May 2015

Software in Military Aviation and Drone Mishaps: Analysis and Recommendations for the Investigation Process

Veronica Foreman
Georgia Institute of Technology

Francesca Favaro
Georgia Institute of Technology

Joseph Saleh
Georgia Institute of Technology

Christopher Johnson
University of Glasgow

Recommended Citation

Veronica Foreman, Francesca Favaro, Joseph Saleh, and Christopher Johnson. "Software in Military Aviation and Drone Mishaps: Analysis and Recommendations for the Investigation Process" *Reliability Engineering and System Safety* (2015). https://doi.org/10.1016/j.ress.2015.01.006

Software in Military Aviation Mishaps: Analysis and Recommendations for the Investigation Process

Veronica L. Foreman [a], Francesca M. Favaró [a], Joseph H. Saleh [a,*], Christopher W. Johnson [b]

[a] *School of Aerospace Engineering, Georgia Institute of Technology, USA*

[b] School of Computing Science, University of Glasgow, Scotland, UK

Abstract: Software plays a central role in military systems. It is also an important factor in many recent incidents and accidents. A safety gap is growing between our software-intensive technological capabilities and our understanding of the ways they can fail or lead to accidents. Traditional forms of accident investigation are poorly equipped to trace the sources of software failure, for instance software does not age in the same way that hardware components fail over time. As such, it can be hard to trace the causes of software failure or mechanisms by which it contributed to accidents back into the development and procurement chain to address the deeper, systemic causes of potential accidents. To identify some of these failure mechanisms, we examined the database of the Air Force Accident Investigation Board (AIB) and analyzed mishaps in which software was involved. Although we have chosen to focus on military aviation, many of the insights also apply to civil aviation. Our analysis led to several results and recommendations. Some were specific and related for example to specific shortcomings in the testing and validation of particular avionic subsystems. Others were broader in scope: for instance, we challenged both the investigation process (aspects of) and the findings in several cases, and we provided recommendations, technical and organizational, for improvements. We also identified important safety blind spots in the investigations with respect to software, whose contribution to the escalation of the adverse events was often neglected in the accident reports. These blind spots, we argued, constitute an important missed learning opportunity for improving accident prevention, and it is especially unfortunate at a time when Remotely Piloted Air Systems (RPAS) are being integrated into the National Airspace. Our findings support the growing recognition that the traditional notion of software failure as non-compliance with requirements is too limited to capture the diversity of roles that software plays in military and civil aviation accidents. The identification of several specific mechanisms by which software contributes to accidents can help populate a library of patterns and triggers of software contributions to adverse events, a library which in turn can be used to help guide better software development, better coding, and better testing to avoid or eliminate these particular patterns and triggers. Finally, we strongly argue for the examination of software's causal role in accident investigations, the inclusion of a section on the subject in the accident reports, and the participation of software experts in accident investigations.

Keywords: Military aviation; mishap; accident investigation; software; Remotely Piloted Air Systems (RPAS).

* Corresponding author. Tel: + 1 404 385 6711; fax: + 1 404 894 2760
E-mail address: jsaleh@gatech.edu (J. H. Saleh)

1. INTRODUCTION

This work is at the intersection of three research areas: software in safety-critical systems; aviation accidents; and military aircraft mishaps, including Remotely Piloted Air Systems, hereafter referred to as RPAS. Software has become pervasive and central to the operation of many military systems [1]. By the same token, software is playing an increasing role, albeit not well understood, in system accidents. This is especially true in avionics where software is a safety-critical element for flight operation and control, and, either independently or in conjunction with the hardware (e.g., actuators, sensors) or liveware (e.g., human-in-the-loop).

There are significant limitations in our current understanding of the role that software plays in accidents–for example, it can be very difficult to identify the particular stage or activity in the development lifecycle that led to a potential failure. This is compounded by a lack of suitable methods for software testing (with the purpose of capturing and eliminating these contributions before systems are fielded for operation). In addition, software contributions are often neglected in accident investigation reports even though it seems clear that code played a significant role in the causes and contributory factors that led to an eventual failure.

Why focus on military aviation accidents? Our motivation is threefold:

(i) In the past five years (2008 – 2012), the United States (U.S.) Air Force reported mishaps in aviation resulting in $5 billion in lost equipment and injuries (excluding ground-related mishaps) and over 9,000 lost workdays. In FY 2012, the Air Force experienced 31 manned aircraft and RPAS Class A mishaps (damages exceeding $2 million) [2]. These are significant penalties, and any contribution for improvements in this area can help save lives, preserve capabilities, and reduce costs associated with these military aviation accidents.

(ii) Operational environment and flight parameters of military aviation offer in some cases distinctive flight conditions that can act as triggers of (deeply buried) software pathogens or lurking software defects. As such, **military aviation accidents offer a unique learning opportunity, especially when software is involved, to address software defects, which are unlikely to be triggered under nominal operating conditions, across the entire military fleet and with spillovers to commercial aviation**. It is surprising that safety research has made very little use of the wealth of information and learning opportunities military aviation accidents provide.

(iii) Our third motivation for focusing on military aircraft accidents stems from the rapid introduction of military avionics

into civil operations. In particular, we are concerned with the rapid integration of RPAS into the National Air Space. The U.S. Air Force reported 33 RPAS mishaps in FY 2012, (11 Class A resulting in damages exceeding $2million, 4 Class B with damages exceeding $0.5 million, and 18 Class C with damages exceeding $50,000) [2]. Ten RPAS vehicles were completely destroyed and the total losses were estimated at $66 million. The RPAS case studies examined in this work are intended to contribute to the public debate about the integration of drones in the National Air Space. In previous publications, Johnson [3,4] examined several operational incidents with RPAS and highlighted among other things the inadequate reliability of the systems examined compared with that of traditional air support. One important finding in these works is the relationship between the decision to rush the deployment of RPAS and the higher technical and organizational risks factors in operating them, which make mishaps much more likely to occur. Other publications in this area are by Tvaryanas [5] and Williams [6] who provided some statistical results related to RPAS mishaps between 1997 and 2003, and an examination of human factors in interface design for remotely piloted aircraft.

To investigate the role of software contributions to military aircraft accidents, we examined for this work the accident database of the U.S. Air Force Accident Investigation Board (AIB). This yielded more than three hundred and fifty incidents. An exhaustive analysis helped to identify those in which software was clearly involved. We have already argued that the role of software is often neglected in many investigations, given that many investigators lack a formal training in software engineering [7]. There may have been other incidents where the role of software was not mentioned in the report even though it played some role in the causes of a mishap. However, our aim here was to highlight both the failure mechanisms and recurrent patterns of software failure, identified using existing investigation techniques. We have argued elsewhere [8] about the limitations of the notion of "software failure" currently defined in various professional standards (e.g., IEEE), as it does not capture the diversity of software's roles in accidents—some of which do not involve a "failure." We developed in its stead the notion of software contribution to adverse events. Later sections will expand on this notion.

The remainder of this work is organized as follows. In Section 2, we provide background information on software contributions to adverse events, the Air Force Accident Investigation Board, and the methodology adopted in this work. In Section 3 we introduce the case studies, and we analyze in detail each accident and the contribution of software to the mishap. Section 4 concludes this works with a brief synthesis and various recommendations.

2. BACKGROUND AND CONTEXT

In this section we provide the context for our study. The Accident Investigation Board is briefly discussed and a visualization

tool for representing accidents is introduced. This tool will be used in Section 3 in conjunction with our case studies.

2.1. Beyond software failures

The analysis of software-related problems (and, by the same token, software-related aviation accidents) involves the notions of software failure and fault. A software failure represents the inability of code "to perform its required function within specified performance requirements"; while a software fault is an "incorrect step, process, or data definition in a computer program" [9]. Some variations on these definitions exist in the technical literature, but it is generally the case that software failure is defined in terms of non-compliance with requirements and the result of flawed implementation of said requirements. Software fault is typically causally subsumed under failure.

