A Case Study: Injecting Safety-Critical Thinking into Graduate Software Engineering Projects

Jane Cleland-Huang and Mona Rahimi
University of Notre Dame, Notre Dame, IN, USA
Email: JaneClelandHuang@nd.edu, m.rahimi@acm.org

***Abstract*—Exposure to safety-critical thinking grows in importance as society increasingly depends upon software to control physical devices with potential safety impacts. In this unique graduate capstone experience we engaged graduate Software Engineering students in the specification, design, implementation, validation, and assurance of potentially safety-critical software-intensive products involving physical devices such as Unmanned Autonomous Vehicles, health-sensors, and/or environmental monitors. While each product had at least one safety-critical usage scenario, it also had harmless test-scenarios which enabled students to design and build with safety-in mind, but to test their product in a safe context. Students engaged in safety-related practices such as hazard analysis, safety-design, safety-assurance, and certification processes. We describe the goals and logistics of the course, discuss student outcomes based on an analysis of the deliverables and student feedback, and suggest ideas for replication and improvement.**

***Index Terms*—Pedagogy, Safety Critical, Capstone**

I. Introduction

Safety-critical projects come in all shapes and sizes, and the degree of rigor invested in building each system varies accordingly [8]. When thinking about safety-critical systems, we tend to focus on systems such as avionics, pacemakers, automobile braking systems, railway signals, or medical infusion pumps which are often developed quite formally following strict certification standards [12]. On the other end of the spectrum, a plethora of systems, with potential safety impacts, are built using comparatively light-weight development processes [16]. Nevertheless such systems can still impact the health and well-being of their users or potential beneficiaries. Examples include certain medical devices, Electronic Health Record (EHR) systems, and Unmanned Autonomous Vehicles (UAVs) which may be used to support activities such as police surveillance, commercial deliveries, or environmental monitoring. In all such projects, the development team is responsible for conducting a safety analysis and designing a solution that demonstrably mitigates potential hazards [6] [20].

While it is not uncommon for universities to include course offerings in Safety-Critical systems, the majority of Software Engineering and Computer Science students do not have the opportunity to engage in beginning to end development of a safety-critical system and are therefore not exposed to practices of safety critical development in a practical, hands-on setting. In this paper we describe our experiences and lessons learned from facilitating a five-month graduate capstone course which emphasized safety-critical software development in a fully immersive setting.

Safety-Critical systems are often defined in terms of criticality levels. For example, the DO-178b guidelines for developing avionics software systems lists five severity levels [4], [21]. From highest to lowest, the levels include (A) Catastrophic – which 'prevents safe flight or landings', (B) Hazardous/Severe-Major – which could potentially 'cause fatal injuries' to a few people, (C) Major – which could cause 'discomfort or possible injuries', (D) Minor – which leads to 'reduced safety margins', and (E) No Effect. Clearly, it is neither desirable nor feasible for students to engage in potentially fatal or harmful software projects; therefore we sought the realism of level A and B projects at the safety of level E.

In this paper we describe our experiences, including course outline, materials, and lessons learned from injecting safety-critical practices into a course entitled "Software Engineering Studio" offered over two 11-week quarters at DePaul University from January to June of 2016. 31 graduate students registered for the first quarter, and 29 continued into the second quarter. Approximately half the students majored in Software Engineering and half in Computer Science. Students were divided into five teams of approximately six students per team. The course was taught by Dr. Jane Cleland-Huang with assistance from Mona Rahimi. The study reported in this paper was conducted under IRB approval.

We made an early decision that all projects would include physical devices because we believed this would increase the realism of the safety-critical development experience. In one case, this decision led us closer to catastrophe than we intended when an Unmanned Autonomous Vehicle (UAV) lost track of its correct GPS location, veered off course, rapidly descended across a highway, crash landed in Lake Michigan, and was never seen nor heard from again. As we discuss later, such realism contributed to a rich educational experience and helped students to understand the implications of their design and development decisions upon the safety of their systems and the potential impact upon innocent bypassers. On the other hand, it instilled in the instructors and students a deep respect for the potential catastrophic hazards of our projects which led to immediate adjustments in field-testing protocols. As one student said "every single UAV project *is indeed* safety-critical."

The published goals of Studio course at DePaul University are to provide graduate level students with the opportunity to design, built, test, and deploy a fully-executable, non-trivial application over the course of five months. To achieve

TABLE I: Course Structure

Setup	Introductory Lectures (5 hours); Team formation; Use case specification; Wireframe prototype; Concept Presentation by teams
Sprint #1:	EARS requirements; Initial safety Analysis (FMECA); Architectural design/analysis; Architectural spikes; Team Presentation: Features and Design mockup
Sprint #2:	Development; Testing; Safety and Security analysis; Clarify assumptions; Introduce Safety Case Notation and tools Team Presentation: Report field test results
Sprint #3:	Development/Testing; Construct and finalize Safety case; Presentation: External panelists; Finalize product; Complete documentation; Presentation: Argue safety case

our emphasis on safety-critical development we extended the learning goals to include the ability to:

- Systematically identify hazardous conditions with software related safety risks.
- Specify safety-related requirements that if fully implemented would mitigate the identified hazards.
- Establish traceability needed to demonstrate product level safety.
- Develop a safety-case to effectively argue for system safety.

