January 2014

Analysis of Software Contributions to Military Aviation and Drone Mishaps

Veronica Foreman
Georgia Institute of Technology

Francesca Favaro
Georgia Institute of Technology

Joseph Saleh
Georgia Institute of Technology

Analysis of Software Contributions to Military Aviation and Drone Mishaps

Veronica L. Foreman, Georgia Institute of Technology
Francesca M. Favarò, Georgia Institute of Technology
Joseph H. Saleh, Georgia Institute of Technology

Key Words: military aviation; mishap; accidents investigation; software; drones.

SUMMARY & CONCLUSIONS

Software is assuming an increasing role in the aerospace industry, and by the same token it is also playing an increasing role in many recent incidents and accidents of both military and commercial vehicles. To better understand this role, we examine two case studies from the accident database of the Air Force Accident Investigation Board (AIB). We previously illustrated the limitations of the notion of "software failure" and developed, in its stead, the notion of software contribution to adverse events. We show here how specific operational scenarios, generally unconsidered during the software development and testing, trigger those contributions. We provide an analysis of the recurrent patterns of those mechanisms and preliminary recommendations for software development and testing. We also suggest ways to consolidate AIB reports' findings and to be more mindful of the chain of causality in the accident sequences.

1 INTRODUCTION

This work is at the intersection of three considerations and research areas: software in safety-critical systems; aviation accidents; and military aircraft and drone mishaps. Each area contributes to the motivation for this work, and collectively they provide broad relevance and applicability of the analyses and findings here reported.

Software has become pervasive and central to the operation of many systems of various scales and complexities, and by the same token, it is playing an increasing role, albeit not well understood, in system accidents. This is especially true in the aerospace industry 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), software is contributing in different ways to adverse events in this industry. In the past five years (2008 – 2012), the U.S. Air Force reported mishaps in aviation (both crewed and unmanned) resulting in $5 billion in lost equipment and injuries and over 9,000 lost workdays [1]. Regarding the unmanned category, 96 drone mishaps (with damages exceeding $ 2 millions each) have been reported in the last ten years [1]. While the media has been concerned with privacy issues of flying drones within the U.S. for surveillance or other purposes, safety concerns are still missing from the public debate and media coverage for this highly-software relying systems. Recent aircraft accidents involving software are a worrying indication of limitations in the current understanding of software contributions to accidents and of the 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 not emphasized enough or carefully examined in accident investigation reports, to the extent that their importance warrants. Examples of such overlooks can be found in [2] and will be highlighted in the presentation of the case studies.

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). Two Class A mishaps – the most grievous category of accidents based on the Air Force classification from Class A to Class D [1] – are here analyzed in detail. We hope the lesson learned from them can advance the thinking about these issues, expanding the intellectual toolkit of safety professionals and researchers, and inviting further scrutiny of software contributions to aircraft accidents by both military and civil communities. We have argued elsewhere [2] about the limitations of the notion of "software failure" currently defined in various professional standards (e.g., [3]), as it does not capture the diversity of software's roles in accidents—some of which do not involve a "failure". Leveson also challenged the notion of software failure as non-compliance with requirements [4]. We argued for a shift in perspective, that software failure is not necessarily an ill-defined concept, but that the notion itself is too limited to capture the diversity of ways in which software can contribute to accidents. We developed the notion of software contribution to adverse events as a better way to frame these issues, and a more important problem to analyze and tackle. We revise and extend this concept in this work, bringing the focus to military aviation and drones accidents. 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 – see [5] for a definition of accident pathogens). As such, military aviation accidents offer a unique learning opportunity to address software defects, which are unlikely to be triggered under nominal operating conditions, across the entire military fleet and *with important spillovers to commercial aviation*.

Before delving into the case studies presentation a caveat is in order: due to the constraining six-page limit for this conference, many implications of the presented findings (that may occur to the reader as insightful contributions) are left as future venue for a broader spectrum work on the current limitations in our understanding of the role of software in accidents and incidents. The remainder of this work is organized as follows. In Section 2 we introduce the case studies, and we analyze in detail each accident sequence and the software contributions to the mishap. Section 3 concludes this works with a brief synthesis.

2 SELECTED CASE STUDIES

The case studies selected for this work were chosen to identify and highlight recurrent patterns of software contributions to military aviation accidents. These constitute a subset of such patterns and, while important, they should not be considered exhaustive. The information for the two following Class A mishaps derives from the respective AIB reports. For each case study, background information relevant to the mishap is first presented, and then the accident sequence is reconstructed and discussed. The software contributions to the accident are then identified and analyzed. Additional concerns that do not specifically relate to software but highlight important safety issues are included at the end of each case.

