2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)

2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET) | 979-8-3503-2259-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICSE-SEET58685.2023.00013

Speak, Memory! Analyzing Historical Accidents to Sensitize Software Testing Novices

Natalia Silvis-Cividjian
Computer Science Department
Vrije Universiteit Amsterdam
The Netherlands
Email: n.silvis-cividjian@vu.nl

Fritz Hager, MS
ABR Certified Radiological Physicist, Retired

***Abstract*—Accidents tend to be traumatic events that one would rather forget than remember. Software testing novices at the Vrije Universiteit in Amsterdam, on the contrary, rewind the past and learn how to safeguard the future.**

In this paper we will present FAIL, a rather unconventional assignment that methodically investigate 13 historical software-related accidents, varying from the Ariane-5 rocket explosion to the Knight Capital trading glitch. Innovative is that software testing students use STAMP, a modern systems-theory-based accident causality model and have the possibility to interview a witness of the famous Therac-25 radiation overexposures.

A recent deployment to 96 CS graduates received positive evaluations. We learned that even a lightweight, yet systematic investigation of failures (1) motivates students, by sensitizing them to the consequences of suboptimal testing, and (2) reveals key soft-skills testers need to prevent disasters, such as defensive pessimism and a strong backbone. Other, more subtle benefits of the proposed approach include (3) really-happened, instead of artificial case-studies that increase a teacher's credibility, and (4) extraordinary test scenarios students will always remember.

These results invite software engineering educators to include safety assessment elements in their curricula, and call on witnesses of software-related accidents to break the silence and share memories. Future work includes crafting a repository of heritage artifacts (narratives, videos, witness testimonies and physical replicas) to reproduce historical software-related accidents, and make it available to interested educators. Our hope is that motivated professionals will emerge, better prepared to engineer the safe software-intensive systems we all can rely on.

***Index Terms*—software testing education, assignments, soft skills, accident investigations, safety science, STAMP, witness accounts, history of computing, Therac-25**

I. Introduction

Today, software is everywhere - in laptops and mobile phones, simple gadgets and complex devices, on the ground and in the air, - enabling a quality of life unimaginable a few decades ago. However, the trust in this "ubiquitous friendly giant" plummets every time computer-related accidents make headlines. Testers are the last ones who can restore this trust and prevent disasters by timely detecting potential defects. Therefore, there is no doubt that a thorough testing of software-intensive systems is key for a safe society.

Unfortunately, a dedicated software testing course is not yet a regular guest in every computing curricula. Even where such courses are in place, students are often not motivated and perceive testing as a boring activity, less exciting than design or coding [1], [2], [3], [4]. The danger is that today's poorly motivated students might become the reckless testers of tomorrow, paving the way for future disasters. The challenge for us, educators is *"to prevent these doom scenarios and motivate students to become tireless and responsible testers, set up for the long haul."*

Naturally, there are many possible pedagogical solutions to this challenge. In this paper, we will report on a less-conventional attempt that sensitizes software testing novices by exposing them to historical software-related accidents. The gist is to touch their souls and scare them that if they fail to test software adequately, people will die. We based our approach on the Santayana's warning that "who does not remember the past, is condemned to repeat it", and on existing scientific evidence that emotional engagement and especially fear, work as a good learning motivator [5].

We implemented this idea in FAIL, the first assignment in a dedicated graduate course on Software Testing, offered at the Vrije Universiteit (VU) in Amsterdam. In this rather unusual assignment, students investigate 13 historical software-related accidents. Examples are the Therac-25 radiation overexposures of cancer patients [6], the Ariane-5 rocket explosion [7], the Patriot missile interception failure [8], the Mars Orbiter loss [9], and the Knight trading glitch [10]. An innovative element is that students are not just searching information on Internet and summarizing it in an essay. They apply instead a systematic safety analysis method based on STAMP, an alternative philosophy in which accidents are not caused by a flawed component, but by a flaw in system's design [11].

In 2020, the course became the area of innovation for VU-BugZoo, a Comenius Teaching Fellow project [12]. The aim of VU-BugZoo was to build an interactive platform that makes teaching software testing more exciting, by exposing students to historical and artificial bugs, injected in standalone and embedded code. One of the envisioned products was a 3D scale model of Therac-25 radiation therapy machine, with the purpose to demonstrate one of its famous software-related accidents, known as "Malfunction 54". Let us sketch the context. Radiation therapy is along with chemotherapy and surgery, a very effective method to treat cancer. It uses ionization radiation to destroy the tumor(s) while taking care that surrounding healthy organs at risk stay as much as possible undamaged.

979-8-3503-2259-0/23/$31.00 ©2023 IEEE
DOI 10.1109/ICSE-SEET58685.2023.00013

A radiation therapy process needs complex machines, called medical linear accelerators (linacs) and an interdisciplinary team, consisting of oncologists, operators, engineers, medical physicists, and nurses. High patients throughput and a fast treatment are important because cancer does not wait. Because of its highly technical nature, radiation therapy was one of the first medical specialties to apply computers to routine clinical procedures [13]. Therac-25 was one of the first fully computer-controlled linacs, commercially produced in 1983 by AECL. Since 1983, due to its very advanced technical design, Therac-25 contributed to saving thousands of cancer patients' lives in both US and Canada. Unfortunately, the same Therac-25 entered the history as a killer, instead of a healer, causing at least 6 fatal cases of radiation overexposure. Two of them are known as the "Malfunction-54" cases. "Malfunction 54" was a cryptic diagnostic message that appeared on the console before the machine would pause. The only explanation operators could find in the user manual was an ambiguous "dose error". So what they did, tired of the many false alarms messages received every day, was to resume the treatment, and repeat the same sequence until the machine eventually would shut down. There was no way to know that in fact all this time, the system was in a highly-dangerous, degraded condition that overdosed the patients.

Soon after we started the project, we discovered that the few articles on this topic, such as [6] and [14] although extensive, were not sufficient to reproduce the linac with high-fidelity. Therefore, we started a tedious quest for the missing information. Luckily, thanks to our international network, we reached Fritz Hager, who back in 1986-1987 witnessed two of these "Malfunction-54" fatal accidents. To our surprise, this was the same Fritz Hager, medical physicist at the East Texas Cancer Center (ETCC) in Tyler, Texas, praised in [6] for his diligence during the investigations. To quote Leveson, "without [his] efforts, the understanding of the software problems might have been delayed even further." Fritz not only helped us to reproduce in detail the user interface screens of Therac-25 and the state of the machine during the accidents, but he also kindly accepted to participate in our course and meet out students. Although the physical replica of the Therac-25 linac is still under construction, our fortunate encounter successfully materialized in a new and exciting component in the FAIL assignment, consisting of an interview with an accident witness. In this extended formula, FAIL was deployed twice, in April 2021 and in April 2022. We evaluated the intervention in May 2022, by means of two anonymous surveys conducted among the 96 students enrolled in the Software Testing course. The exit survey contained the following research questions.

RQ1. Is a software testing course the right place for an accidents investigation exercise? We asked this question because in FAIL, students do not practice with test generation or execution techniques, as expected from a conventional software testing course; instead, they practice with techniques and skills that actually belong to a safety science course. Currently, this link between software testing and safety science is not exploited in regular CS or SE curricula.

RQ2. What is the learning impact of FAIL in a software testing course? We asked this question because investigating accidents can unlock strong, negative emotions, such as sadness, anger or fear. Will the learning impact prevail ? Or will opening the "Pandora box" result in a traumatic undermining of learning and increase of the dropout rate?

RQ3. What is the added value of meeting an accident witness? We asked this question because facilitating witnesses sessions require organisational and emotional efforts from both sides. Naturally, we wanted to find out whether these efforts have been worth it.