Leveson [10] challenged this notion of software failure as non-compliance with requirements. Similarly, Knight [11] concluded, *"it is clear that we have difficulty stating exactly what software is required to do, hence rendering the definition of software failure ineffective."* In a previous work [8], we argued for a shift in perspective, that software failure is not necessarily ill-defined, but that the notion itself is too limited to capture the diversity of ways in which software can contribute to accidents. We examined several **cases in which software complied with its requirements yet directly contributed to or led to an accident. Such cases would not fall under the traditional category of software failure, yet they deserve careful attention by accident investigators and researchers for the feedback they can provide to software developers and testers and other stakeholders**. These cases emerged for different reasons, either because of missing or flawed requirements – such as those not suited to the system's operational conditions at the time of the accident – or because of unconsidered operational scenarios (which would cause off-nominal and/or untested input values to the software).

In short, a growing number of authors [8, 10, 11] have argued that the notion of software failure is narrow in scope. In its stead, we adopt the expression of "software contributions to adverse events" as a better way to frame these issues. Section 3 provides specific examples and analyses of such contributions.

2.2. The Air Force Accident Investigation Board (AIB)

As mentioned, the case studies for this work are drawn from the U.S. Air Force Accident Investigation Board (AIB) Class A mishaps. Following an aviation mishap, the Air Force conducts two separate investigations: the first is known as a "safety investigation", and it is conducted by the Safety Investigation Board (SIB) with the objective to "prevent future mishaps [and] assess possible force-wide implications on the combat readiness of systems" involved in the accident; the second is known as the "accident investigation" and it is "conducted [by the AIB] to provide a report for public release" [12]. This dichotomy is interesting and its justification in terms of cost, benefits, and relevance, will be examined later in this work. The SIB takes

precedence over the AIB investigation. The SIB report consists of two parts: the first part is factual and data-centric (the collection of evidence and facts), while the second contains privileged information, and as such, it is not releasable outside the Air Force. The factual part of the SIB report is passed to the AIB investigation and is then incorporated in the AIB report. After reviewing the first part of the SIB report, the AIB may gather additional evidence and conduct its own investigation (e.g., interview witnesses, perform additional tests). The AIB Board President then appends to the report his/her opinion about what caused the mishaps using a "clear and convincing evidence" standard [12]. The AIB reports are completed in 60 to 90 days following the accident. Once they are submitted and approved internally, the AIB reports can be released to the public. The information used to reconstruct the accident sequence in our case studies is based on the factual part of the corresponding AIB reports (which is common with the SIB reports). Our work focuses on the AIB reports; this is justified because it provides the recommendations that can be used to guide the future development and operation of both civil and military software.

The Air Force classifies its mishaps into four categories, Class A to Class D, from the most grievous adverse events (Class A) to the less severe ones (Class D) based on the total cost of damage and/or the severity of injury to personnel or fatalities[1]. Accident classes are used to determine the appropriate investigative and reporting procedures [15]. Class A mishap is defined as an adverse event in which "the resulting total cost of damage to the Government and other property is $2,000,000 or more, a DOD aircraft is destroyed, or an injury or occupational illness results in a fatality or permanent total disability" [2]. In this work, we only examine Class A mishaps; this is justified not only by the consequences of these incidents but also by the observation that these investigations offer the most detailed evidence about the causes and contributory factors leading to an incident. We recognize that the other classes of mishaps offer an important area for future research.

2.3. A visual tool for accident sequences: the STEP diagram

Accidents often stem from concurrent interactions between multiple actors; it can be difficult to provide an overview of these interactions across many pages of prose. To complement our discussion of the case studies and facilitate the understanding of the events involved, we adopt a modified version of the Sequential Timed Event Plotting (STEP) diagram. This graphical technique was developed by Hendrick and Benner [16]; STEP diagrams are structured as matrices, each row contains the actions or events ascribed to one agent, and leading to the accident. Agents are indicated in the first column of the matrix. Agents in STEP need not have volition. Events are typically represented when agents change state and/or interact with each other. In this manner, the diagram provides a high level map of the interactions between different actors [8]. We clarify that STEP is only

[1] In the following, we use the term mishap as the overarching category of aviation adverse events under consideration. In the military context mishaps are divided into four classes (A to D) based on their severity [13]. Similarly, in the civilian and commercial context, mishaps are classified as either accidents (more grievous consequences) or incidents (less severe consequences) [14].

used here as a visualization tool to help the reader better follow the concurrent events leading to the accident. We include a STEP diagram for each of the case studies here analyzed, and we identify different pieces of software as actors to highlight their contributions to the accident sequence.

2.4. Other tools, and organizational considerations

Two clarifications and caveats are in order before we proceed with the case studies:

Caveat #1: STEP is not the only tool available for examining and visualizing accidents. Other tools exist in the literature, such as Acci-Map, FRAM, STAMP, MORT, and several others. These and other are briefly reviewed by Sklet [17], and a detailed comparison of STEP and FRAM can be found in reference [18]. Acci-Map and STAMP are reviewed and compared in reference [19]. These techniques differ in their scope and emphasis (given their underlying model of accident causation)[2], and the way they display information. In some cases the conceptual differences are minor. For example, the Acci-Map tool developed by Rasmussen [20] examines and displays failures across several levels, including the regulatory level, the organizational level, and the technical level of a system where the accident occurred. The STAMP tool, developed by Leveson [21, 22], examines controls and constraints across several hierarchical levels, and then identifies failures in this control structure. Accidents in the STAMP model result from inadequate controls or missing constraints, and a taxonomy of control failures is provided to characterize such omissions or inadequacies. For the purpose of our study, STEP was chosen as the appropriate visualization tool for two reasons:

1. Its visual appeal/ease of construction when mapping the temporal relationships between the different events leading to the accident (per agent);

2. Its flexibility in terms of the level of abstraction included within each diagram. As mentioned previously, the AIB reports differ significantly in terms of the information provided about the role of software in the incident. Some of the alternate visualization techniques, listed above, assume that greater detail can be provided in terms of the evidence that is available about an incident – for example, the regulatory context of military software development is seldom if ever considered within the AIB reports.

Caveat #2: Our case studies belong to a specific class of adverse events termed organizational accidents or system accidents[3]. The two qualifiers, "organizational" and "system", are used to indicate on the one hand an organizational contribution to

[2] For example, safety barriers, organizational/managerial contributions to accidents, coupling between different levels of hierarchy within a socio-technical system, etc.

[3] These were initially termed "man-made disasters" [23] and "organizational accidents" [24]. This class of adverse events was also examined by sociologists and management scientists, for example in Perrow's *Normal Accidents* [25], and in the collective works on *High Reliability Organizations* [26, 27, 28]. For a brief review of this literature, see reference [29].

accident causation beyond the traditional technical and human factors, and on the other hand a recognition that accidents can occur due to the *interactions* between the elements of a system, rather than failures of the elements themselves [22]. Two distinctive features of a system accident are [30]:

i. The temporal depth of causality: the causes and contributory factors behind an accident extend beyond the immediate operational context, with a build-up of accident pathogens long before an initiating event triggers an accident.

ii. Their diversity of agency: The safety value chain, that is, groups and individuals who influence or contribute to the accident occurrence/prevention, extends far beyond the immediate victims, who may or may not have contributed to the accident.

The mishaps examined next, as in any other system accident/incident, have multiple causes includes technical, operational, organizational factors. While we acknowledge the diversity of these factors, we chose to focus only on software's roles in these mishaps, first in order not to dilute the scope and message of this work, and second, as we shall see, because the role of software is often neglected in accident investigations. As such, the choice of focusing on software's contributions to mishaps should not be interpreted as disregarding the importance of other contributory factors.

3. CASE STUDIES: ANALYSIS AND DISCUSSION

The case studies selected for this work were chosen to highlight recurrent patterns the contribution of software to military aviation accidents. Given disclosure restrictions and the potential limitations of existing investigatory techniques, our decision to focus on public reports means that **this analysis is indicative and not exhaustive**. As previously mentioned, the information for each of the following Class A mishaps derives from the factual part of the AIB report. For each case study, background information relevant to the mishap is first presented. The software contributions are then analyzed. Additional concerns that do not specifically relate to software but highlight important safety issues are included at the end of each case.

3.1. T1-A Jayhawk Accident, August 16, 2003

The first case study involves a T1-A jet trainer, a twin-engine typically used for advanced stage pilot training. Upon landing, the aircraft overran Runway 21 at Keesler Air Force Base in Mississippi with a student and an instructor pilot onboard. As the aircraft departed the runway, the left main landing gear became stuck in the mud. The aircraft pivoted counter clockwise as the left wingtip impacted the ground, and the nose and right landing gears collapsed. The aircraft continued to rotate, and came to a stop 190 feet off the end of the runway. While there were no fatalities associated with this accident, the cost of damage to the aircraft and associated military property was estimated at $2.5 million [31].