The course was divided into an initial project launch phase of five weeks followed by three development iterations of approximately five to six weeks each. In the first two weeks of the course, the instructor described the projects, the software artifacts to be created, and the process that would be followed for the duration of the project. During the first week, all students filled in a detailed preference survey to rank their skills, learning goals, and project preferences. Our primary goals were to place students into one of their top ranked projects while balancing the interests and skills of each team. Teams were then formed in the second week of the studio course. Teams had two weeks to scope out their projects including specifying high-level use cases, sketching out a candidate architecture and low fidelity screen mockups, and modeling runtime data flows. In week five, each team presented their planned product to the class.

The remainder of the course was conducted using agile processes including SCRUM. It was structured around three sprints of five to six weeks duration. The slightly elongated sprints reflected the part-time nature of the project. All teams used Jira to manage their requirements, issues, and tasks and had a private Github repository to manage source code and other artifacts. We also used the Assurance Safety Case Environment (ASCE) tool for building safety cases. All software was used under free educational licenses. The primary deliverables required from each sprint are shown in Table I.

As this was a graduate level Studio course, each team was given significant freedom to evaluate and to select frameworks and programming languages that they wished to use. As a result, there was significant diversity in the delivered projects.

At the end of each sprint all students were asked to describe their individual contribution to the project as well as to provide their perspectives on the Software Engineering and Safety-related activities they had engaged in. These questions were all open-ended and student responses are woven throughout remaining sections of this paper.

II. Safe-for-Students Safety-Critical Projects

Our first task was to identify realistic projects which had clear safety-implications but which would be safe for students to deploy. We quickly ruled out automobile systems, avionics, train signals, and medical device development for obvious reasons including cost, equipment availability, the five-month time constraint, our lack of domain knowledge to build such products, and their high criticality level. We established the following constraints on project selection: (1) the project must feasibly be completed by a part-time team of approximately 6 students within 5 months, (2) equipment costs per project should not be more than US$3,000, (3) the project must have a clear safety-critical scenario matching DO-178b level B or higher, and (4) the project must also have a non-critical and realistic placeholder for the safety-critical scenario matching DO-178b level E. We illustrate how a project could meet these goals by drawing on one of our projects. The environmental monitoring product was designed to monitor toxic gases and nuclear radiation; however, the safe placeholder used sensors that measured ordinary environmental pollutants. In other words, the students built the system with toxic gases/radiation in mind, but tested for non-toxic variants. Each product had its own safety-critical scenario as well as a safe placeholder scenario:

• **Medical Delivery by Drones:** Following a natural or man-made disaster, medicines and other critical supplies may need to be delivered to remote locations. *MedFleet* allows a remote disaster worker to request medical supplies using a mobile app.

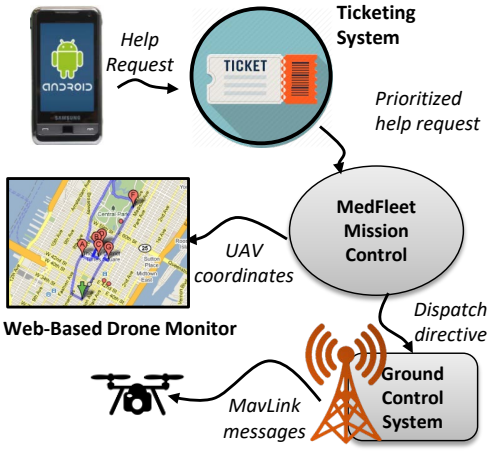

Fig. 1: High Level Architectural Design of the MedFleet System. Safety aspects included UAV control and coordination, and ticketing and prioritizing critical medical cases.

A central triage system receives requests for helps, prioritizes them, and dispatches drones to deliver supplies. **Hardware:** 3DR Iris drones, capable of flying for 20 minutes with a range of 1000 meters from the base station. **Safety Aspects:** Timely delivery of medicines; ensuring drones do not accidentally land on medics, patients, or public; and ensuring that drones do not crash into each other or into obstacles in midflight. The project was a physical simulation of a far larger scale system in which larger, more robust drones could carry medical supplies (or even collect patients), over distances of many miles. Figure 1 illustrates the scope of the MedFleet project.

• **Environmental Monitoring** Environmental pollution including chemical and radioactive sources– whether caused by a catastrophic event or as a byproduct of industry and vehicle emissions creates a health hazard. This product monitors environmental pollutants using fixed sensors as well as crowd-sourced mobile sensors. Data is streamed to a central server where it is analyzed and displayed. Pollution levels are displayed via a mobile app in the form of GPS aware annotated maps and warnings. **Hardware** Cooking Hacks Waspmote Gas Sensors. **Safety Aspects:** Preventing collectors from entering unsafe areas, and ensuring that dangerous gases or radioactivity are reliably detected and warnings sent. The project collects CO_2, CO, and O_3; however, it is designed with the intent of handling more noxious gases and radioactivity.