The positive results we received encouraged us to share here our experiences. The remainder of the paper is organized as follows. Section II is a plea in the favour of including accidents investigations in software testing curricula. Section III outlines an original methodology that helps software testing students to analyze and learn from historical accidents. The methodology uses STAMP, a modern systems-theory-based accident causality model. In Section IV, we describe and justify the design of the FAIL assignment. In Section V we discuss the results of a FAIL evaluation in May 2022 and we share our lessons learned. In Section VI we relate our work to similar contributions. We draw conclusions in Section VII, and wrap up by suggesting ideas for further research.

II. Why should software testers investigate accidents?

A legitimate question one might ask in the first place is, why do we need a FAIL assignment ? Why not just start the course directly with exercises practicing software testing techniques, like any other traditional course does? Let us explain why we believe that a software testing student can benefit from an accident investigation exercise.

The most important and rather obvious benefit is that students will be shocked by the huge impact that inadequate testing can have on society. Next, from accidents stories, students will also realize that a tester is not a robot; to prevent accidents, a tester will need also soft, non-technical skills. Why? For example, to report bugs without blaming the developers, or dare to refuse to launch a product that is not tested enough. Finally, scared by the consequences of sub-optimal testing, students will become more motivated to listen to the rest of the course and acquire the needed technical skills to prevent accidents. This last learning objective can be refined further, based on Figure 1, showing the multiple dimensions of software testing.

First, students will understand the need to test for more than just functionality. Let us first remember that software testing is a process of checking that a product is behaving as specified. The problem is that under "specified" one understands many quality attributes; in most cases this is functionality, meaning that the product does what it is supposed to do, but it can also be safety, defined as the absence of accidents, performance (how fast it is, how many requests it can handle, how well can it work with poor resources), or security. The majority of traditional software testing courses only teach testing for

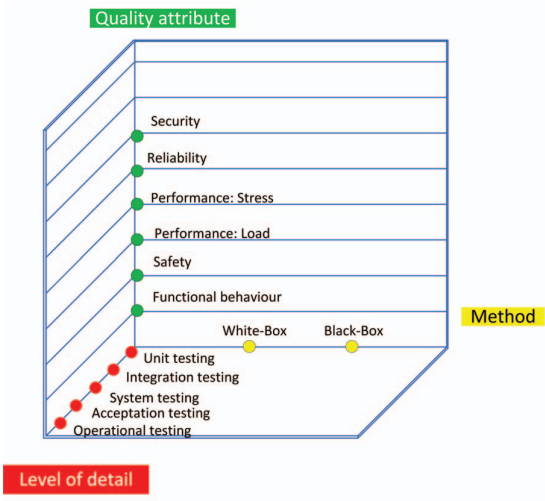

Fig. 1. Different dimensions of software testing. Adapted from [15].

functionality. However, history learns that accidents happen also when the software on board is working as specified and passed all the functional tests. Therefore, we expected that FAIL will show students the need to test for more than just functionality, especially for safety, but also for high load performance, or high usability.

Second, students will understand the need to test the whole system, instead of only its embedded software. Another limitation of the traditional approaches is that they mainly teach unit testing, whereas the root cause of many accidents is a flaw in higher levels of testing, such as integration (like in the case of the Mars Orbiter loss [9]), or system testing (like in the Ariane-5 aborted mission [7]). Therefore, we expected that FAIL will show students that also other test levels, beyond unit testing are important, such as integration, system or operational testing.

Next, we expected students to understand the need to test for abnormal, degraded conditions. In FAIL they will learn that accidents are caused not by a total lack of testing, but by an incomplete testing. The question is why? One of the reasons is in our opinion the overconfidence that makes developers and testers forget an important source of test cases, namely the ”rainy-days”, degraded conditions. Examples are sensors that stop working, or operators that get tired of the warnings that keep popping up on the screen and learn to ignore them. We strongly believe that by analyzing past accidents, students will start worrying about what can go wrong in the software-intensive systems they are so proud of. Therefore, our expectation was that FAIL will teach students to generate interesting, otherwise unimaginable test scenarios that could have prevented accidents. Moreover, they might use these scenarios again in their future career.

Finally, we expected that really-happened, instead of hypothetical accidents will serve as a powerful rationale to introduce certain test techniques in class, which will add to teachers' credibility. For example, after FAIL, we will be able to replace the quite dogmatic *”Today we will be talking about integration testing.”* with the more inspiring *“Today we will be teaching integration testing, to prevent in the future a Mars Climate Orbiter type of loss”*.

III. How should software testers investigate accidents?

In the previous section, we presented the reasons and benefits of investigating accidents in a software testing course. Unfortunately, the idea cannot be implemented directly in a software testing course. The reason is simple - traditional software engineering curricula do not instruct students in how to investigate accidents. This is why in this section, we will outline an original methodology that guides software testers in investigating and learning from accidents. The methodology uses STAMP, a modern, systems-theoretic accident causality model imported from safety science. In this way we built an unusual bridge between safety science and software testing.

In safety science, *accidents* are defined as unplanned events associated with harm, injury or loss. For example, an accident is an event where passengers die in a car crash. Accidents are caused by *hazards*, defined as dangerous, unsafe conditions. An example of a hazard is that the car left its designated area on the road. There is always a *causal scenario* that led to this hazard. For example, it was late in the evening, the driver decided to drive even when he was tired, but unfortunately he fell asleep for a second and lost the control over the vehicle, and eventually hit a tunnel wall. The aim of an accident investigation is to discover this scenario and enforce correction measures to prevent a similar scenario from happening in the future. For example, a correction measure could be to build in the car a non-intrusive driver's attention monitoring system. Note how easy it is to already draw a parallel between safety science and software testing domains. For example, the “hazards” from safety science are in fact the “threats” or “bugs” in software testing, whereas “accidents” are similar to the “failures” in software testing.

Traditionally, an accident investigation rests on the use of event-chain models, that explain accidents as a chain of failure events and human errors that led up to the actual loss event. All classical safety analysis methods use this model. The problem is that all these models originate from 1960's and 1970's, a time when critical systems were mainly electro-mechanical. For that time, the approaches worked well, because people were good in predicting when individual electro-mechanical components, such as pumps, valves or motors will fail. The situation changed when software entered the scene. The problem is that software has a different behaviour than an electro-mechanical component. For example, software will never get tired after being intensively used, and a redundant software module with the same design will never prevent an accident from happening, because the second module will fail in exactly the same way as the first one. Moreover, since the ubiquitous presence of software in everyday' s life, the accidents' nature also changed. For example, in modern accidents, the root cause

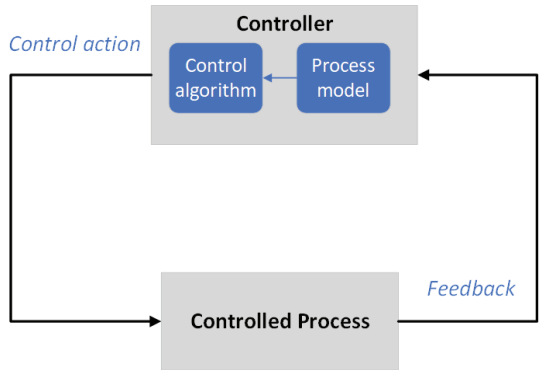

Fig. 2. A generic control structure used in STAMP.

is often not a defect in an individual component, but a sub-optimal interaction between perfectly working components. This makes traditional safety analysis methods less successful in understanding recent accidents, and calls for new methods to assess the safety of modern software-intensive systems, to better understand accidents they are involved into. STAMP, the acronym of System Theoretic Accident Modeling Process, is an answer to this call. STAMP, introduced by Leveson at MIT [11], is a different attempt to approach safety, based on systems- and control theory, instead of the traditional probability-based reliability theory. The novelty of STAMP consists in modeling any process as a system of interrelated hardware, software and human operators that are kept in a state of dynamic equilibrium by feedback loops of information and control. In this holistic view, the basic concept becomes a constraint, instead of an event.