Several contributing factors were involved in this accident and we briefly mention them before examining the role of software: (1) the landing runway was wet due to scattered thunderstorms moving through the area; (2) the instructor configured the aircraft for an approach based on distance information from a different radio frequency (Gulfport) than the one of the landing runway at Kessler AFB (why this error occurred was not discussed in the report; it may reflect an important issue in human factors and training, and it deserves careful attention in accident investigations as a consequence, not just a cause). This misconfiguration directly led to the initiating event of this accident sequence in the following manner: Keesler AFB is about 5 miles east of Gulfport, and as a result, the pilots believed they were 5 miles further away from their assigned runway. When the situation was recognized, the aircraft was 800 feet above the correct glide scope at 3 miles from the runway, and the pilot adopted a higher sink rate and slightly faster airspeed than recommended. The accident sequence that unfolded will be examined shortly.

It is important to recognize that while training and human factors were clearly involved in the T-1A accident, the software strand was arguably neglected in the investigation, and yet it contributed significantly to the incident.

3.1.1 Software and the braking system on the T-1A

An overview of the aircraft braking system is necessary to understand the software contribution to this accident. To prevent unintentional in-flight activation, neither the spoilers nor the landing gear speed brakes on the T1-A can be engaged until either the left or the right ground safety switch indicates aircraft touchdown. Touchdown is confirmed once a ground safety switch senses full compression of the landing gear struts for at least 2.5 seconds. As an additional safety precaution, the speed brakes become available if either one of the landing gear wheels reaches a spin speed greater than 37 knots.

Once touchdown is confirmed, the ground sensors switch to ‘Ground’ mode by removing the electrical ground within the sensor relays. The touchdown protection circuit is then opened and allows the use of the hydraulic pressure to engage the speed brakes. Failure to fully compress the struts, or full compression lasting less than 2.5 seconds, yields continued ‘Air’ mode software functionality. When in Air Mode, the software overrides all braking commands by the pilot, and the spoilers cannot be engaged.

This particular braking configuration and control, or variations on it, is present in both military and civilian avionics, and the current design presents a particular hazard. A similar runway overrun event involving the aborted takeoff of a Bombardier Learjet 60 occurred in September of 2008. Using a similar Air Mode and Ground Mode designation, the Full Authority Digital Engine controls (FADEC) software onboard the aircraft took invalid inputs from the failed main landing gear sensors and used

that information to override pilot commands to deploy the thrust reversers. The accident resulted in the deaths of the captain, first officer, and two passengers, as well as substantial damage to the aircraft and airport property (see reference [8] for details).

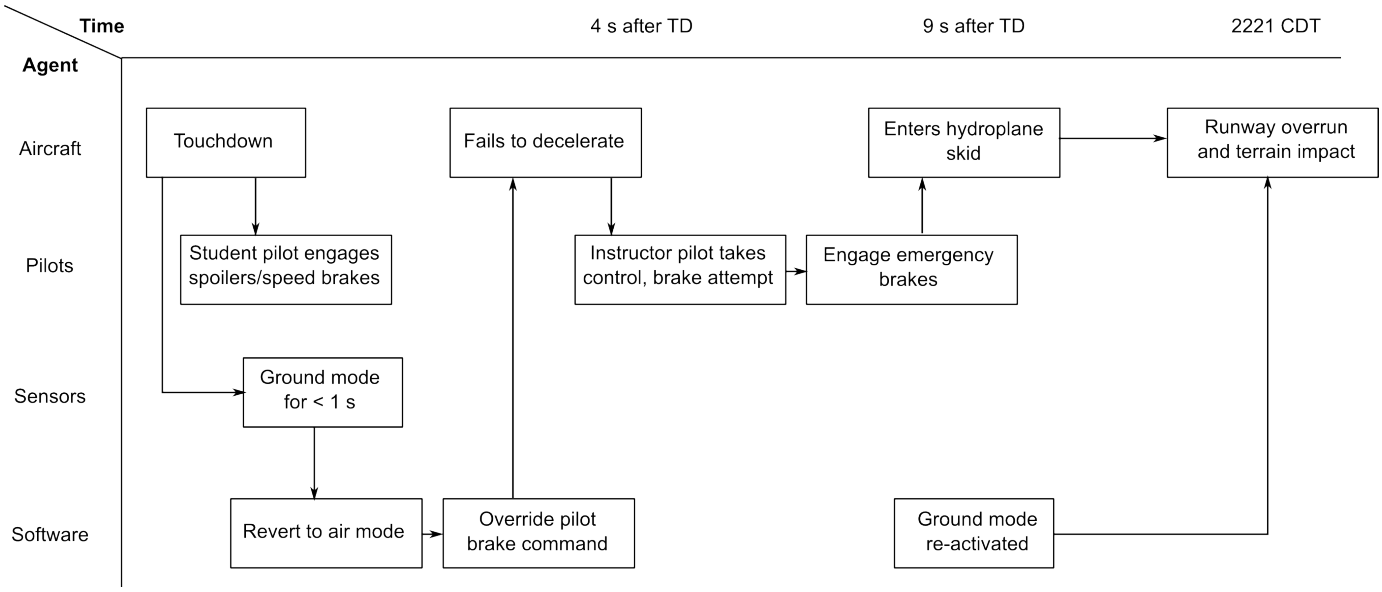

Figure 1. STEP Diagram representing the T1-A accident sequence.

3.1.2. The T-1A Accident

We resume the narrative of the T-1A case with the aircraft at 800 feet above the correct glide scope at 3 miles from the runway. The pilot recognizes the situation and adopts a higher sink rate and slightly faster airspeed than recommended. This is an off-nominal situation but not an unusual one; even when considering the human factors behind it, it does not contain the seeds of an imminent accident. The software, we argue, was the more important factor in this accident.

The aircraft touched down approximately 1500 feet down the runway, landing approximately 12 knots faster than recommended with a throttle setting at 7% above idle.

Upon touchdown, the pilot activated the brakes but the aircraft did not respond or decelerate. **The Flight Data Recorder shows that that "the aircraft was in Ground Mode for less than one second, and then reverted to Air Mode for nine seconds"** [31]. **During these crucial nine seconds, the flight software, by design, prevented the activation of the brakes and overrode the pilot's repeated commands to engage the brakes**. The instructor then engaged the emergency brakes, which did not have anti-skid protection (a serious hardware flaw or accident pathogen waiting to be activated), and as a result the wheels locked into position. The wet conditions on the runway transformed this situation into "skidding on water or hydroplaning" and resulted in the accident. Figure 1 captures the main events in this accident.

3.1.3. Software Contributions to the Accident

Software contributed to this accident in two important and related ways: (1) it estimated the aircraft to be in the air (Air Mode after touchdown) while it was on the runway and for several seconds; and (2) it had control authority to act on this wrong assessment and to override the pilot's repeated commands to engage the brakes. Said differently, a critical input by the pilot during a critical phase of the mission was disallowed by the software given its flawed assessment of the situation.

Why this mismatch emerged between the software estimated and actual flight condition is an important learning opportunity and ought to be thoroughly investigated to prevent recurrence of such accidents. This was not conclusively discussed in the report. What is known is that the flight control software entered Ground Mode for less than one second after touchdown, and reverted to Air Mode immediately afterward. **The possibility of Ground Mode disengagement after touchdown is an important flaw in the software design and may be ascribed to an unconsidered scenario and missing software requirement**. Since details about this mismatch are not available, we can only hypothesize that the likely cause for this was either an improper setting of the threshold on the Ground Safety Switch (GSS) sensors, or a flawed software logic or implementation of the constraints for triggering the Ground Mode as we discuss next: (1) a threshold of strut compression is required to validate Ground Mode. For a bumpy landing such as this one, the readings from the pressure sensors may have fluctuated and dropped below this threshold and the aircraft exited Ground Mode (recall the aircraft was in Ground Mode for less than one second before reverting to Air Mode); (2) the spin speed threshold for the landing gears should have averted this situation, but it was either implemented incorrectly in the software (as an AND constraint for example), or that the conditions on the runway did not result in the threshold spin speed for the wheels (37 knots). The report includes contradictory statements to this effect, but notes that the "wheels were spinning at 37 knots due to the speed of the aircraft at touchdown". It should be noted that the estimated spin speed was exactly that of the Ground Mode triggering threshold, making this statement in contradiction with other statements in the report, which suggested that the aircraft was in Air Mode.