A clarification is in order before proceeding. Following a safety accident, 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"; the second is known as the "accident investigation" and it is "conducted [by the AIB] to provide a report for public release" [6]. The SIB report consists of two parts: the first part is factual and data-centric, while the second is analytic and 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. The AIB Board President then appends to the report his/her *Statement of Opinion* about what caused the mishaps using a "clear and convincing evidence" standard [6]. The reconstructions of the following mishaps are based only on publically releasable information—the AIB reports. We acknowledge that additional privileged information may be available but withheld from the public in the SIB reports, and as such, our analyses may be limited in some ways and to varying extent. Critical thinking however was exercised throughout our examination of the AIB reports, and while some context information and elements in each accident sequence may be missing, we believe our conclusions regarding the software contributions to the mishap are robust to such omissions.

2.1 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 on the evening of August 16, 2003, 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, [7].

Several contributing factors were involved in this accident and we briefly mention them before examining the software contribution to this accident: 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 the chain of causality in system accidents involve multiple strands, and while training and human factors were clearly involved in the T-1A accident, the software strand was rather disregarded in the investigation, and yet it contributed significantly to the accident.

2.1.1 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 aircraft software systems, 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 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 four people as well as substantial damage to the aircraft and airport property (see [2] for details).

2.1.2 The T-1A Accident Sequence

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 link in the chain of causality between this off-nominal landing condition and the accident.

The aircraft touched down about 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" [7]. 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.

2.1.3 Software Contributions to the Accident

The 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 the aircraft was on the runway 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. Sine 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; 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" [7]. This information is a bit suspicious and contradictory with other statements in the report confirming 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.

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. 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.

2.2 MQ-1B Predator Accident, May 7, 2011

The second case study involves a MQ-1B Predator Drone that crashed into the Gulf of Aden on May 7, 2011. The drone 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 drone impacted the water about a half mile off the coast of Djibouti. The loss of the drone was estimated at $4.4 million [8].

Although software played a secondary role in this accident – electrical problems and operational factors were much more prominent causal factors – we selected this case study for the distinctiveness of the software contribution to the accident. Unlike the previous case, here software became an issue far downstream the accident sequence, and 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.

2.2.1 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 drone via satellite link from a remote location. 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, [8]. After the mission is completed, the drone is handed back to the LRE for approach and landing. This split operation will have some implications for this accident.

The drone has a primary flight control system and a backup system. In addition, separate wing modules control the ailerons; 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. It is interesting to note that many of the Class A mishaps investigated since 2000 are attributed to various 'catastrophic electrical failures' and that the RWCM was a causal factor in the loss of another MQ-1 in Afghanistan on May 8, 2009 (whose loss was valued at $4.6 million [9]). Whether this is coincidental or indicative of a systemic problem, this electrical failure pattern, including the performance of the RWCM, deserves careful attention across the entire fleet of MQ-1B in service.

2.2.2 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. We surmise nonetheless that this failure mode may 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. How the fail-safe principle can be implemented for the ailerons (and other control surfaces) would require some creativity and technical ingenuity [10], but it is not beyond skills of aircraft manufacturers. 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 drone 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 drone 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" [8]. 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. We encourage a reevaluation of this software constraint, and if needed, an examination of a different implementation (for instance, more gradual and possibly with force feedback).

Due to the inability to symmetrically match the right aileron down position, the drone continued its left roll until it impacted the water.

2.2.3 Software Contributions 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 case.

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 [11], probably of lesser familiarity and significance than the traditional causal factors in accidents. 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 drone, 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. Several maintenance issues with this drone 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* [8]. Within this same time period, the drone experienced problems with all exhaust temperature sensors on two occasions. Pressure sensors were also found deficient and *replaced on three occasions*. The Secondary Control Module (SCM) was *replaced six times*, and the Primary Control Module was removed twice within the same time period (once for a software upgrade).

For an independent observer, these repeated occurrences might reflect a deficient safety culture and lax attitude toward safety in operating and maintaining the drones. This however need not be the case, and a more useful conclusion can be reached by further reflecting on these occurrences. 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" [8]. Fault analysis guidance as well as maintenance policies and guidelines were clearly deficient in this case, and if these apply to other military systems, we strongly recommend that they be revised and upgraded, to account for repeated failures in the same item for example, and trigger a different response than the band-aid solution of replacing faulty elements without proper analysis and understanding.

Finally, we note that a non-military version of this drone is already in use in the continental U.S., by Border Patrol for example. In light of the discussion in this subsection, we hope these non-military drones are subject to much more stringent safety testing, certification, and regulations than their military counterparts.

3 CONCLUSIONS

Software has wiggled its way into our daily lives to an extent we may not have taken the full measure of yet. Modern life in the developed and developing world is inconceivable without software. But with this increasing reliance on software come new and increasing vulnerabilities. Software is playing a growing role in adverse events, and our present understanding of its 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.

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 two case studies of software contributions to military aviation and drone 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 rarely get attention from the media, or the academic and research community, yet they offer significantly rich learning opportunities that extend far beyond their particular context, especially when software is involved. 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. As such, *military aviation can be thought of as an accelerated, and high stress, testing environment and regime for all aviation software.*

Second, the military is an *early adopter* of drones and has extensive operational experience with these systems. Given the recent push to integrate drones in the U.S. National Airspace, this military experience is particularly rich for probing and analysis, and making better-informed decisions. The drone mishaps here examined was included as the proverbial tip of the iceberg, and it 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 commercial airliners. We hope our work helps inform a richer public debate on this subject beyond, or in addition to, the privacy issues that drones raise, and that this subject attracts more scrutiny by academics and safety professionals.