A STAMP-based analysis begins by graphically modelling the process using control structures. In a control structure, controllers interact with controlled processes in terms of *control actions* and *feedback*, as illustrated in Fig. 2. In STAMP philosophy, safety is a systems-emerging property, guarded by constraints, such as "the driver should always keep the car between two white lanes on the road". Accidents are then caused by control flaws, which can be an unsafe control action that violated a constraint or even worse, a lack of a safety constraint in the system's requirements specifications.

Two methodologies based on STAMP philosophy are currently available: Systems Theoretic Process Analysis (STPA) for proactive hazard analyses, and Causal Analysis based on STamp (CAST) for accidents and incidents investigations. Both methods have been gaining popularity lately, being mentioned in an increasing number of published reports [16], [17], [18].

To accomplish FAIL, students make use of an extended version of STAMP-CAST, which we in-house tailored for software testers. We will walk through all the steps of this methodology and take as example the recent Boeing 737 MAX accidents [19]. Because the official investigations of these accidents are not finalized yet, we would like to make clear here that this is just an example on how to apply CAST, and not an attempt to blame any party involved.

CAST-Step 1. Assembly basic information

The first step is the same as in any other incident investigation method. One must collect information available to understand what happened. One need to read newspapers, watch video's, listen to witnesses' stories. In the case of the Boeing 737 MAX accidents, two fatal crashes killing all 346 passengers happened shortly after each other, in 2018 and 2019. Both were linked to the malfunctioning of a newly designed flight stabilizing software module, called MCAS (Maneuvering Characteristics Augmentation System).

CAST-Step 2. Graphically model the system using safety control structures

This step is difficult to grasp for beginners, especially when the controllers are human operators, instead of machines. The control structure in CAST does not need to be complete; it should feature only the controllers involved in the accident. We start with the most obvious suspect, in this case, the MCAS controller. Its controlled process is the aircraft, as shown in Fig. 3. The MCAS reads the status (position) of the aircraft using an angle of attack (AoA) sensor. Next, when needed (a too high AoA measured), the MCAS will take the decision to send a command to push down the nose of the aircraft. The decision is made based on a control logic algorithm, which is in its turn influenced by the process model. Leading in the process of finding more suspect controllers is the question: *Which controller could prevent the accident ?* In this way, we identify other interesting controllers, such as the pilots, who controlled the aircraft and who theoretically could deactivate the MCAS, but also the airlines, who's responsibility was to train the pilots; also the manufacturer, Boeing itself, who by law had to re-test the whole system after the introduction of the MCAS.

By watching the control structure in Fig.3. it is now very easy to anticipate some possible issues, formulate the first questions and identify the human controllers that need to be interviewed. For example, one may ask if there was any feedback from the MCAS to the pilots? Or, were the pilots informed that there was an MCAS on board? If yes, then we can continue asking whether they were instructed on how to switch off the MCAS. And what about the manufacturer, who claimed towards FAA that an external re-testing of the system after the MCAS installation was not needed, because MCAS was not an anti-stall device and the changes were small, compared to the previous versions?

CAST-Step 3. Analyze each component in loss, by specifying its responsibility and contribution to the accident

In this step, we build a table similar with Table I, that summarizes all the controllers involved, their responsibility and the unsafe control action (UCA) that contributed to the loss. In the case of MCAS, the UCA that lead to the accident was that the software (wrongly) pushed the aircraft's nose down when this was NOT needed (the plane was not stalling

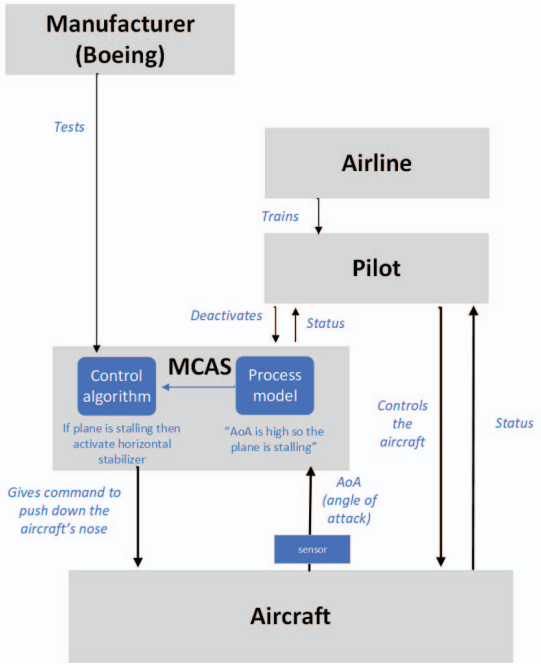

Fig. 3. The control structure for the Boeing 737-MAX accidents.

TABLE I
THE CONTROLLERS INVOLVED IN THE BOEING-737 MAX ACCIDENTS, THEIR RESPONSIBILITIES AND CONTRIBUTIONS TO THE LOSS.

Controller	Responsibility	Contribution to loss
MCAS	Compensate for an excessive angle of attack by pushing down the nose when the airplane is stalling and AoA is high	Pushed the nose down when the airplane was not stalling and AoA was low
Pilot	Controls the plane, maintains the plane in a stable flying position	Pilot did not maintain the plane in a stable position
Airline management	Facilitates training sessions for pilots	Did not train pilots on using and deactivating the MCAS
Manufacturer (Boeing)	Tests new software in degraded conditions	Did not test the MCAS for the case when the AoA sensor malfunctions.

and AoA was low). Other UCAs or hazards that contributed to the loss are: "The pilot did not deactivate the MCAS when the aircraft was heading towards the ground", and "Airline management did not adequately train the pilots on how to deactivate the MCAS".

CAST-Step 4. Identify control structure flaws and their causal scenarios

Now that we know the hazards that contributed to the loss, we can brainstorm on their causal scenarios. The leading question is here "Why a certain controller acted the way it did? What was their process model?". For example, in the case of MCAS, the software did exactly what it was instructed to do by its control logic, namely to push down the nose of the aircraft when it is stalling (a high AoA measured). The problem was the inconsistency between the process model of the MCAS, (which "thought" wrongly that the aircraft was stalling) and the reality (where the aircraft was not stalling). The controller did not know that the AoA sensor is sending wrong data.

Likewise, the pilots did not deactivate the MCAS because probably they were not aware of its existence or, because they did not know how to deactivate it. We have to accept that many questions in an investigation will stay unanswered, like why was the training of pilots so short (2 weeks on a tablet), or why did Boeing decide not to test thoroughly the MCAS software, also for the case of a malfunctioning AoA sensor. Interviews with people involved are very useful in this step.

CAST-Step 5. Create a safety improvement plan

The final result of an accident analysis effort is an improvement plan, containing corrective measures that will prevent similar accidents from happening in the future. These mitigation measures can address software, hardware or procedures. The most evident measure that results from our CAST analysis is to enhance the aircraft design and build in a second, redundant AoA sensor, together with additional software to compare the two readings and trigger an alarm when significant differences are detected. Moreover, the manufacturer should offer this code as a standard feature on all aircraft software stacks. Another recommendation is a better training of pilots, that takes the MCAS into consideration and the possibility to deactivate it when needed. Finally, CAST also suggests to enforce by law that regression testing of any change in software or hardware should be conducted by independent third parties. It is reassuring to read that the same measures were officially formulated in the FAA improvement plan [20] issued after the accidents, later that year. We quote as an example, "an FAA Airworthiness Directive approved design changes for each MAX aircraft, which would prevent MCAS activation unless both AoA sensors register similar readings, eliminate MCAS's ability to repeatedly activate, and allow pilots to override the system if necessary. The FAA began requiring all MAX pilots to undergo MCAS-related training in flight simulators by 2021".