We recommend to aircraft software developers (and acquisition officers to consider including requirements to the following effect), that once Ground Mode has been validated, the software should not be sensitive to the fluctuations of pressure sensors; this is particularly important in cases of bumpy landings, which are not uncommon in military and civilian aviation. In addition, it is a serious automation mistake for the software to be able to override the pilot trying to engage the brakes after the software has validated that the aircraft has touched down. A requirement to this effect can be included to eliminate this possibility.

Moreover, if the part in our second hypothesis of a flawed implementation of the spin constraint were correct, this would have important consequences for software testers to consider, (e.g., why was this situation not caught during software testing of the breaking system, and why was the system certified to comply with its requirements when in fact it was not).

This case study, its accident report, and many others we have examined also prompt us to make a recommendation to accident investigators: software's role in accidents has been conspicuously missing in accident investigation reports, even when that role was prominent as in this case. The absence of this feedback loop constitutes a missed learning opportunity for contributing more efficiently to accident prevention. We recommend that the AIB includes in the template of its reports a section on the role of software in the mishap, and both train and urge each Board president to carefully examine this role when warranted. We also recommend that software experts be appointed as part of the accident investigation team.

3.2. QRF-4C and F-22 Accidents (September 28, 2004 and May 13, 2011 respectively)

We examine in this subsection the QRF-4C remotely piloted aircraft accident, and we briefly mention an F-22 mishap as well. These two aircraft, with significantly different costs, complexities, and capabilities, experienced in one respect a similar software contribution to their accidents. This similarity, despite the widely different mission profiles, highlights one general pattern by which software contributions to accidents can arise, and the lessons learned for preempting it can be beneficial for a broad range of systems (and software development) beyond the peculiarities of the aircraft here examined.

3.2.1. QRF-4C Accident and System Background

On May 13, 2011, a QRF-4C remotely piloted aircraft en route from Tyndall Air Force Base in Florida to overwater airspace experienced an electrical malfunction. This malfunction significantly affected the flight control software and the aircraft performed highly unusual maneuvers (discussed next) not commanded from the ground. When the aircraft did respond to the ground command to perform a pre-defined All-Attitude Recovery (AAR) maneuver, the backup flight control software, which was executing the AAR instead of the primary one, did not have an angle of attack limiting protection for the maneuvers it commands and executes. As a result of this and the unusual attitude of the aircraft, the software-commanded maneuver resulted in a stall. The aircraft experienced several post-stall gyrations before impacting the water. There was no surface traffic around the crash area, and no injuries or fatalities ensued. The aircraft was unrecoverable, and the cost of the total loss of the aircraft was estimated at $2.8 million [32]. While this was in one sense a lucky outcome, it is sobering to reflect on the possibility of such a situation unfolding over inhabited areas and within the vicinity of commercial airliners. Serious considerations for testing and safety certification should be given before drones are integrated into the National Airspace.

The QF-4 is a modified F-4 Phantom fighter jet, which is used in support of the Weapons Evaluation Group of the Air Force, to supplement controller training and, in the case of the QRF-4, to provide an unmanned, full-scale target RPAS for air-to-air weapon system evaluation and testing. Some retired F-4s with remaining structural service life are converted to QF-4 in Mojave, California, and then flown to Tyndall AFB for evaluation. After conversion, the aircraft performs a **manned** [emphasis added] Systems Acceptance and Flight Evaluation (SAFE) qualifying flight. Once accepted, the aircraft is either qualified as "unrestricted" and it is flown with a pilot or "restricted", and it is flown as a RPAS (unmanned). From the information that can be gathered publicly, the RPAS version, which requires its own specific flight control software beyond the unrestricted manned version and some RPAS peculiar equipment, does not undergo flight-testing. We will revisit this observation after discussing the accident sequence.

A brief explanation of the QRF-4 computer architecture is necessary to understand the software contribution to this accident. The RPAS is remotely controlled from a ground control station. The operator can set up the uplink controls manually or by selecting an automatic sequence from a library of pre-defined maneuvers (push-button for automatic takeoff for example). The RPAS has two almost-redundant flight control systems, a primary one and a backup, and it can switch to the backup in case of failure or unresponsive primary control system. These systems are not fully redundant for a few reasons, one of which is that the primary flight control system has angle of attack limiting protection but no load factor (G) protection on its maneuvers; while the backup system provides load factor protection but no angle of attack protection. This system design choice is disconcerting, and while it was not investigated in the accident report, it was a clear causal factor in this accident, as we will see shortly. The absence of a limiting protection on the angle of attack in the backup control system can be conceived of as an accident pathogen lurking in the software, and it will be triggered by the specific circumstances of this case.

One last piece of background information is useful to understand the accident sequence: the RPAS has two AC power generators, left and right. A relay switch can disengage one in case of failure or power loss, and the remaining one can assume the electrical load of the aircraft. According to the AIB report, a condition that repeatedly occurs with this aircraft power system is that of a BUS TIE OPEN, which "indicates that the [two power] generators are out of synch either in phase or frequency". A work-around this condition was devised, and it consisted in cycling one generator off, then on. Additional information about the QRF-4 will be provided as we discuss the accident sequence next.

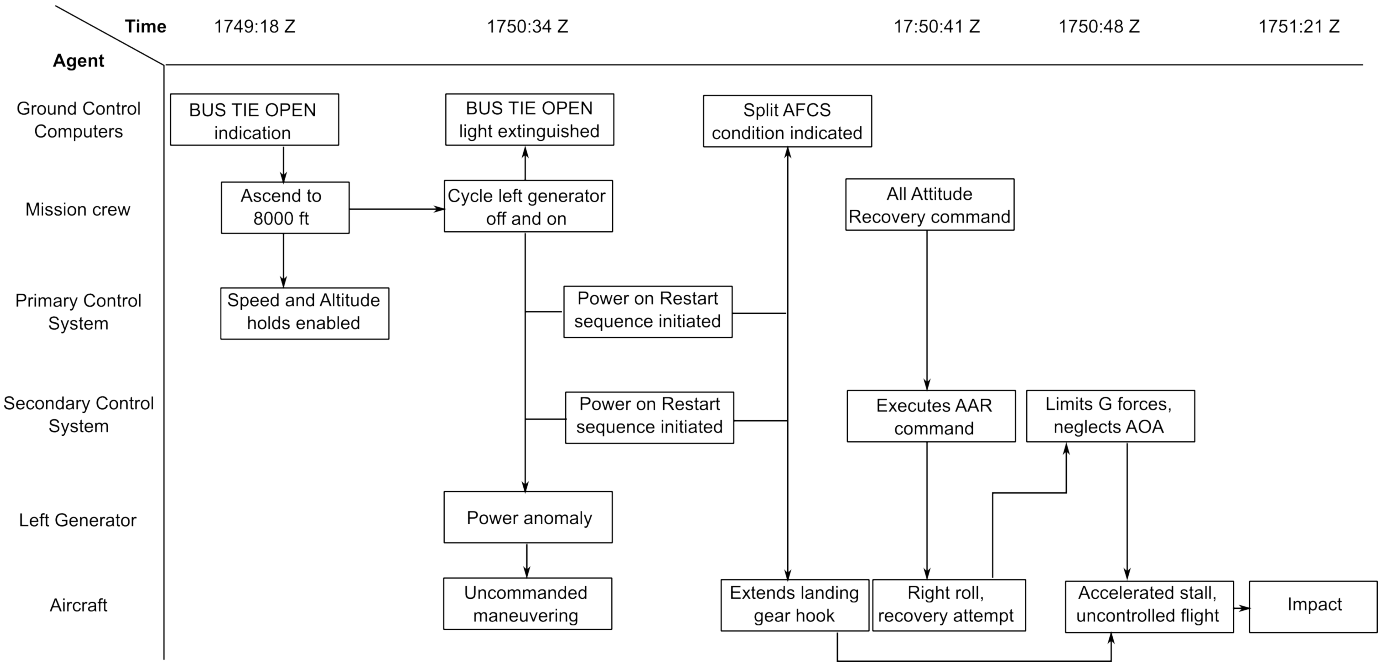

Figure 2. STEP Diagram representing the QRF-4C accident sequence.