Third, in the past five years 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 [1]. Any contribution for improvements in this area can help save lives, preserve military capabilities, and reduce costs associated with mishaps. The case studies here examined led to a number of recommendations, some software-centric and broadly relevant, and some specific to the systems studied, and others were procedural or organizational in nature and specific to the Air Force and the AIB reports. We hope this work encourages and invites more academic interest and research in this area. Doing so would bring some public scrutiny and provide thought-partnership to the various military safety centers in their efforts to improve safety, and help better capture and disseminate lessons learned.

REFERENCES

1. US Air Force (2013a): "Air Force Safety Center Annual Report". headquarters Air Force Safety Center, Fiscal Year 2012, Kirtland Air Force Base, NM.

2. Favarò, F. M., Jackson, D. W., Saleh, J. H., and Mavris, D. N. (2013). "Software contributions to aircraft adverse events: Case studies and analyses of recurrent accident patterns and failure mechanisms". *Reliability Engineering & System Safety*, *Vol. 113*, pp. 131-142.

3. ISO/IEC/IEEE (2010): System and Software Engineering Vocabulary 24765. Institute of Electrical and Electronics Engineering, NY.

4. Leveson, N. G. (1995). "Safeware - System Safety and Computers". Addison-Wesley, New York, NY.

5. Saleh, J. H., Saltmarsh, E. A., Favarò, F. M., Brevault, L. (2013). "Accident precursors, near misses, and warning signs: critical review and formal definitions within the framework of Discrete Event Systems". Reliability Engineering & System Safety.

6. US Air Force (2013b): "Air Force safety and accident board investigations". U.S. Air Force fact sheet. Available at: http://www.acc.af.mil/library/factsheets/factsheet.asp?fsID=2356

7. U.S. Air Force (2003). Aircraft Accident Investigation Board Report: T1-A, S/N 91-0092, 16 August 2003. Keesler Air Force Base, Mississippi. 6 October 2003.

8. US Air Force (2011). Aircraft Accident Investigation Board Report: MQ-1B, T/N 06-3173, 7 May 2011. 432D Wing, Creech Air Force Base, Nevada. 28 July 2011.

9. US Air Force (2009). Aircraft Accident Investigation Board Report: MQ-1B, T/N 05-3140, 8 May 2009. Ghazni, Afghanistan.

10. Saleh, J. H., Marais, K., Favarò, F. M. (2013) "A Synthesis of System Safety Principles: Multidisciplinary Engineering Perspective". Submitted to Reliability Engineering and System Safety, May 2013.

11. Brevault, L., Favarò, F. M., Saleh, J. H. (2013). "On primitives of causality: from the semantics of agonist and antagonist to models of accident causation and system safety". Presented at the ESREL 2013 conference, Amsterdam, September 29th – October 2nd, 2013.

BIOGRAPHIES

Veronica L. Foreman
Georgia Institute of Technology
270 Ferst Drive, MK 211
Atlanta, GA, 30332, US

e-mail: vforeman3@gatech.edu

Ms. Foreman is an undergraduate student at Georgia Institute of Technology. She is majoring in Aerospace Engineering, with a minor in Economics. She is a Stamps Leadership Scholar, a Dean's List Scholar, and an undergraduate researcher in Georgia Tech's Space Systems Design Lab. In the past, she served as an undergraduate researcher for Georgia Tech Aerospace System Design Lab.

Francesca M. Favarò
Georgia Institute of Technology
270 Ferst Drive, MK 211
Atlanta, GA, 30332, US

e-mail: ffavaro3@gatech.edu

Mrs. Favarò is a PhD student in Aerospace Engineering at Georgia Institute of Technology. She earned a BE and ME in space engineering at Politecnico di Milano, Italy. Mrs. Favarò authored and co-authored multiple journal papers on accident causation and system safety and is the recipient of the Amelia Earhart Fellowship for 2013.

Joseph H. Saleh
Georgia Institute of Technology
270 Ferst Drive, MK 321-2
Atlanta, GA, 30332, US

e-mail: jsaleh@gatech.edu

Dr. Saleh is an Associate Professor in the School of Aerospace Engineering at the Georgia Institute of Technology. He received his undergraduate degree from Supaero in Toulouse, France, his Masters degree from Harvard University, and his PhD from MIT. He serves on the Editorial Board of *Reliability Engineering and System Safety*, and is an associate Editor of the *IEEE Transactions on Aerospace and Electronic Systems*. Dr. Saleh is an AIAA Associate Fellow and a Senior Member of the IEEE. He received several awards for his research, teaching and mentoring. His research platform covers three broad topics: Reliability and Multi-State Failure Analysis; Programmatic Engineering; and Accident Causation and System Safety.