CAST-Step 6. Extracting test scenarios that could have prevented the loss

So far, one followed the CAST methodology as prescribed by the STAMP textbooks [11]. This last step is new and very important for a software tester, because it generates the test scenarios that could have prevented the accident. Of course, one cannot turn back the time, but these scenarios can be used to test an improved, hopefully safer future version of the system. Moreover, we are confident that our students will remember these extraordinary test scenarios and use them later in their career. In this step, for each corrective measure formulated in the previous step, we can generate a test scenario. For example, for the corrective measure to add a second AoA sensor together with the afferent software,

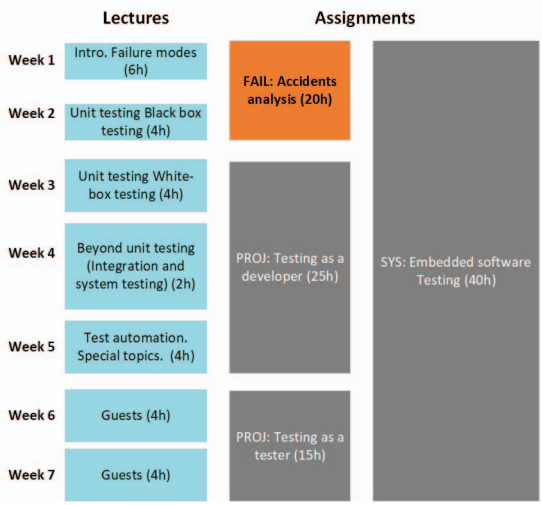

Fig. 4. The Software Testing course organization.

a test scenario could be simulating a malfunctioning AoA sensor and expecting to observe that the new code triggers an adequate ”Sensor malfunction”-kind of alarm. If Boeing had tested this simple scenario, the crashes could have been prevented. This step illustrates the essence of our approach showing how software testing meets safety science.

IV. FAIL: An accident investigation exercise for software testers

A. The design

The design of the FAIL assignment followed the rationale outlined in Sections II and III. Instead of presenting all historical accidents in another boring lecture, we opted for a more exciting solution. We created an assignment that engages students in an accident investigation, where they formulate critical questions and even suffer the same way people involved did. The closest we could get them to such an experience was organizing the interview sessions with the witness of Therac-25 accidents. Our hope, inspired by Elie Wiesel’s teaching vision [21], was that in this way, students will become themselves witnesses who carry on the message to the next generations.

FAIL is embedded in a graduate Software Testing course, offered at the Vrije Universiteit in Amsterdam. The course is mandatory for the Software Engineering track and elective for other tracks, with an yearly enrolment of 50 to 100 students. A detailed description of the course and its innovative elements was published elsewhere [22], [23]. The organisation of the course is illustrated in Fig. 4. The course is seven weeks long and requires a total workload of 156 hours. It features 10 lectures (20h) and 4 guest lectures (8h). 70% of the final grade can be earned with group assignments. The rest of the final grade (30%) can be earned with individual exams.

FAIL is a group assignment, planned for the first two weeks. The workload was estimated to be around 20h. To support FAIL, in the first week, we give two lectures on “Failure modes”, where we introduce different safety assessment methodologies. Starting from the third week, the course follows the ”standard” flow, with lectures that teach test generation techniques for unit, integration and system testing, test adequacy criteria to learn when to stop testing, followed by principles for automated testing generation and execution, and modern, less conventional methods, such as AI for test automation and fuzz testing. The conventional software testing topics closely follow the widely-used textbooks authored by Mathur [15], Pezze and Young [24] and Copeland [25]. Starting from the third week, students work on other, more test-related assignments, such as PROJ (40h), a term project where students practice with testing in both developer and tester views, and SYS (40h), a hands-on practical where students learn how to code and test embedded software.

FAIL starts with an analysis of the Therac-25 accidents, because we believe that these are still the most prominent software-caused disasters in history, from which also testers can learn a lot. Next, each group of four students investigates another accident from Table II, summarizes its findings and presents them in class. Let us first explain how we composed the list of historical, software-related accidents to investigate presented in Table II. First, we decided to analyze dissimilar accidents, to show students that improper testing can lead to a wide spectrum of problems, that vary from an inconvenience, (like in the case of the delayed Denver airport automated luggage system launch), to serious losses, of space mission assets (Ariane-5, Mars), financial assets (Knight), brand image (Intel), and even peoples’ lives (Patriot, London ambulance). Also, we wanted students to discover that in some cases, an accident is caused by a single fault in code (like in the case of the AT&T network failure), but also by a combination of unrelated factors, such as war (Iran Airbus, Patriot), routinised operators (Therac-25), or delays in the schedules (Cali). Finally, we wanted to emphasize that failures are not a deprecated concept that belong to the past - they still happen every day -, so testers should always stay alert. For this reason, we included the Panama accidents that occurred in 2000, 15 years after the famous Therac-25 cases.

We instructed students to investigate accidents using STAMP-CAST, a methodology originating from safety science. The reason was the existing evidence that this method can better explain modern accidents [34], [35], [36], [37]. In CAST, as in any other investigation method, interviews with people involved are required, in order to understand why an accident happened.

B. Task 1: An investigation of the Therac-25 accidents

The first task is to briefly analyze the Therac-25 accidents. In week 1, students read a very extensive article on this topic, by Leveson and Turner [6]. Then they came to class and together listened (in Zoom) to a unique testimony of Fritz Hager, a medical physicist who witnessed a particular kind of Therac-25 accidents, known as MALFUNCTION 54. Students had the opportunity to ask some short, urgent

TABLE II
THE HISTORICAL SOFTWARE-RELATED ACCIDENTS ANALYZED IN THE FAIL ASSIGNMENT.

	Accident	Short description	Flawed testing
1	Therac-25 overexposures (1985-1987) [6]	Multiple fatal over-exposures related to the first fully computer-controlled radiotherapy linear accelerator.	Unit testing, System testing, Operational testing
2	Ariane-5 maiden flight failure (1996) [7]	An European Space Agency rocket exploded 40 seconds after launch. The reason was an overflow exception that was not caught in a software module reused from Ariane-4.	Unit testing, System testing
3	Intel Floating Point Division Bug FPDIV (1994) [26]	Inaccurate scientific calculations caused by an incomplete lookup table operation	System testing
4	Iran Airbus 655 shooting down (1988), [27]	During the Iran-Irak war, a civil plane was shot down because software labelled it as a foe on the radar screen.	Usability testing of GUIs
5	London Ambulance service glitch (1992), [28]	New software-based communication system got jammed, people died before the ambulance could reach them.	Load testing
6	Mars Climate orbiter loss (1999), [9]	NASA Mars climate orbiter lost because two development teams working on the navigation software used different measurement units (Imperial vs Metric)	Integration testing
7	AT&T network crash (1990), [29]	A missing break statement in C code produced an avalanche of denials of service.	Unit testing. Error messages testing.
8	Denver airport (1994), [30]	A promising automated luggage handling system misbehaved during its official opening	Load testing
9	Patriot air defense system failure (1991), [8]	During the Iraq war, an angle of attack calculation error accumulated in time, and became so large that a Patriot air defense system failed to intercept an incoming Scud missile, leading to casualties.	Reliability testing, communication
10	Panama radiotherapy overexposure (2000), [31]	Multiple cancer patients were overexposed during radiotherapy, because there was no clear in which sequence to input the shielding blocks coordinates in a treatment planning software	Requirements verification, Operational testing
11	American Airlines B757 AA965 Cali accident (1995, [32]	An air-crash caused by a wrong destination input in the navigation software.	GUI testing
12	Knight Capital trading loss (2012), [10]	Huge financial loss due to an old software version running on one of the servers.	System testing, Communication
13	Hawaii false missile alarm (2018), [33]	False (live instead of test) ballistic missile alarm was sent by software that produced panic among the population.	Testing for emergency situations

Fig. 5. VU software testing class in dialogue with Fritz Hager.

questions immediately after the first presentation. This first session took 2h. After that, the students had one week to reflect on what they heard, read the Leveson's article again, and prepared as a group two questions for the witness. After one week, students returned to class for an interview with Fritz Hager, who joined the meeting remotely in Zoom (see Fig. 5).