3.2.2. Accident Sequence

Shortly after takeoff, the RPAS experienced an electrical malfunction and the ground station received a BUS TIE OPEN error message. The RPAS operator continued ascent of the aircraft to 8000 feet MSL before beginning to address this condition. In accordance with the QRF-4 checklist, the left generator was cycled off, and the BUS TIE OPEN indicator light went out. Three seconds later, the generator was cycled back on, and a power disruption resulted, triggering the accident sequence. The AIB report suggests this disruption was likely a transient current spike.

While this electrical power disruption was the initiating event of the accident sequence, it is misguided to consider it "the cause of the mishap" as the report's statement of opinion advances. Other causal factors in the accident sequence occurred that are more significant in leading to the accident. More importantly, the lessons to be learned for preventing future similar mishaps reside in these other factors, as we will argue shortly. The cycling of the generator and the resultant power disruption uncovered two major flaws in the architecture and software of the drone:

(i) A split flight control condition occurred in which the RPAS was controlled by the backup flight control system, while the remote operator was interacting with the primary flight control system. The result of this split condition is that the RPAS was unresponsive to most ground commands;

(ii) A more egregious software or architectural flaw was activated by the power disruption: in the case of power failure (probably during the short period of time when both flight control systems were affected by the current spike), the aircraft according to the accident report, "forgets it is in RPAS mode, and reverts to a normal F-4, referencing [cockpit] handle and switch positions", that is, it responds to commands from the unmanned cockpit.

As a result of (ii), the drone, while flying at about 350 knots, partially extended its landing gear and arresting hook. Several unusual and uncommanded maneuvers followed, including rapid yaw movements, pitch up, and significant roll, which placed the RPAS in a nearly inverted attitude. The remote pilot attempted to level the flight path but found the RPAS unresponsive. An All-Attitude Recovery (AAR) maneuver was then commanded, and despite the split condition, the RPAS did execute this maneuver. It is at this point that the software accident pathogen previously mentioned is activated: recall the backup flight software does not have an angle of attack limiting protection on its maneuvers. In executing the AAR, the angle of attack rapidly increased and resulted in a stall. The RPAS remained fully stalled until it impacted the water. Figure 2 captures the main events in this accident sequence.

3.2.3. Software Contributions to Accident

The web of causality in this accident, as in the previous one, involves many strands, and it is important to acknowledge the multidisciplinary nature of causes in such accidents. **These different causal strands lead to a portfolio of possible safety interventions, with each element in this portfolio having some degree of effectiveness in preventing recurrence**. For example, the AIB report noted that human factors were a causal factor in this accident, that the "ground console did not adequately communicate flight-critical cautions and warnings, [and that] pertinent data was scattered across [two] screens, requiring the controller to look in several to find and process relevant information." More importantly, the repeated electrical problems in the RPAS were not properly understood, and as such they could not be eliminated, but rather had to be worked around with power cycling.

Equally important were the software-related contribution to this accident, the emergence of the split condition (that it was possible in the first place, and more generally the inability of the computers to restart safely), the fact that the RPAS could still respond to commands from the cockpit (that this possibility was not eliminated in making a RPAS out of the F-4), and that **critical flight constraints on angle of attack and load factor were not properly implemented in both flight control software**. Why these conditions existed and were tolerated was not examined in the report, but we believe they constitute more fundamental causes to the accident than power disruption or ill-designed human interface, and they provide loci for efficient safety interventions to prevent future recurrence. Although details about the software and computer architecture were not

available, we believe these software-related contributions to the accident can be an “easy fix” for competent software developers — especially the faulty implementation of the flight parameter constraints. This situation also occurred in a mishap involving an F-22, and we will briefly mention it for the important lesson it provides to software developers and testers.

This case study further supports our previous recommendation for the AIB report to include in its template a section on software contributions to the accident, in the same way that in the current template, sections on maintenance and human factors for example are mandated. The absence of such a section perpetuates a software blind spot in military aviation accidents, and an important feedback loop and learning opportunities are missing.

We also make one RPAS-specific recommendation: recall that only the manned version of the refurbished F-4 undergoes a Systems Acceptance and Flight Evaluation (SAFE) qualifying flight. Since a number of changes and additions in equipment and software occur following this flight to convert the refurbished manned F-4 into a drone, we recommend that some amount of flight or ground testing occurs after these additions are made, a sort of abridged SAFE test for the RPAS. A number of adverse conditions may be uncovered in this manner and costly mishaps avoided later.

3.2.4. F-22 Raptor Accident, September 28, 2004

Another F-22 mishap had a similar software contribution to the QRF-4 accident. The F-22 is a significantly more complex and expensive system than the QRF-4, and it was operating a very different mission profile when the mishap occurred. The similarities in the role of the software in both mishaps, despite these fundamental system differences, suggest a general lesson for software developers, testers, and acquisition officers to study and adopt to prevent recurrence of this pattern of failure in a broad range of military systems.

An F-22 Raptor was flying a test mission near Edwards AFB in California when it entered the wake of a target aircraft. The aircraft experienced pitch oscillations and varying load factor and angle of attack conditions. These oscillations triggered the aircraft’s pre-programmed auto-pitch recovery (APR) sequence. The APR is a piece of the flight control software, which automatically corrects off-nominal pitch angle orientations using the horizontal stabilizers. The APR commanded full trailing edge up position and rejected all pilot inputs. While software for nominal flight condition includes load factor (G) limiting protection on its maneuvers, the APR does not. The APR, in executing this maneuver, greatly exceeded the structural limitations of the F-22 airframe and resulted in approximately $3.6 million worth of damage to the aircraft [33]. The pilot was able to return the aircraft to base safely following the mishap. Despite successful recovery, the mishap is considered as a potential precursor to other accidents with more serious consequences [34].

The software contribution to this accident resulted from a missing or neglected general constraint on a critical flight parameter, the load factor, in one part of the flight software, the APR (in the same way that a limiting angle of attack protection was missing in the backup flight system of the QRF-4). While this can be seen as a trivial case of forgetfulness, it can be indicative of other issues, which in turn offer several opportunities for eliminating this flaw before it finds its way into the software of fielded systems. We propose the following:

(i) **Requirement flow-down**: the AIB report does not address whether the load factor limitation was part of the requirements of the APR or not. This omission is significant and reflects more general concerns about the handling of software in incident investigations. It arguably also reflects a problem in software requirement flow-down: a general requirement that should always be satisfied (e.g., the load factor constraint) should be present in every individual piece of the flight software, not just in a few select automatic maneuvers or for nominal flight conditions.

(ii) **Software testing**: We cannot tell from the limited information in the AIB, whether or not the requirement on the limitation of the load factor *was* specified for the APR. We propose that software testing be adjusted to cover general requirements that should always be satisfied (again, such as the load factor constraint) in every piece of the flight software. This also suggests an opportunity for improving checking on compliance with requirements to capture and prevent this potential flaw from finding its way into the software of fielded systems.

The F-22 mishap can be the result of two very different failure root causes: (1) the flight envelope protection requirements were not made to all the components of the software (in which case, the recommendation would target the officers writing the flight requirements); (2) the requirements were made to all pieces of the flight code, but they were not implemented in one or mode pieces (in which case, the recommendations would target the coders). The fact that one cannot tell whether the missing software requirement was the result of (1) or (2) is an important message in and of itself: **that the accident investigation failed to look into and investigate this software contribution to the accident is a significant flaw in the investigation process and major missed learning opportunity to prevent accident recurrence**.

3.3. MQ-1B Predator Accident, May 7, 2011

The last case study involves a MQ-1B Predator RPAS that crashed into the Gulf of Aden on May 7, 2011. The RPAS experienced two uncommanded rolls on final approach due to a malfunctioning right aileron. The operator handled the first roll but was unable to counter the second one because of a particular software limitation. The RPAS impacted the water about a half-mile off the coast of Djibouti. The loss of the RPAS with one missile was estimated at $4.4 million [35].

Software played a minor role in this accident, electrical problems and operational factors were prominent. However, this incident also illustrates how important lessons can be learned even when avionics only play a contributory role in a mishap. Unlike the previous cases in which software played a prominent causal role in the accident, in this case, software became an issue far downstream of the accident sequence. Its contribution amounted to "preventing the prevention" of the accident, as we will explain shortly. This is a particular shade or primitive of causality and it is worth recognizing in accident investigations.

