports. *The International Journal of Aviation Psychology*, *10*, 189–206.
Sarter, N. B., & Woods, D. D. (1995). How in the world did we ever get into that mode? Mode error and awareness in supervisory control. *Human Factors*, *37*(1), 5–19.
Shappell, S. A., & Wiegmann, D. A. (1997). A human error approach to accident investigation: The taxonomy of unsafe operations. *The International Journal of Aviation Psychology*, *7*, 269–291.
Voas, J. (1997). Fault injection for the masses. *Computer*, *30*(12), 129–130.
Wing, J. M. (1990). A specifier's introduction to formal methods. *Computer*, *23*(9), 8, 10–22, 24.

Manuscript first received: May 2011