This time, the students had the word. Each student group could ask their two questions. We had 96/4= 24 groups, so 48 questions were collected in advance and sent to the witness. The second session took also 2h. At the end of the second session, we wrapped up and together formulated a few lessons for software engineers and testers. We video-recorded both sessions, so that the students could watch them later. Students wrote a report with their findings, including their answers to following questions.

Why do you think it is difficult to make witnesses talk? Give an argument that could convince a witness to tell their story. Draw conclusions with respect to the Therac-25 accidents investigation in general, and to testing radiotherapy software in particular. For example, what did you understand only from the meeting with the witness that was not mentioned in the Leveson's article? What surprised you the most? Which testing could have prevented the accident? Do you think the situation is much safer now? Search on Internet for some recent accidents in radiotherapy. What should you worry about as a radiotherapy software tester? Why do you think accidents still happen in radiotherapy?

C. Task 2: An investigation of a second software-related accident

For Task 2, each group applied the first steps of a STAMP-CAST method to investigate an accident from Table II, other than Therac-25. All the sources were from Internet because for this accidents, they did not have the possibility to interview witnesses. For CAST-Step 1, students had to describe what happened, by identifying the losses, the system hazards and the safety constraints violated. For CAST-Step 2, they had to draw a high-level safety control structure, identifying the controllers involved and their contribution to the accident. According to STAMP, accidents are caused by flawed controls that could not prevent the accident from happening. Therefore, students

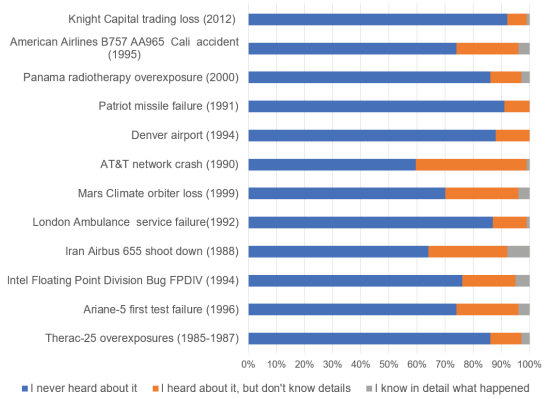

Fig. 6. The results of an entry knowledge survey (74 participants).

had to identify in their control structure the unsafe control actions that caused the loss. For CAST-Step3, students tried to understand and imagine possible causal scenarios that lead to these hazards. We anticipated that they will only partially manage to understand why the accidents happened, as they had limited information. Instead, we asked them to formulate two questions they would have liked to ask to an hypothetical witness. Students had to accept the bitter truth that some questions will always remain unanswered.

Finally, students classified the software-related root-cause using a bug taxonomy, such as for example the widely-used Beizer's taxonomy [38]. Possible answers were: an omission/ commission/mode confusion; a fault in the requirements/design/code; a flaw in unit/integration/system testing; a fault in data transfer; or a fault in communication. Also they had to specify which testing could have prevented the accident. At the end of week 2, students presented in class their findings related to Task 2. In this way, each participants could hear about 11 additional software-related accidents.

V. Evaluation and Lessons Learned

A. The results of evaluations and discussion

We evaluated the FAIL assignment in its new shape in May 2022. In this section, we will discuss the results. Fist, we conducted two student surveys, an entry and an exit one. Both surveys were published anonymously in Canvas, the web-based learning management system used by our university. The entry knowledge survey, aimed to poll to what extent were students familiar with the 13 accidents, was conducted prior to the publication of the assignment. The results of this survey, completed by 74 out of the 96 enrolled students, are shown in Fig. 6. From here we learned that most of the students were not familiar with the accidents. For example, 86% of the students never heard about the Therac-25 disasters before. This made us confident that FAIL will succeed in surprising our students and widen their horizon.

surprise is important for students — they want to be engaged

After the completion of the FAIL assignment, we conducted an exit survey, featuring closed and open questions. Fig. 7 shows a statistics of 49 students' answers to the closed

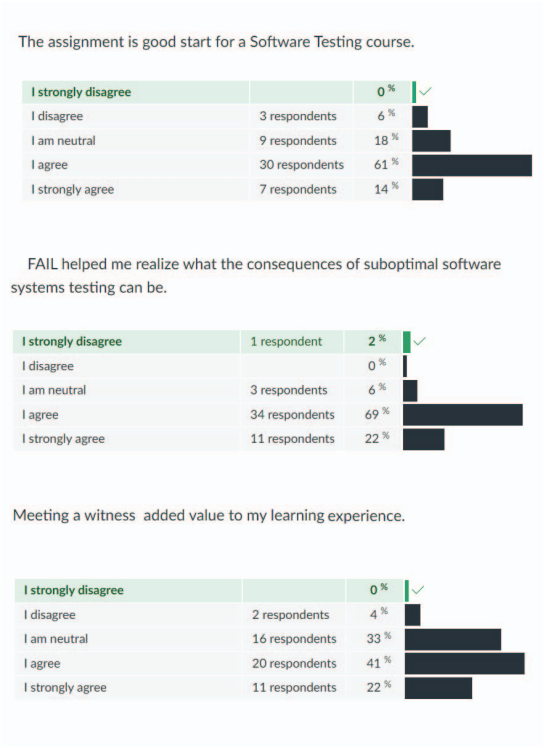

Fig. 7. Exit survey: students' answers to the research questions (49 participants).

questions. Three concerns will be discussed, matching our research questions- the place of FAIL in the curriculum, its learning effect and the added value of the interview with an accident witness.

RQ1. Is a software testing course the right place for an accident investigation exercise?

75% of the participants agreed that the FAIL assignment was a good start for a software testing course, which is a very reassuring result. Only 6% of the respondents disagreed. We understand that not all students are interested in investigating accidents, because not all of them will work later in a safety-critical industry.

RQ2. What is the learning impact of FAIL in a software testing course?

A vast majority of students responded that FAIL helped them to become aware of the consequences of a suboptimal testing (81 % were positive and only 2% disagreed). Moreover, to find out more about the students' perception, we included in the survey also open questions. In Table III we reproduce the most relevant answers, from which one can deduce that FAIL succeeded in motivating students to thoroughly test their software in the future. These results confirm our expectations.

RQ3. Is there any added value in interviewing an accident witness?