• **Cardio Rehabilitation** Patients recovering from critical health events, such as cardiac arrest or surgery, must follow careful exercise regimes to build strength and to recover their health. A therapist creates a remediation plan and sets goals for the patient. A health-metrics baseline is established through an initial evaluation session using the health sensors. Safe thresholds and goals are established and the progress of the patient is monitored during the recovery period. **Hardware:** Cooking Hack sensors with capabilities for monitoring heart-rate, temperature, pulse, oxygenation, and Electro-cardiogram activity. **Safety Aspects:** Ensuring that the patient's vital signals remain at acceptable levels during rehabilitation sessions and that therapists set safe levels. While the software was designed for use with patients recovering from health-related events such as heart-attacks, it was tested on healthy individuals.

• **Rescue Mission** The location and health of rescuers such as fire-fighters or earthquake responders on a critical mission are tracked in real-time. Mission decisions are made by central control and communicated via android apps with optional visor displays. The GPS location of each team member is tracked, and rescuers are visually informed of the relative positions of team members. The health of each rescuer is monitored using health sensors with data streamed to central control. **Hardware:** Health sensors, Visors, GPS location from phone. **Safety Aspects:** Accurately tracking health of team members in an environmentally challenging scenario, informing team members of critical mission decisions in a timely manner. The software was tested in a public park.

Fig. 2: The Research Assistant created a proof-of-concept application to read Sensor data prior to the start of the course.

• **Aerial Reconnaissance:** In a search-and-rescue or tracking scenario the UAV is used to perform a systematic search across a variety of terrains to find and track a subject. The subject may be a prison escapee, a missing child, or an injured hiker. Drones are used to perform aerial reconnaissance and to send images back to a central server where they are pieced together and displayed on a map. The goal of the project is an interactive search application. **Hardware:** Camera, Iris 3DR drones **Safety Aspects:** Safe drone flight, clear imagery so that operators can clearly identify targets without loss of subjects.

Given the use of physical devices in each of the projects, there were non-trivial start up costs for this course. To assist other who may wish to replicate our efforts in their own courses, we have established a website[1] which describes the projects in more detail with links to associated hardware components as well as to student websites. Hardware selection for each of the projects involved identifying candidate hardware products, requesting cost estimates and specifications, evaluating the ease of interfacing with the device through its API and supported programming languages, and ultimately committing

[1]Course website: sarec.nd.edu/studio

to a product. We began this process about 4 months prior to the start of the course. We hired an undergraduate student about 6 weeks in advance of the start of the class to work with the physical devices and to establish simple proof-of-concept solutions demonstrating the feasability of integrating software components with each of the hardware devices. The student spent over 80 hours, evaluating instructions provided by the manufacturers, setting up environments, writing and testing code, and developing some "how-to" tutorials which are all publicly available on our Studio website. This student served as an invaluable knowledge resource to the graduate students throughout the Studio course. While the initial investment in setting up physical devices alleviated many problems; interfacing with hardware still created a number of challenges for the students as discussed in more depth throughout the paper.

III. Team Formation

The composition of each team is essential to the success of a project. This was especially true given the challenging nature of the projects in this studio course and the diversity of skills needed to deliver a successful product. Students were asked to classify each of the five projects according to whether the project would be a preferred, acceptable, neutral, or undesirable placement. Students also separately ranked their current skill sets and their 'desire to learn' skills for areas of project management, User-Interface (UX) design, Linux, Java, Low-Level languages, Mobile development, Server-side development, physical device interfacing, Software Architecture, Communication, and Requirements. Our aim was to distribute skills across project teams to the fullest extent possible while simultaneously honoring student's personal preferences and individual learning goals. Based on student preferences, we formed the five teams. 17 students were placed in preferred projects, 13 in acceptable projects, and 1 in a neutral project. Furthermore, we were able to ensure skill distribution across each of the teams. This is particularly important for the targeted projects, which required a blend of safety-analytical skills and hard-core programming.

IV. Suitability of an Agile Process

The studio course offered at DePaul University has a tradition of using Agile development processes as they are a good fit for many characteristics of a student based project including continual requirements discovery, short delivery-based iterations, and team-based efforts. Prior to offering this course we assessed the applicability of using an agile process. Numerous studies have shown that agile processes, if correctly adjusted, can be highly effective for safety-critical projects [13], and several papers report successful industrial projects adopting agile processes for safety-critical systems such as medical devices and automotive braking systems [25], [17]. Gary et al., claim that agile processes are 'suitable for safety-critical systems because these methods are synergistic with safety principles, not orthogonal to them.' They claim that agile methods provide strong support for 'process management and software construction', while following the philosophy that enables 'safety-oriented practices to the extent they are warranted' [7]. This philosophy pervaded the entire Studio course - dictating the effort spent on each safety activity at various phases of the project. For example, students performed an initial safety analysis, but then carefully revisited potential failure modes associated with each feature as it was placed into the sprint backlog. The safety-analysis activities resulted in requirements (aka user stories) which were fed into a Sprint backlog and selected for development in each iteration. While we could not claim all five final products to be certifiably safe for use; we could claim that safety-issues had been analyzed, expressed as mitigating requirements (or user stories), placed into the backlog, and at least partially addressed through three initial sprints which fit into the studio timeplan.