3.3.1. System Background

The Predator is typically operated by two crews, the first is known as the launch and recovery element (LRE), which is located on the same airfield as the aircraft, and the second is the mission control element (MCE), which operates the RPAS via satellite link from a remote location from the airfield. The LRE is "typically deployed in a theater of operations, where it will launch the aircraft, get it to specified altitude, accomplish a systems check", and then pass it on to the MCE. After the mission is completed, the RPAS is handed back to the LRE for approach and landing. This split operation will have some implications for this accident.

The RPAS has a primary flight control system and a backup system. In addition, the ailerons are controlled by their respective wing control modules, for example the right aileron is controlled by the Right Wing Control Module (RWCM). An electrical malfunction in the RWCM, and its subsequent failure, were the initiating event of this accident sequence and the main cause of the mishap, as we discuss next. Many of the Class A mishaps investigated since 2000 are attributed to 'catastrophic electrical failures.' Furthermore, the RWCM was a causal factor in the loss of another MQ-1 in Afghanistan on May 8, 2009 (the total loss of the RPAS was valued at $4.6 million). This electrical failure pattern, including the performance of the RWCM, deserves careful attention across the entire fleet of MQ-1B in service.

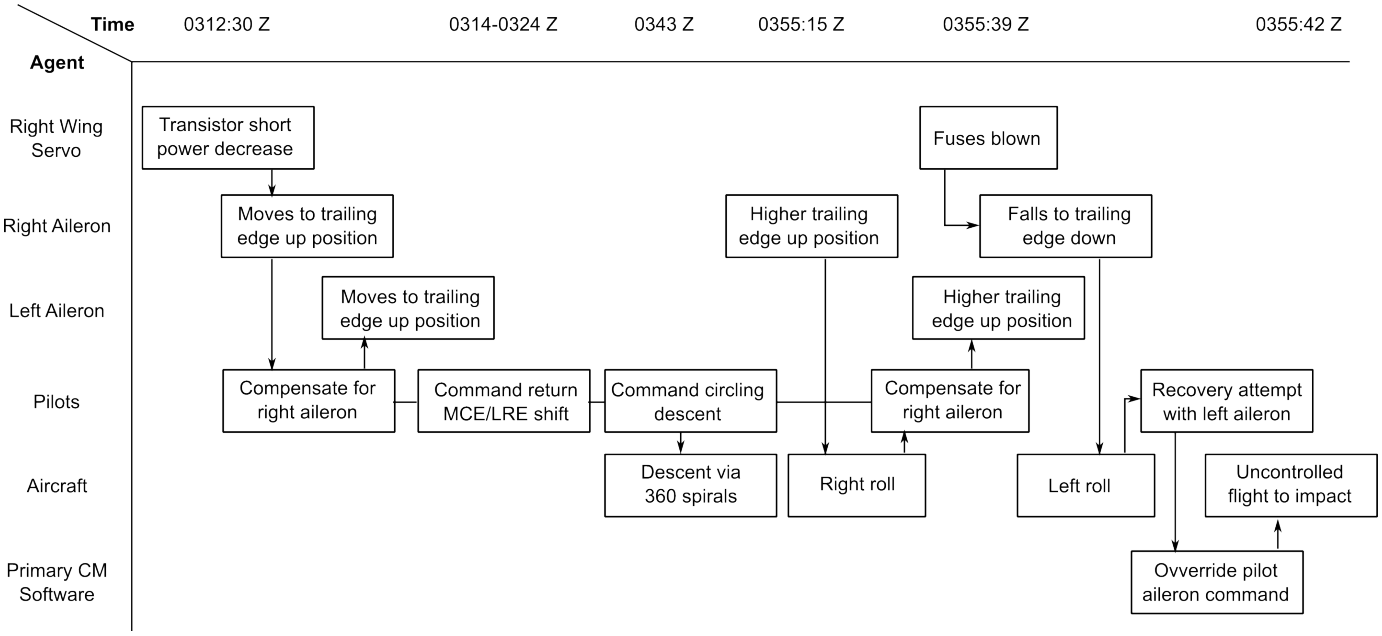

Figure 3. STEP Diagram representing the MQ-1B accident sequence.

3.3.2. Accident Sequence

Approximately 57 minutes into the flight, following a nominal launch and operating team transition from LRE to MCE, a transistor in the right wing control module shorted out, initiating the accident sequence. The transistor short resulted in an excessive current draw, which caused the RWCM to fail and led the actuator servo to place the right aileron in a trailing edge up position.

The electrical architecture and specific hardware design that led to this particular failure mode in the right wing/aileron is not available for examination, and it was not investigated in the accident report. However, this failure mode might have been averted if the designers had implemented a fail-safe principle to the ailerons, which, in case of wing control module failure, would lock the ailerons in their level position [36]. As a result of the RWCM failure, the ground operators could no longer control the right aileron. The right servo, which was drawing excessive current but still functioning, held the right aileron in the trailing edge up position. This situation will change on final approach, as we will see shortly.

The operators worked around this situation by raising the left aileron to compensate for the "stuck up" right aileron. They turned the RPAS around toward the base, and handed its operation over to the LRE crew at 16,000 feet instead of the standard 6,500 feet handoff altitude. This added about 5 minutes to the flight, a relevant detail in this accident sequence.

To address the excessive altitude, the LRE crew performed two descending spirals, which likely imposed further stresses on the ailerons. Upon final approach, the right aileron moved again, uncommanded, to a higher trailing edge up position, and the resulting right roll was countered by the left aileron. Two minutes before landing, the accumulated stresses reached a breaking point and precipitated the accident: the right servo, which had been drawing excessive current for about 40 minutes, failed and could no longer hold the right aileron up. As a result, the right aileron dropped to a trailing edge down position.

At this point, the software contribution to the accident emerges: to counter this drop of the right aileron and the left roll that ensued, the RPAS operator attempted to command a right roll. This required a left aileron down position. However, the "MQ-1B software prohibits the […] aileron from moving to a position below level." In other words, while the hardware allows the deflection of ailerons from trailing edge up to trailing edge down positions, the software restricts this degree of freedom from trailing edge up to the level position. The presence of this hard software constraint is surprising, and the rationale for its existence was not examined in the report; again reiterating the need to pursue the underlying causes of software failure back in the procurement and development lifecycle.

Due to the inability to symmetrically match the right aileron down position, the RPAS continued its left roll until it impacted the water. Figure 3 captures the main events in this accident sequence.

3.3.3. Software Contributions to Accident and Additional Concerns

As noted earlier, software became an issue in this mishap late downstream in the accident sequence, and its contribution to the accident was of a different causal nature than in the previous cases.

While electrical problems and operational factors were the main causal factors in this accident, as indicated earlier, software played a particular causal role by *preventing the prevention of the accident*. An operational solution to preventing the accident was hardware-available, and the crew attempted to execute it; the software-imposed limitation prevented this execution and led directly to the loss of the drone. Said differently, despite the significant electrical problems with this drone, the unresponsive right aileron, and the ill-advised operational decisions (e.g., the handover at 16,000 feet and subsequent spiral descent), these adverse conditions would not have resulted in an accident had it not been for the software contribution and the limitation it imposed on the ailerons deflection.

We recognize this software contribution reflects a particular shade or primitive of causality, probably of lesser familiarity and significance than the traditional causal factors in accidents [37]. It is, however, worth examining in accident investigations,

after the traditional causes have been identified. The implications of such contributions to software developers and testers constitute a fruitful venue for future research.

Additional concerns with this RPAS, not software-related, are worth mentioning as well, as they are distinctive and offer an opportunity for further reflection and lessons learned for preventing future mishaps.