We present here the opinions of all the parties involved - students, witness and teachers. When asked whether a meeting with a witness added to their learning experience, 63%

TABLE III
EXIT SURVEY: AN EXCERPT FROM STUDENTS' FEEDBACK (49 PARTICIPANTS)

What was in your opinion the main message of the FAIL assignment?
How people should take responsibility of their software and really learn the value of testing.
Poor design has consequences.
Software development needs a healthy dose of pessimism, especially in systems that could spare or take lives.
That you have a certain responsibility as a software developer to know the consequences of a possible flaw in your design.
Proper testing saves lives.
The entire experience of learning from past mistakes and how little errors can lead to massive problems.
The moral implications of software testing and the fact that we can do a lot of harm with improper testing methodology.
I have come to the conclusion that this task was extremely beneficial to me, and as a result, I am eager to commit a sufficient amount of time to the process of testing.
What do you think about the interview with the Therac-25 witness?
Some aspects of an interview with patients who had been injured by radiation were utterly upsetting for me, and it served as a potent reminder for me to exercise extreme caution with regard to what I design and develop in the course of my work.
I liked that it was research about a real accident that happened and also a person that was in the accident talked to us. So we do know that all the stuff are true.
The witness session made it so much better than just reading things from articles and from the Internet.
I really liked the personal view of the witness and being able to talk to someone with experience. The variety with the different presentations was quite interesting as well.
I especially enjoyed listening to the witness's own experiences and how we got to ask him our own questions. It was an emotional but insightful assignment.
Firstly I liked to interview Fritz. He is a living sample. Secondly, researching a true story was good experience for me. But it made me sad.
I got highly motivated by the attitude of Fritz Hager and his courage to stand in oppose to his supervisors and risk his career to save innocent people. I would like to believe that this interview gave me enough energy and courage to overcome my fears and behave the same if I am in a situation like that in the future.

of the student participants responded positively, 33% stayed neutral, and only 4% disagreed (see Fig. 7). In Table III we included their most relevant open answers, reflecting the strong motivating effect they perceived.

Next, Table IV presents the thoughts of our guest, Fritz Hager, who reflects at the traumatic days back in 1986-1987. Important here are, besides the facts, the emotions that such an event unlocks for the persons involved.

Finally, we evaluated the intervention from the teacher's point of view. The thing that impressed us the most was the electrifying atmosphere during the interviews, which was very different from what we are used to experience in a "normal" lecture. We remember that it was very quiet in the class; you could feel the emotions in the air. Students were really listening and looked concerned, some of them scared, upset, or sad. However, students were far from being apathetic. In contrary, we were surprised that students dared to ask very personal and confronting questions, such as: *"Why did you accept to talk to us, when other people involved choose to stay silent?"* or *"Why didn't you close down the ETCC radiotherapy treatment facility after the first fatal incident?"*. We realized that also for the witness, preparing for the interviews and rewinding those stressful days must have been very intense. This is why we appreciated even more the willingness of Fritz to open a dialogue with our students.

TABLE IV
THE THERAC-25 WITNESS'S REFLECTION.

"I was excited to participate in this class project that investigates the Therac-25 software accidents. These accidents changed the way software safety was viewed by the medical linear accelerator community, manufacturers and users alike.

After the first accident, we exhausted all theories that could result in an acute injury to a patient including allowing the manufacturer access to the machine. This resulted in the first problem, which was the manufacturer not being fully transparent with the accident history of the Therac-25. There were at least three previous incidents that produced injuries to patients and the chief engineer had given a deposition under oath that the Therac-25 had given a very high dose of radiation to a patient. When asked directly whether the Therac-25 was capable of doing this he flatly denied it. He lied. This led to the fourth accident, the second one I was involved with. When I reproduced the Malfunction 54 event and measured the dose our patient received, I knew that we had killed him. This brought about a feeling of despair and I realized that if I was at fault, my career as a medical physicist would be over. When reported to the manufacturer, AECL of Canada, they were able to duplicate the Malfunction 54 incident on their factory Therac-25. Their response was to initiate a corrective action plan of disabling the up arrow key on the keyboard, which disabled editing the patient's treatment. They did not inform the other users that a catastrophic event had occurred, which would likely bring about the death of two patients.

That was the second problem associated with these accidents - the lack of accountability. I informed all the Therac-25 users in the United States and the chief physicist for the Therac-25s being used in centers in Canada. I told them how to reproduce a Malfunction 54 and that I had measured a lethal dose of radiation to our patient. All users reported back to me of their successful reproduction of Malfunction 54. This led to an immediate demand for AECL to determine the root cause of the Malfunction 54. Part of the problem was that AECL had removed most of the hardware safety interlocks. Unfortunately, there were users that continued to use their Therac 25 machines with the inadequate corrective action plan and another incident occurred resulting in yet another fatality. The FDA and Canada's regulating body stepped in and shut down all Therac- 25 linacs until the root causes were determined and corrected.

A third problem that occurred was the restriction of the flow of information. I was told by our organization's president that I was to have no contact with our first patient that was injured. I did not have access to important information that could have led to a swifter solution to the Malfunction 54 problem. I was also told not to give any information or account of the accidents to any colleagues or news media. This resulted in my reputation as a qualified medical physicist to suffer from incomplete information being disseminated by unknowledgeable people.

Events such as the Therac-25 accidents should be freely and widely reported, so that others will not make the same mistakes. It is much better to learn lessons in the class room rather than when peoples' lives are at risk".

B. Lessons learned

During the last 30 minutes of the last interview session, students summarized the lessons they could learn from the

Therac-25 accidents. Below, we enumerate these lessons. We believe that some of them are new, not present in the Leveson's paper [6].

- An accident is often not caused by a single mistake, but by a combination of factors.
- Computers are very useful for radiotherapy. But when things go wrong, they can go very wrong, turning the linac in a lethal weapon.
- In radiotherapy, software is not alone. Therefore it should be tested together with the hardware and its users.
- Users can react in a different way than a developer expects.
- The failure scenario that led to the MALFUNCTION 54 accidents was very difficult to predict.
- A tester needs courage to speak up and tell the management that the product is not ready to be launched in production.
- A user should refuse to use a system that misbehaved once before there is clear what the reason was.
- A software engineer has to anticipate the degraded conditions a system can fall into. A proactive hazard analysis is needed prior to any design, development or testing.
- A software engineer (developer and tester) needs some domain knowledge of the process they control. Vice-versa, domain professionals working with software systems also need to know more about how software is engineered.
- By interviewing a witness of an accident, one learns more than by just reading a report.

As it goes with any initiative, we discovered aspects that need improvement. We noticed for example that students sometimes did not fully understand how to perform STAMP-CAST and had difficulties especially with drawing a correct control structure. Also, they did not understand in detail how the systems involved in accidents work. We agree that radiotherapy machines, airplanes and rockets are indeed very complex systems. In the future, we will improve the clarity of the STAMP-CAST instructions and we will design special handouts with some more technical details for each system involved in the analyzed accidents. The first step in this direction will be to finalize the Therac-25 scale model, as promised in the VU-BugZoo project. Some students considered that the name of the course, Software Testing is misleading because we often cross the boundaries of only software. They are right, we will consider changing the name to a more appropriate name, maybe Software Systems Testing.

To summarize, we believe that the approach described in this paper achieved the goals as outlined in Section II. After completing the FAIL assignment, each student analyzed two accidents in depth, and heard about other 11 failures they were mostly not aware of. Students were emotionally engaged and realized the consequences of suboptimal testing. As a result, they became motivated to learn technical skills needed to thoroughly test software-intensive systems, in order to prevent future accidents. Some necessary soft, non-technical skills were also identified, such as perseverance and a constructive pessimism, a strong backbone and a clear interpersonal communication. Moreover, students became familiar with STAMP, a modern safety assessment method they can use in the future anytime they need to analyze an incident or to conduct a proactive hazard analysis. In both cases, they will be able to extract interesting scenarios for safety testing. Finally, we as teachers now have a pool of examples we can use in our course to introduce different test techniques in an inspiring way.