V. Engaging in Safety-Related Activities

In a Road Map paper on Safety-Critical development, Lutz [14] identified several key aspects of a safety-critical project. These included hazard analysis, safety requirements specification, designing for safety, testing, and certification and standards. Our course exposed students to each of these areas and therefore we structure the remainder of the discussion around each of these topics.

A. Hazard Analysis

A hazard analysis identifies and assesses hazards in terms of their severity and likelihood. It typically starts with a preliminary analysis in which unwanted events and their causes are identified. It is then followed by a more systematic analysis of hazards and their contributing faults. Two commonly used techniques are Fault Tree Analysis [22], [23] and Fault Mode Effect Criticality Analysis (FMECA) [19], [15]. A fault tree refines an initial hazard into a series of lower level intermediate or basic events, which, if they occur, would cause the occurrence of the hazard. It uses boolean logic to depict causal events leading to the root node. A FMECA represents a bottom-up, inductive analysis which identifies and documents failure modes, probabilities of their occurrence, and their potential impact on system safety. Failure modes typically include untimely and failed operations as well as lost, intermittent, erroneous, or invalid input. We opted to utilize FMECAs for several reasons. First, we observed their extensive use in our prior work with the US Food and Drug Administration (FDA) on small medical devices [16]. Second, constructing fault trees requires more training than constructing FMECAs, and is more difficult to perform correctly. While safety was an important aspect of this course, we needed to balance this with time needed for other parts of the complete development life-cycle. Sample faults for the MedFleet system are shown in Table II.

Process and Outcomes: We provided students with sample FMECAs taken from a command and control domain. Despite these examples, in the first iteration, students tended to produce rather high-level faults such as "The drone may crash" or "The drone may get shot down in flight". While these are

TABLE II: Fault Mode Effect Criticality Analysis (FMECA)

ID	Fault	Description	Consequence	Level
FM-D1	Faulty data	GPS coordinates do not reflect actual position of drone.	Drones fail to maintain minimum separation distance, crash in midair, and debris falls to the ground.	Critical
FM-E1	Faulty Algorithm	Drone's route is miscalculated	Drone flies into prohibited airspace	Catastrophic

real possibilities, the first is overly high-level, and the second is a security-related threat which could be addressed with missile avoidance features well outside the scope (and budget) of our projects. We encouraged students to refine them into hazards that would be addressable in the software, given the defined scope of the projects. As a result, each team ultimately produced a satisfactory set of more technical hazards and associated failure modes. An example from the mission team is "Individual Unit and/or control center have connection failure and cannot transmit/receive information" or from MedFleet "Drones stats" (i.e. position coordinates and/or health) "are incorrectly stored in the database." Teams identified an average of 7.8 specific failure modes ranging from 4 to 15. While the identified failures were clearly incomplete, they were clearly specified and actionable in terms of design solutions and testing.

Lessons Learned: Students tended to initially short-circuit the hazard analysis process by identifying only the most obvious hazards. There was a tendency to focus on non-software related hazards. The instructor needed to encourage more technical thinking. For example, in the case of the MedFleet and Aerial Reconnaissance projects, given the hazard that the drone might crash, we encouraged students to ask *why?* and to continually ask *why* and *how* questions until they discovered technical faults related to failures in data or events. Because this activity was new to the students, it was most effective when conducted as a small group activity with instructor input. In future offerings of the course we will jumpstart the process with a guided workshop type activity, and also provide students with additional examples of data and event-based faults.

In a follow-up survey at the end of the Studio project over half of the students specifically mentioned that performing hazard analysis was one of the most important lessons they learned. For example, one student said that "the most important lessons I learned" during this project experience "were about FMECA construction and how to represent potential failures." Another student observed that "risk mitigation and fault handling, are critical to the successful performance of the application," while another observed the importance of "identifying potential failures, understanding the effects of such failures, and developing preventive or mitigating solutions to these failures."

B. Safety-Related Requirements

The second safety-related aspect that we emphasized in the Studio projects was that of specifying Safety-requirements. While the use of formal methods [2], [3] enables analysis of the system for completeness and correctness properties and can therefore enhance the safety of a system, we opted against their use in this course for several reasons: (1) students would need to study formal methods in depth, (2) none of the Software Engineering students or Computer Science students in the Studio course had such preparation, (3) the kinds of projects we focused on are rarely supported by formal methods in industry, and (4) our projects were already very extensive in nature and time-constraints did not allow for the inclusion of formal methods training. Therefore, while we clearly see the opportunity for team-members with adequate training to apply formal methods within the context of the Studio projects, we did not include them in this experience.

