Reliability Engineering and System Safety 137 (2015) 101–111

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Software in military aviation and drone 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, Atlanta, GA, USA
[b] School of Computing Science, University of Glasgow, Scotland, UK

ARTICLE INFO

Article history:
Received 14 June 2014
Received in revised form
8 January 2015
Accepted 10 January 2015
Available online 19 January 2015

Keywords:
Military aviation
Mishap
Accident investigation
Software
Remotely Piloted Air Systems (RPAS)

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.

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, 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

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

http://dx.doi.org/10.1016/j.ress.2015.01.006
0951-8320/

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 9000 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]