We hope that these positive results will encourage other software testing educators to include a similar assignment, where students use safety science methods to remember the past and understand how to safeguard the future. We also call on professionals who in the past witnessed unsafe situations, to share their stories with students. We agree that sharing stories is an emotionally-intensive process, but the noble purpose of educating new generations should prevail. In the future, we will craft a novel heritage repository with historical artifacts (narratives, testimonies, videos and physical replicas) to reproduce notorious software-related accidents, and we will make it available to interested educators.

C. Threats to validity

A validity issue could be the fact that students interviewed only one witness. Ideally, one should have witnesses for each accident. Unfortunately, we know from experience that people involved in recent accidents are reluctant to talk and remember. Luckily, witnesses of older accidents are more cooperative and eager to share experiences and educate new generations. Moreover, our witness was a medical physicist and not an IT specialist. We agree that more or maybe different lessons could have been derived if the witness was a software tester. However, we feel very lucky we could interview this guest; he helped us to learn valuable lessons and moreover, it was good for our students to communicate with other domain experts and understand the despair of the users of a suboptimally tested software.

Another limitation is that in each yearly iteration we analyze more or less the same 13 accidents. More accidents might be needed in the future. Luckily, the critical industry is relatively safe nowadays, showing that one managed to learn from the mistakes of the past. However, this means for us that it will be difficult to provide students with serious ”fresh” accidents. As an alternative, the “failures memory” could be extended with the less-critical, yet frequent near-misses that are still jeopardizing modern mission-critical digital systems. Another problem is that the exit survey was filled by only 49 out of the 96 enrolled students. Luckily, the students who did not answer did not drop the course. Moreover, as the assignment took only two weeks, the CAST accident analysis could not be conducted in very much depth. Nevertheless, we believe that our students spent more time on accident investigation than it normally happens in a conventional software engineering course. We demonstrated that even such a lightweight, yet systematic analysis managed to wake up the students, take them out of their comfort zone and “program” them in the

right mindset to start a software testing course, which was our main objective. Of course we do not know for sure how our students will behave after the completion of the course. A follow-up survey would be helpful to verify that indeed the motivation level increased due to the FAIL assignment.

The main external validity issue is related to the possibility to generalize our results in other contexts. The good news is that there is a lot the information freely available on Internet about all the accidents listed in Table. II. We are confident that any teacher can craft their own assignment using the same formula. A less good news is that interviewing a witness of the Therac-25 accidents was an extraordinary opportunity, which can not be easily repeated by other educators. However, we believe that such an interview is nice to have, but not strictly necessary for the success of the intervention. After all, FAIL assignment has been deployed with good results for 15 years without interviewing any witness. Nevertheless, we video-recorded the sessions and plan to build a repository with accidents "memories" for future generations of interested students and teachers. A similar assignment can be embedded in practically any introductory software engineering course, be it on requirements engineering or software modeling. However, we believe that a software testing course is the most suitable host, because the pressure to prevent accidents is higher for testers than for their fellow developers. A FAIL-like assignment is an excellent instrument in achieving this goal. Finally, we did not have a particular reason to schedule this software testing course in the graduate program. We agree that the FAIL assignment could be placed even earlier in the curriculum, in the undergraduate phase.

VI. Related work

We found a valuable collection of notorious historical software-related accidents in the papers by McQuaid [39] and Wong [40]. However, as the main goal of the authors was just to gather accidents, there was no place for a detailed, methodical investigation. McQuaid in [39] mentioned though that STAMP is a promising method to analyze software-related accidents. Wong in [41] summarized many accidents caused by software and also derived interesting lessons, some of them being similar to ours. Israelski in [42] modelled the Therac-25 scenarios from a user's perspective and discussed how usability testing could have prevented the accidents. Accidents analysis is used in teaching engineering courses, such as the course on systems safety taught at MIT by Nancy Leveson [43]. However, their lessons learned are more general, not targeting in particular software testers. Some accidents, such as the Therac-25 overexposures and Ariane-5 mission loss are widely used in software engineering textbooks and CS ethics courses [44], [45]. However, their investigations are not conducted using a systematic approach the way we do. An interesting, slightly different approach, having the same goal, to boost learning by exposing students to failures, is reported in [46], where the teachers engage students in a digital crime investigation to motivate students enrolled in an introductory course on hardware and systems software. To our knowledge, a similar approach of motivating software testing students by investigating accidents using STAMP and meeting witnesses, has not been reported before. In many industries, organizations learn from accidents. Learning from incidents (LFI) is defined as a process through which employees and the organisation as a whole seek to understand any negative safety events that have taken place in order to prevent similar future events [47]. However, software as a newcomer is often not regarded as a risk factor in a system's safety assessment. More efforts will be needed in the future to include software on the list of usual suspects and include software testers in an accident investigation team.

VII. Conclusions and future work

In this paper we shared our experiences with an unconventional assignment where CS students learn from historical software-related accidents. Evaluations show that such an exercise is an excellent prelude to a software testing course. Although traumatic, an in-depth accident analysis can sensitize students to the severe impact of sub-optimal testing on society. As a result, they lose a bit of their naivety and confidence in software; in return, they get the motivation to become better testers, blessed with defensive pessimism, good communication skills and a strong backbone. An encounter with a close-to-the-fire witness adds credibility and power to the proposed approach.

Our positive results invite software engineering educators to include safety science elements to their curricula and call on witnesses of software-related incidents to break the silence and share their valuable, intense experiences. Next step will be creating a repository with artifacts needed to reproduce historical software-related accidents, and making it available to interested institutions. Our hope is that in this way we contribute to educating emotionally engaged professionals, ready to engineer the software-intensive systems we all can rely on.

Acknowledgments

The authors would like to thank Hans van Vliet, for his suggestion to start in 2007 a software testing course at the Vrije Universiteit (VU) in Amsterdam; Nancy Leveson (MIT), who understood like no other the root-causes of Therac-25 accidents, for her inexhaustible energy and dedication ever after, to widely promote systems-thinking for a safer world; Marcel van Herk, Jake van Dyk, Mike Speiser, enthusiastic medical physicists who helped us to connect and eventually set this wonderful project; our reliable and ingenuous VU Software Testing teaching team, with Joshua Kenyon, Jasper Veltman, Glenn Visser, Koen Kahlmann and Erik Link, for their year-by-year help in smoothly running the course; all CS students enrolled in the VU Software Testing course in 2020/2021 and 2021/2022, for their endurance and serious response in our unconventional educational initiatives.

The VU-BugZoo project was funded by the NRO, The Netherlands Initiative for Education Research, as part of a Comenius Teaching Fellow grant.

REFERENCES

[1] T. Astigarraga, E. M. Dow, C. Lara, R. Prewitt, and M. R. Ward, "The emerging role of software testing in curricula," in *2010 IEEE Transforming Engineering Education: Creating Interdisciplinary Skills for Complex Global Environments*. IEEE, 2010, pp. 1–26.

[2] A. Deak, T. Stålhane, and G. Sindre, "Challenges and strategies for motivating software testing personnel," *Inf. Softw. Technol.*, p. 1–15, 2016.

[3] L. F. Capretz, P. Waychal, J. Jia, D. Varona, and Y. Lizama, "Studies on the Software Testing Profession," in *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*, 2019, p. 262–263.

[4] V. Garousi, A. Rainer, P. Lauvås, and A. Arcuri, "Software-testing education: A systematic literature mapping," *Journal of Systems and Software*, vol. 165, p. 110570, 2020.

[5] R. Sprinkle, S. Hunt, C. Simonds, and M. Comadena, "Fear in the classroom: An examination of teachers' use of fear appeals and students' learning outcomes," *Communication Education*, vol. 55, no. 4, pp. 389–402, 2006.

[6] N. Leveson and C. Turner, "An investigation of the Therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18–41, 1993.