Process and Outcomes: Students were instructed to carefully analyze safety-related faults in the FMECAs, determine mitigating strategies, and write a set of safety-related requirements (sometimes referred to as design constraints) [14]. Despite our choice to adopt an agile approach, we opted to represent requirements using EARS (Easy Requirements Specifications) [18] instead of following the agile practice of user stories [1]. A user story is typically represented in the form: *As a <type of user>, I want <some goal> so that <some reason>*. However, we felt that this format provided insufficient support for defining the full breadth of systems level requirements required for our studio projects. EARS provides specific keywords to support the specification of four different types of normal operation and one unwanted behavior. It is simple to use and leads to clear and expressive descriptions of the desired functionality. In EARS, Event-driven requirements are specified using the keyword *when*, state-driven use *while*, and optional behaviours use *where*. Ubiquitous requirements, which must always hold, have no keywords. Finally, required responses to unwanted behaviors are specified using conditional clauses (i.e. if.. then the...). Examples of EARS requirements from the MedFleet project are depicted in Table III.

Students were also instructed to think about implicit assumptions they may have made in specifying their requirements, and to document them explicitly as assumptions. For example, a safety-requirement stating that *Drones must maintain a minimimum separation distance of three meters from each other* may depend on domain assumptions concerning the *maximum speed of the drones*, their *wing span*, the *accuracy of the GPS*, and the *responsiveness of the hardware to directives*.

Lessons Learned: Based on our prior use of EARS in the classroom, we already knew it to be highly effective for student learning. One student noted that it was "applicable beyond safety critical systems" and that as "requirements gathering is often a little abstract" it "gives a solid structure for writing clear requirements." During this studio course, we observed that EARS fit well into the overall agile process as implemented in Jira and that it provided natural support for

TABLE III: Sample EARS requirements for the MedFleet project.

Type	Keyword	Example Requirement
Ubiquitous	n/a	The drone shall constantly monitor its remaining flight time against the flight time to return to base.
Event-driven	when	When receiving a new waypoint, the drone shall adjust its flight path and fly to the new waypoint.
State-driven	while	While in flight, the drone shall detect obstacles in its path.
Optional	where	Where the drone has drop capabilities it shall identify a drop point that is at least 5 feet, and not more than 20 feet from the target.
Unwanted	if.. then the	If the drone loses communication with the base station, the drone shall return to its base station position.

expressing a broad range of safety requirements. Only one student commented on the role of environmental assumptions, remarking that safety critical development "puts more pressure on the developers to" be "aware of environmental assumptions and influences."

One other aspect of the requirements process worth mentioning is the fact that we only provided students with general usage scenarios, leaving them the task of scoping their own products and identifying and specifying appropriate requirements. A couple of students mentioned this in their responses, with one stating that "the loose requirements, allowed us to implement whatever features we wanted;" however, another stated that "it would have been nice to have more structured requirements" because setting their own requirements made "it tough to find the balance between ambition and caution." However, while different teams did scope their projects quite differently, the agile nature of the process allowed each team to set a realistic project velocity for individual sprints. Frequent meetings with the instructor (who served as a customer) also helped to alleviate such concerns. Despite quite varied difference in scope, four out of the five project teams delivered the majority of features they initially specified; however, safety requirements were achieved at varying degrees. We discuss this further in the next section on design thinking.

C. Designing for Safety

Students taking the Studio course, especially those enrolled in the Software Engineering MS degree, had significant experience in software- and architectural-design. All had taken Object-Oriented design courses, and several had taken multiple architecture courses. Many students were therefore knowledgeable in the theory of developing high-reliability systems, and practices such as architectural evaluation. They were therefore well-prepared to design architectural solutions which satisfied a broad-gamut of requirements including safety-related ones. In this case study we focus only on the extent to which students were able to effectively design safe solutions.

Process and Outcomes: Each team was required to design and present their architectural solution at various stages – starting from an initial high-level concept through to the level of clearly specified components and interfaces. During these presentations, the students were required to demonstrate how the desired functionality was achieved and also to explain how the design satisfied safety-related requirements. Students showed significant evidence that the safety requirements impacted their design. For example, the environment team originally planned to use a heatmap to represent high pollutant areas, but were unable to "validate that the data displayed was an accurate enough representation" to support the safety-requirements. As a result, their final design used a more accurate marker-based system. In another example, one of the safety-requirements for the MedFleet project specified that "Drones in flight must maintain a minimum separation distance". The team was unable to achieve a general solution by the end of the third sprint; however, the physical reality that Drones could crash in mid-air if this requirement were not handled effectively led the team to deliver a product with reduced functionality allowing only two drones in flight at any one time – each one assigned to a distinct region. This design decision reflected the students understanding that multiple drones could not be simultaneously deployed until this safety-requirement was correctly handled.

Lessons Learned: The students in this Studio Course tackled non-trivial, multi-faceted design problems. The projects were deliberately designed to be challenging and the agile process made it acceptable for the solution to be designed, implemented, and deployed incrementally. Students successfully delivered functionality in parallel to ensuring safety-features at the cost of delivering less functionality. The students reflected on various aspects of their design and design process through their journal entries. For example, one student stated that he/she had focused significant effort on getting reliable readings from the sensors. Given that our sensors were on the inexpensive end of the spectrum this required implementing redundancy into the design – which is a well known strategy for achieving reliability in safety-critical systems. Another student reported that he/she had invested significant effort 'designing and implementing some complex modules' to 'detect and mitigate the causes of hazards' which demonstrated early signs of runtime monitoring and fault recovery. Several students reflected on the need for security in order to achieve safety with comments such as 'security to authenticate and authorize different user types' or in the *mission* system to implement 'several levels of handshake security to ensure connecting units are valid and recognized.' Overall, the students showed significant maturity in the way they designed safety into the system. The non-trivial scope of the projects gave them the opportunity to experiment with safety-related features such as runtime monitoring and failure recovery which would not have been possible without the realism and failures inherent to working with physical devices.