Many maintenance issues with this RPAS were logged within a short period of time prior to the accident. For example, in the 60 days prior to the mishap, electrical problems occurred repeatedly, and the Right Wing Control Module (RWCM) experienced **partial failure on three occasions**. Within this same time period, the RPAS experienced problems with all exhaust temperature sensors on two occasions, the first resulting in a loss of 500 feet in altitude due to loss of thrust, and 1500 feet the second time. Pressure sensors were also found deficient and **replaced on three occasions**. The Secondary Control Module (SCM) was **replaced six times** in the 60 days prior to the mishap, and the Primary Control Module was removed twice within the same time period (once for a software upgrade). The AIB report notes that "maintenance correctly followed the fault analysis guidance and performed the required operational checks", and the AIB President noted in his statement of opinion: "since maintenance performed all tasks correctly […] I do not find any maintenance procedures, supervision, or actions of any individual as causal or contributory to this mishap." Fault analysis guidance as well as maintenance policies and guidelines were clearly deficient in this case. Replacing a Control Module six times in less than sixty days on the same RPAS is probably more expensive and less efficient than understanding the cause of failure and addressing it. A system perspective on maintenance with the possibility of a graduated response, which accounts for the repeated-ness of similar failures, is worth exploring (and testing) by military agencies.

Finally, we note that a non-military version of this RPAS is already in use in the continental U.S., by Customs and Border Protection agency for example[4]. In light of the discussion in this subsection, **we strongly urge that these non-military RPAS be subjected to much more stringent safety testing, certification, and regulations than their military counterparts. An accidental loss of a few thousand feet during flight, or uncommanded maneuvers, can have dramatic consequences for a nearby commercial airliner**.

4. CONCLUSION AND RECOMMENDATIONS

Software has wiggled its way into our daily lives to an extent we may not have taken the full measure of yet. Power and energy distribution, transportation, communications, and a host of network systems and equipment, intrinsic to our ways of life,

[4] The Customs and Border Protection agency had one of their RPAS crash in Nogales, Arizona, in April 2006. The accident is carefully examined in [38].

are intensely software-dependent. Our present understanding of software failure mechanisms and contributions to system accidents is significantly lacking. **There is a widening safety gap between the software-intensive capabilities we create and our understanding of the ways they can fail or contribute to accidents, and hence our ability to prevent accidents** [1].

To advance the thinking about these issues and expand the intellectual toolkit of safety professional and researchers, we adopted a pragmatic approach in this work and examined several case studies of software contributions to military aviation and RPAS accidents. This choice was motivated by several considerations, some are military (Air Force) centric, and some are broadly relevant to the safety community.

First, military aviation mishaps offer significantly learning opportunities that extend far beyond their particular context, especially when software is involved. The concept of **technology dual-use** can be expanded to reflect the **significant safety spillovers that military aviation mishaps can offer to commercial aviation**. As noted in the Introduction, the operational environment and flight parameters of military aviation offer distinctive flight conditions, which can act as triggers of (deeply buried) software pathogens or lurking software defects.

Second, the military is an *early adopter*. In particular, the military has considerable operational experience of RPAS (e.g., about 350,000 hours of flight in FY 2012 for the Air Force) [2]. Given the recent push to integrate them with the U.S. National Airspace, this military experience is particularly rich for probing and analysis, and making better-informed decisions. The two RPAS mishaps here examined were included as the proverbial tip of the iceberg, and they highlighted a number of safety issues in the design, operation, and maintenance of drones. We recommended these systems be subject to much more stringent and informed safety testing, certification, and regulation than their military counterparts before they are allowed to share the same resource, the National Airspace, with civil airliners.

Third, in the past five years (2008–2012), the Air Force reported mishaps in aviation resulting in $5 billion in lost equipment and injuries (excluding ground-related mishaps) and over 9,000 lost workdays. Within this time frame, the Air Force lost 787 lives to all safety mishaps [2]. These numbers may be sidelined by bigger budgets or more consequential items, or they may elicit a resigned attitude, but they remain significant penalties, and they should not be accepted as the *cost of doing business*, especially as "most of these mishaps are preventable" [2]. Any contribution for improvements in this area can help save lives, preserve military capabilities, and reduce costs associated with mishaps.

One contribution of the present work lies in the identification of a number of software-centric recommendations from the analysis of the case studies, some specific to the system studied, and some broadly relevant to help guide software development, coding and testing:

1. Specific recommendations were provided within each case study, for example on proper requirement flow-down to every piece of the flight software, as in the case of the QRF-4C and F-22, and in relation to specific systems, for example the braking system of the T-1A, electrical issues with the MQ-1B. The details are in the text and will not be repeated here to avoid further stretching a long article.

2. The case studies here examined illustrate clear software contributions to accidents, either because of compliance with requirements that were unsuited for the operational conditions at hand, or because of missing or flawed requirements. The particular software contributions to the accidents emerged or were **triggered** because of interactions between the software and the sensors, the actuators, or operational conditions, which constituted unconsidered scenarios and for which the requirement were unsuited. The identification of these and other specific mechanisms by which software contributes to accidents can help populate a library of patterns and triggers of software contributions to adverse events, a library which in turn can be used to help guide better software development, better coding, and better testing to avoid or eliminate these particular patterns and triggers.

3. As evidenced in this and in previous works by several authors [8, 10, 11], the traditional notion of "software failure" as non-compliance with requirements and inability of the software to perform its assigned function is too limited to capture the diversity of software's roles in accidents. This semantic shortcoming creates a blind spot in seeking to understand and eliminate this role. We believe the traditional notion of "software failure" should be obsoleted. The idea of "software contribution to adverse events" offers more interpretive possibilities, as is a better way to frame the issues that ought to be analyzed and tackled. This shift in perspective has also **important implications for software development and testing**, for example on the generation of test cases related to the notion of software failure-triggering fault interaction, or FTFI [39, 40]. This constitutes an important and rich area for future research, and we hope it attracts the interest of software researchers and developers.

Another contribution of this work is in the challenge we provided to the AIB investigations, both in process and in content/findings. The close examination of the AIB reports led us to the following procedural and organizational recommendations:

1. Accident investigations should examine, and the **SIB/AIB reports should include a section on software contributions to the accident, in the same way that the current report template includes a section on**

maintenance and human factors for example. Software contributions to military mishaps seem to be disregarded in investigations (**an institutionalized software blind spot**), and the absence of this feedback loop constitutes a missed learning opportunity for contributing more efficiently to accident prevention. **We strongly recommend timely action on this issue, and that each Board president be trained and urged to include an examination of software role's, among other factors, in the mishap**. We also recommend that software experts be appointed as part of the accident investigation technical advisors. This expertise can be sought in-house within the service or contracted through external support (from Federally Funded Research And Development Centers for example or academia). At a more senior level, the influential DoD Instruction 6055.07 [41] requires that all heads of DoD components, that is, organizational entities within the DoD, "shall collect, maintain, analyze, and report human errors, human factors, and human performance data identified in safety investigations." We strongly urge the addition of a similar statement regarding software contributions to mishaps. **The learning opportunities and (long-term) benefits in terms of cost and lives saved, and military capabilities preserved can be significant**.

2. Finally we recommend a careful examination of the benefits and costs (resources expanded, e.g., 6 board members and 8 potential technical advisors for a period of 60 to 90 days) of having an AIB report in the first place. This cost-benefit analysis should be complemented with the development and assessment of alternatives to the AIB, which can fulfill the dual role of shielding the safety privilege information in the SIB report on the one hand, and satisfying the public right to know on the other hand, especially when mishaps include civilian casualties or damages; there are other purposes to the AIB reports that should be considered as well, such the gathering and preservation of evidence for claims, litigation, or disciplinary actions [42]. The reason we make this recommendation is the following: the AIB report consists of Part I of the SIB report, the factual part, to which is appended the Statement of Opinion of the Board President. **Although the quality of the Statement of Opinion varied, we found some of the ones consulted generally lacking, sometimes internally inconsistent, and occasionally in contradiction with the factual information provided**. For example, initiating events of accident sequences or intermediate causes were presented as main causes of the mishaps, upstream and downstream causal factors, beyond the ones identified were disregarded or overseen, and **more generally the Statement of Opinions were not *safety-operationalizable***. This may not be their intent, but we doubt in their current form they would satisfy inquiring minds within the public and the media. This assessment parallels a recent finding by DoD Inspector General who investigated an AIB report into an F-22 mishap [43]. The investigation noted the lack of detailed analysis of various factors involved in this accident, inaccurate references to the facts, and a collection of three incompatible causal factors presented in the statement of opinion as the

cause of the mishap. **The military should ask the following question: can an alternative to the AIB be found, which fulfills its purpose and is done in a more cost-effective manner?** (e.g., sanitization of the SIB report, and a drafting of a statement of opinion based on the factual information and sanitized report, with the support of an external safety advisory board). The need for a public report into military mishaps has a long history, starting in the 1940's when they were known as "collateral investigations" [42]. These reports have evolved over the years in response to major accidents, public outcry, and legislative pressure. While this history has to be contented with, it may be time to consider the next evolutionary step for these reports in response to budgetary pressure and a more safety-informed public.