[7] J. L. Lyons and Y. Choquer, "Ariane 5 : Flight 501 Failure Report By The Inquiry Board Paris," 2013.

[8] "Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia," https://www.gao.gov/products/imtec-92-26, accessed: 2023-30-01.

[9] "Mars Probe Lost Due to Simple Math Error," https://www.latimes.com/archives/la-xpm-1999-oct-01-mn-17288-story.html, accessed: 2023-30-01.

[10] "Knight Capital Says Trading Glitch Cost It $440 Million," https://archive.nytimes.com/dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/, accessed: 2023-30-01.

[11] N. G. Leveson, *Engineering a safer world: Systems thinking applied to safety*. The MIT Press, 2016.

[12] N. Silvis-Cividjian, R. Limburg, N. Althuisius, E. Apostolov, V. Bonev, R. Jansma, G. Visser, and M. Went, "VU-BugZoo: A Persuasive Platform for Teaching Software Testing," in *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '20, 2020, p. 553.

[13] T. Taylor, G. VanDyk, L. W. Funk, R. M. Hutcheon, and S. O. Schriber, "Therac 25: A new medical accelerator concept," *IEEE Transactions on Nuclear Science*, vol. 30, no. 2, pp. 1768–1771, 1983.

[14] J. Jacky, "Programmed for disaster: Software errors that imperil lives," *The Sciences*, vol. 29, no. 5, pp. 22–27, 1989, september/October.

[15] A. Mathur, *Foundations of Software Testing*. Addison-Wesley, 2008.

[16] R. Patriarca, M. Chatzimichailidou, N. Karanikas, and G. Di Gravio, "The past and present of System-Theoretic Accident Model And Processes (STAMP) and its associated techniques: A scoping review," *Safety Science*, vol. 146, p. 105566, 2022.

[17] A. P. Goncalves Filho, G. T. Jun, and P. Waterson, "Four studies, two methods, one accident–An examination of the reliability and validity of Accimap and STAMP for accident analysis," *Safety science*, vol. 113, pp. 310–317, 2019.

[18] Y. Zhang, C. Dong, W. Guo, J. Dai, and Z. Zhao, "Systems theoretic accident model and process (STAMP): A literature review," *Safety science*, p. 105596, 2021.

[19] "Boeing 737 MAX: What's Happened After the 2 Deadly Crashes," https://www.nytimes.com/interactive/2019/business/boeing-737-crashes.html, accessed: 2020-10-10.

[20] "Summary of the FAA's Review of the Boeing 737 MAX," https://www.faa.gov/foia/electronic_reading_room/boeing_reading_room/media/737_RTS_Summary.pdf, accessed: 2022-13-10.

[21] A. Burger, *Witness: Lessons from Elie Wiesel's Classroom*. Harper One, 2018.

[22] N. Silvis-Cividjian, "A safety-aware, systems-based approach to teaching software testing," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, 2018, p. 314–319.

[23] N. Silvis-Cividjian, "Awesome bug manifesto: Teaching an engaging and inspiring course on software testing," in *2021 Third International Workshop on Software Engineering Education for the Next Generation (SEENG)*, 2021, pp. 16–20.

[24] M. Pezzè and M. Young, *Software testing and analysis - process, principles and techniques.* Wiley, 2007.

[25] L. Copeland, *A Practitioner's Guide to Software Test Design*. Artech House, 2004.

[26] D. Price, "Pentium FDIV flaw-lessons learned," *IEEE Micro*, vol. 15, no. 2, pp. 86–88, 1995.

[27] D. Linnan, "Iran air flight 655 and beyond: Free passage, mistaken self-defense, and state responsibility," 1991. [Online]. Available: http://hdl.handle.net/20.500.13051/6252

[28] "Oct. 26, 1992: Software Glitch Cripples Ambulance Service," https://www.wired.com/2009/10/1026london-ambulance-computer-meltdown/, accessed: 2023-30-01.

[29] P. Coy and M. Lewyn, "The day that every phone seemed off the hook," *Business Week*, pp. 39–40, 1990.

[30] "The Failure of Denver International Airport's Automated Baggage System," https://strategicppm.wordpress.com/2010/06/01/the-failure-of-denver-international-airports-automated-baggage-system/, accessed: 2023-30-01.

[31] C. Borras, "Overexposure of radiation therapy patients in Panama: problem recognition and follow-up measures," *Rev Panam Salud Publica*, vol. 20, no. 2-3, pp. 173–87, 2006.

[32] "Did Mistakes By Crew Doom Flight To Cali? – Debate Surrounds Training, Pilots' Reliance On Computers," https://archive.seattletimes.com/archive/?date=19960118&slug=2309559, accessed: 2023-30-01.

[33] "Hawaii Panics After Alert About Incoming Missile Is Sent in Error," https://www.nytimes.com/2018/01/13/us/hawaii-missile.html/, accessed: 2023-30-01.

[34] T. Song, D. Zhong, and H. Zhong, "A STAMP analysis on the China-Yongwen railway accident," in *Computer Safety, Reliability, and Security: 31st International Conference, SAFECOMP 2012, Magdeburg, Germany, September 25-28, 2012. Proceedings 31*. Springer, 2012, pp. 376–387.

[35] R. Pereira, C. Morgado, I. Santos, and P. R. Carvalho, "STAMP analysis of deepwater blowout accident," *Chemical Engineering Transactions*, vol. 43, pp. 2305–2310, 2015.

[36] F. Mason-Blakley, R. Habibi, J. Weber, and M. Price, "Assessing STAMP EMR with Electronic Medical Record Related Incident Reports: Case Study: Manufacturer and User Facility Device Experience Database," in *2017 IEEE International Conference on Healthcare Informatics (ICHI)*, 2017, pp. 114–123.

[37] S. Gao, S. P. Low, and H. J. A. Howe, "Systemic lapses as the main causes of accidents in the Singapore construction industry," *Civil Engineering and Environmental Systems*, vol. 35, no. 1-4, pp. 81–98, 2018.

[38] B. Beizer, "Software testing techniques," 01 1990.

[39] P. A. McQuaid, "Software disasters—understanding the past, to improve the future," *Journal of Software: Evolution and Process*, vol. 24, no. 5, pp. 459–470, 2012.

[40] W. E. Wong, X. Li, and P. A. Laplante, "Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures," *Journal of Systems and Software*, vol. 133, pp. 68–94, 2017.

[41] W. E. Wong, V. Debroy, A. Surampudi, H. Kim, and M. F. Siok, "Recent catastrophic accidents: Investigating how software was responsible," in *2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, 2010, pp. 14–22.

[42] E. Israelski and W. Muto, "Human factors risk management as a way to improve medical device safety: a case study of the Therac 25 radiation therapy systems," *Jt Comm J Qual Saf.*, vol. 30, no. 12, pp. 689–695, 2004.

[43] N. Leveson, "16.63J System Safety. MIT OpenCourseWare," https://ocw.mit.edu, accessed: 2023-30-01.

[44] I. Sommerville, *Software Engineering*, 9th ed. Addison-Wesley, 2010.

[45] C. Huff and R. Brown, "Integrating Ethics Into a Computing Curriculm: A Case Study of the Therac-25." [Online]. Available: https://onlineethics.org/print/pdf/node/45431

[46] M. S. Conklin, "Teaching it hardware concepts using computer forensics as a motivator," *SIGITE Newsl.*, vol. 2, no. 1, apr 2005.

[47] D. Lukic, A. Littlejohn, and A. Margaryan, "A framework for learning from incidents in the workplace," *Safety Science*, vol. 50, no. 4, pp. 950–957, 2012, first International Symposium on Mine Safety Science and Engineering 2011.