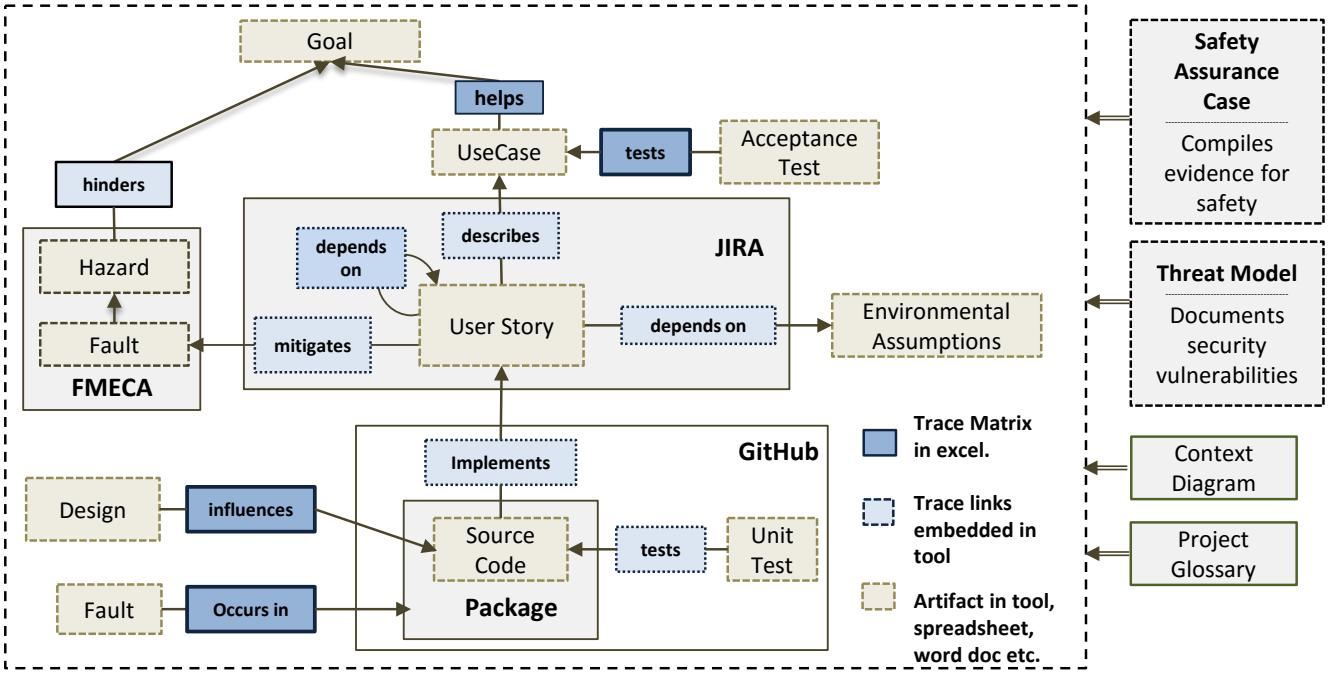

Fig. 3: Traceability Information Model (TIM) showing the artifacts, tools, and traceability paths used for Studio Project

D. Testing Functional and Safety-Requirements

Our Software Engineering students all had prior experience in the area of testing. Students were required to perform testing at multiple levels including unit tests, integration tests, and field-tests. If more time had been available, several of the projects could have benefited from running simulations and/or model checking; however this was out of the scope of our Studio experience. We do not elucidate on testing in depth except to discuss a few safety-related lessons learned.

Process and Outcomes: We did not dictate the specific environment that students should use for testing. However each team invested in establishing test environments. For example, one team set up a Jenkins server to use as a foundation for testing and monitoring the project's webservices and to provide support for continuous integration.

Lessons Learned: Testing was an important element for demonstrating safety. Each team was required to conduct several field tests and to use the results to improve features. We did not observe specific behaviors related to safety, perhaps because each of the products were tested using the 'safe' placeholder scenario and not the actual safety-critical one. However, the inclusion of physical components made things more challenging and one student stated that "our field test turned out to be a little chaotic because of unexpected bugs" and that in retrospect they should have tested more things "prior to going out in the field." The most important lesson we learned however, was to require all field-test protocols to be approved in advance. This would have avoided the case of the drone flying in the vicinity of the highway and crashing into Lake Michigan. In another interesting incident, the environmental team decided to stress-test their CO and CO_2 sensors in a Chicago side-street next to the Computer Science building by taking a transparent tube, placing the sensors and a Raspberry Pi in one end, and setting a small fire in the other end of the tube. While the test in itself was relatively harmless, the physical appearance of the 'test object' may have caused considerable alarm if noticed by security personnel. Lessons learned are clear – all field tests should follow preapproved protocols with no exceptions.