REFERENCES

1. Johnson, C.W. A Handbook of Military Incidents and Accidents, Glasgow University Press, Scotland, UK, 2011.

2. U.S. Air Force. Air Force Safety Center Annual Report, FY2012. Kirtland AFB, NM: Headquarters Air Force Safety Center; 2013.

3. Johnson, C.W. Military Risk Assessment in Counter Insurgency Operations: A Case Study in the Retrieval of a UAV Nr Sangin, Helmand Province, Afghanistan, 11th June 2006. Third IET Systems Safety Conference, Birmingham, UK, 2008.

4. Johnson, C.W. Act in Haste, Repent at Leisure: An Overview of Operational Incidents Involving UAVs in Afghanistan (2003-2005). Third IET Systems Safety Conference, Birmingham, UK, 2008.

5. Tvaryanas AP. USAF UAV mishap epidemiology, 1997-2003. Presented at: Human Factors of Uninhabited Aerial Vehicles First Annual Workshop. 2004 May 24-25; Phoenix, AZ.

6. Williams KW. A summary of unmanned aircraft accident/incident data: Human factors implications. Oklahoma City (OK): Federal Aviation Administration Civil Aerospace Medical Inst. (US); 2004 Dec. 14 p. Report No.: DOT/FAA/AM-04/24.

7. Johnson, C.W. Competency Management Systems to Support Accident and Incident Investigators. Proceedings of the 29th International Systems Safety Society, Las Vegas, USA 2011.

8. Favarò FM, Jackson DW, Saleh JH, Mavris DN. Software contributions to aircraft adverse events: Case studies and analyses of recurrent accident patterns and failure mechanisms. Reliability Engineering & System Safety 2013 May;113:131-142.

9. IEEE. IEEE Std. 610.12-1990, IEEE Standard of Software Engineering Terminology. New York, NY. The Institute of Electrical and Electronics Engineers; 1990.

10. Leveson NG. Safeware - System Safety and Computers. New York, NY. Addison-Wesley; 1995.

11. Knight, J. C. "Safety critical systems: challenges and directions." In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*, pp. 547-550. IEEE, 2002.

12. U.S. Air Force. Fact Sheet: Air Force safety and accident board investigations [Internet]. U.S. Air Force; 2013 Jan 11 [cited 2013 Jun 26]. Available from: http://www.acc.af.mil/library/factsheets/factsheet.asp?fsID=2356.

13. U.S. Air Force Instruction (AFI) 91-204. Safety Investigations and Reports, Corrective Actions applied on 10th April 2014. http://static.e-publishing.af.mil/production/1/af_se/publication/afi91-204/afi91-204.pdf

14. National Transport Safety Board (NTSB), Title 49 Transportation Code of Federal Regulations, Part 830.2, as amended at 75 FR 51955, 2010. http://ecfr.gpoaccess.gov/

15. Department of the Army, U.S. Government. Accident reporting and records. Washington, DC: Headquarters, Department of the Army; 1994 Nov 1. 46 p. Regulation No. 385-40.

16. Hendrick K, Benner, L. Investigating Accidents with STEP. New York, NY: Marcel Dekker; 1987.

17. Sklet, S. Comparison of some selected methods for accident investigation. Journal of Hazardous Materials, 111(1-3): 29-37, 2004.

18. Herrera, IA, and Woltjer, R. Comparing a multi-linear (STEP) and systemic (FRAM) method for accident analysis. Reliability Engineering & System Safety, *95*(12), 1269-1275, 2010.

19. Salmon, PM., Cornelissen, M and Trotter, MJ. Systems-based accident analysis methods: A comparison of Accimap, HFACS, and STAMP. Safety Science 50.4 (2012): 1158-1170.

20. Rasmussen, J. Risk management in a dynamic society: A modeling problem. Safety Science, 27(2-3): 183-213, 1997.

21. Leveson, N. A Systems Model of Accidents, 20th International Conference of the System Safety Society, August 5–9, 2002. Denver, Co.

22. Leveson, N. A new accident model for engineering safer systems. Safety Science, *42*(4), 237-270, 2004.

23. Turner, BA. Man-made disasters. London, England New York: Wykeham Publications (London) ; Crane, Russak, 1978.

24. Reason, JT. Managing the risks of organizational accidents. Aldershot, Hants, England ; Brookfield, Vt., USA: Ashgate, 1997.

25. Perrow, C. Normal accidents: living with high-risk technologies. New York: Basic Books, 1984.

26. Roberts, KH. Managing High-Reliability Organizations. California Management Review, 32(4): 101-113, 1990.

27. Roberts, KH. Some Characteristics of One Type of High Reliability Organization. Organization Science, 1(2): 160-176, 1990.

28. Rochlin, GI, La Porte, TR, and Roberts, KH. The self-designing high reliability organization (1987). Reprinted in Naval War College Review, 51(3): 17, 1998.

29. Saleh, JH, Marais, KB, Bakolas, E, and Cowlagi, RV. Highlights from the literature on accident causation and system safety: Review of major ideas, recent contributions, and challenges. Reliability Engineering and System Safety, Vol 95, Issue 11, 2010, pp. 1105–1116

30. Cowlagi, RV, and Saleh, JH. Coordinability and consistency in accident causation and prevention: formal system theoretic concepts for safety in multilevel systems. Risk Analysis, 2012.

31. U.S. Air Force. Aircraft Accident Investigation Board Report: T1-A, S/N 91-0092, 16 August 2003. Keesler AFB, MS: U.S. Air Force; 6 October 2003.

32. U.S. Air Force. Aircraft Accident Investigation Board Report: QRF-4C, T/N 65-0845, 13 May 2011. Tyndall AFB, FL: U.S. Air Force; 4 August 2011.

33. U.S. Air Force. Aircraft Accident Investigation Board Executive Summary: F/A-22, 91-4003, 28 September 2004. Edwards AFB, CA: U.S. Air Force; 2004.

34. Saleh JH, Saltmarsh E, Favarò FM, Brevault L. Accident precursors, near misses, and warning signs: critical review and formal definitions within the framework of Discrete Event Systems. Reliability Eng & Syst Saf. 2013 Jun;114:148-154.

35. U.S. Air Force. Aircraft Accident Investigation Board Report: MQ-1B, T/N 06-3173, 7 May 2011. Creech AFB, NV: U.S. Air Force; 28 July 2011.

36. Saleh, J.H., Marais, K. B., Favaro "System safety principles: A multidisciplinary engineering perspective". Journal of Loss

Prevention in the Process Industry, vol. 29, 2014, pp. 283–294.

37. Brevault L, Favarò FM, Saleh JH. On primitives of causality: from the semantics of agonist and antagonist to models of accident causation and system safety. Proceedings of the ESREL 2013 Conference. Forthcoming 2013.

38. Johnson, C.W. Insights from the Nogales Predator Crash for the Integration of UAVs into the National Airspace System under FAA Interim Operational Guidance 08-01. Proceedings of the 27th International Conference on Systems Safety, Huntsville, Alabama, USA 2009.

39. Kuhn DR, Wallace DR, Gallo AM. Software fault interactions and implications for software testing. IEEE Trans on Software Eng. 2004 Jun;30(6):418-421.

40. Horgan J, Mathur A. Software testing and reliability. In: The Handbook of Software Reliability Engineering. Lyu MR, editor. Los Alamitos, California: IEEE Computer Society Press; c1996. 531-565 p.

41. Department of Defense, U.S. Government. Accident Investigation, Reporting, And Record Keeping. 2000 Oct 3. Instruction No. 6055.07.

42. Glenn Parr E. A Primer for the DoD Task Force on Prevention of Suicide by Members of the Armed Forces. Air Force Accident Investigation Boards. Lecture conducted at: Air Force Legal Operations Agency; 2009; Arlington, Virginia.

43. Inspector General, Department of Defense, U.S. Government. Assessment of the USAF Aircraft Accident Investigation Board (AIB) Report on the F-22A Mishap of November 16,2010; 2013 Feb 6. 80 p. Report No. DODIG-2013-041.