E. Certification and Standards

The final safety-critical focus area defined in Lutz' roadmap is that of certification. As our projects are not governed by formal certifiers such as DO-178b/c or the FDA etc, we decided to focus on the use of Safety-Cases, which are required as part of some certification processes. A Safety-Case provides a semi-formal argument to justify claims that the system will operate safely in its intended environment [10]. It consists of claims representing safety requirements that the system must satisfy, evidence to demonstrate that the safety requirements have been met in the new system, and supporting safety arguments [9], [5], [10]. The claims and evidence in a safety case are supported by traceability links between various software artifacts. We therefore required students to construct such links throughout the SDLC and then to use them to formulate safety-cases.

Ideally, safety-cases should be constructed incrementally throughout the entire project. In an agile project, the safety-case produced at the end of each sprint may be incomplete, but annotated to show areas of incompleteness and/or uncertainty. We experienced a dilemma in deciding when to introduce the notion of Safety-Assurance cases. On one hand, we wanted students to start building safety cases from the start of the

project and to improve them iteratively as the project progressed; but, on the other hand, the students were already stretched with the need to learn new architectural frameworks, new project environments (Git, Travis etc), new languages, and new safety-related techniques (e.g. FMECA). We therefore decided to defer the introduction of safety-cases until halfway through the second sprint. By this point the projects were progressing well but students still had time to perform the safety analysis. Students were therefore only required to deliver a single safety-case at the end of the project.

Process and Outcomes: To support the creation of the safety-cases, each team was required to construct trace links throughout the software development life-cycle. We provided guidance in the form of a Traceability Information Model (TIM) shown in Figure 3. The TIM depicts project artifacts (e.g. requirements, hazards etc) and traceability paths between pairs of *source* and *target* artifacts. None of our students had previously engaged in a project which required the creation and/or use of trace links; however, we provided support for this task through the shared Jira/Github environment. Trace links were captured and documented in three different ways according to the characteristics of the source and target artifacts. Links between requirements (in Jira) and source code (in Github) were automatically created using a Jira/Git bridge whenever students tagged a Git commit with the ID of the Jira user story. Links between Jira requirements and other artifact type (e.g. faults, environmental assumptions) were stored internally within customized fields defined by the instructor within Jira. Finally, links between other pairs of artifacts (i.e. not including Jira or Github) were stored externally in spreadsheets.

For the Safety Cases, we acquired a free educational license to use the Assurance Safety Case Environment (ASCE). Because of the learning curve associated with Safety-Cases, one of the instructors (Rahimi) gave a class presentation early in the second quarter, and then met individually with each team to help them create safety cases. For illustrated purposes we present a small excerpt of a safety-case for MedFleet in Figure 4. This safety case provides claims and evidence arguing that the system is able to compute an upper bound on the error in GPS location. Constructing safety cases required students to learn a new tool and new notation.

Lessons Learned:

Our students had quite a lot to say about the use of safety cases. For example, one mentioned that "Learning about the construction of a safety case was a valuable experience; it helped to put development efforts into a clear context (e.g. we build fault tolerance because the system has to be able to handle failures etc.)" while another stated that "if the software is well documented and there is a comprehensive safety case analysis, then one can have strong evidence that the piece of software is truly safe to use."

They also demonstrated understanding about the role of traceability in the process with comments such as the "trace links of the various elements of safety critical software are extremely important as these will ensure that the various aspects

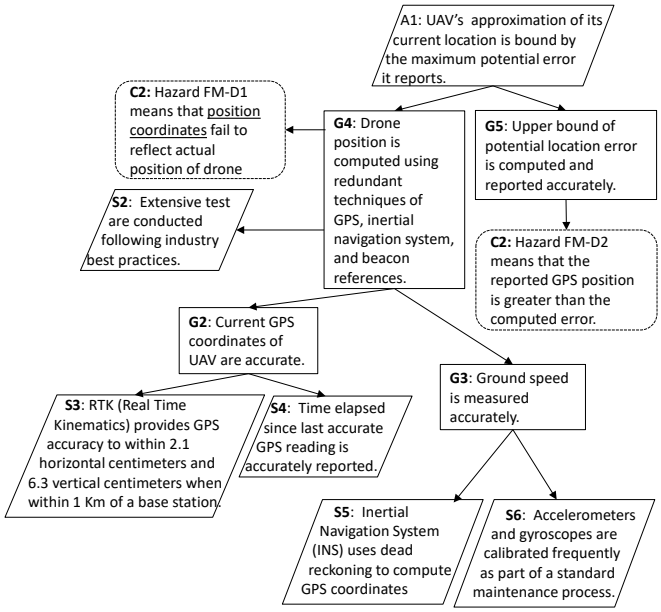

Fig. 4: A Partial Safety Case for the MedFleet project. Note: This version is created by the instructor

of the software are verified to be safe to use". Such comments demonstrated awareness of the benefits traceability can bring to a safety-critical project. In retrospect, the hands-on trainingwas essential for producing high quality safety cases. Despite the fact that we had deferred safety-case construction to late in the project, students were uniformally positive about their experiences. Their journal entries indicated that they understood the importance of constructing an evidence based safety case.

VI. Overall Student Perceptions

This studio course was particularly challenging, for the instructor in terms of planning, prototyping, and purchasing hardware, and also for the students in terms of the additional overhead of working with physical components. The previous discussion has shown that the projects exposed students to safety-critical thinking; however, we were also interested in their subjective perceptions of the course and whether the additional cost and effort was worthwhile in terms of achieving learning goals. The end-of-project survey questions are listed in Table IV.

TABLE IV: End of Studio Survey Questions

Q1	Did you like the safety-critical focus of the projects? Why or why not?
Q2	Did you like the emphasis on physical components? Why or why not?
Q3	To what extent did you feel the following activities contributed to your understanding of safety-critical development? (Please see activity list in Figure X)
Q4	What were the most important lessons (if any) you learned during Studio Course about developing safety-critical software intensive systems?
Q5	In retrospect, what would you like to change about your studio experience?

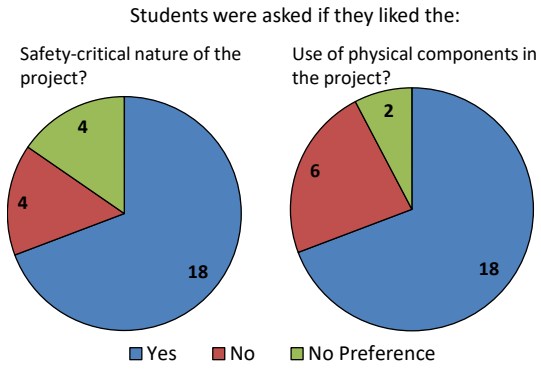

Fig. 5: Student feedback on the focus of Studio Projects

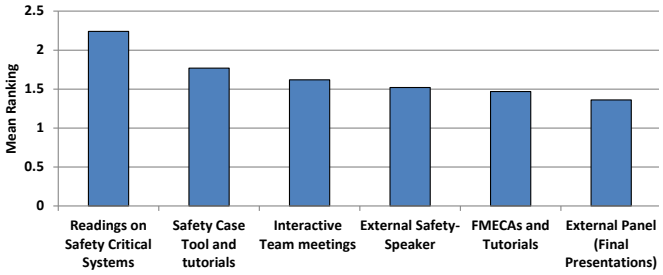

Fig. 6: Student Perceived Usefulness of various Activities

We first asked students for their opinion about the safety-critical emphasis of the course (Q1) and the use of physical computing (Q2). Results are depicted in Figure 5. 18 students liked the focus on Safety-Critical development stating that it added a "real-world" aspect to the course, forced them to "consider the user of the product", "gave insight into some of the more bureaucratic processes that software development has to go through", offered "a first-hand experience in developing safety-critical systems" and for students who had "never had exposure to safety critical applications" required them to "think about what could go wrong and do risk assessments." Finally, one student said "it was the most realistic project I've ever had in an academic setting."

However, not all students felt positively about the safety-critical emphasis. Four of them expressed no preference either way, while four said that they did not like the safety-critical emphasis. These students explained their misgivings with comments such as "I saw it as a random add-on that detracts from the main focus. Safety critical systems could/should be it's own course." Another said that "it gave the project an interesting spin but the purpose was a little confusing at first." Finally, one student mentioned that he felt that his team needed to "stretch for safety-critical." This was a fair comment which likely originated from a member of the aerial project. This project was less directly safety-critical than the other projects and we would likely exclude it from future courses.

Overall, student responses showed positive support for the emphasis on Safety-Critical development. Based on our experience in teaching the course we see the value in teaching a more theoretical course in Safety-Critical systems followed by the hands-on studio project. On the other hand, such a course would likely be offered as an elective and taken by only a subset of students.

We were also interested in whether students felt the physical computing components were an important aspect of the course. In response to this question, 18 students said that they liked the inclusion of physical devises giving a number of different reasons. Positive responses included: "it is interesting to see my software comes physically alive!", "it was unique compared to the overall academic experience.. and provided project experience that was more connected to the real world and forced us to think of broader considerations than other academic projects would," and finally it "made it so fun. No other project I've ever done was like it." The only real negative comment was that "too much time was spent getting these into production" and that a "more fully featured software platform would have better use of time (sic)."

We then asked students to rank several activities in terms of their usefulness for learning about Safety-Critical development with 5 being the highest and 0 being the lowest (Q3). These are shown in Figure 6 and indicate that in general, students found safety-critical readings provided by the instructor and/or discovered themselves to be the most helpful. These were followed by use of the ASCE (safety-case) tool, interactive team meetings, an external speaker on safety, FMECAs, and an external panel that gave interactive feedback towards the end of the project experience. While Studio is not intended to be a lecture-based environment, it is clear that providing external reference material was critical for the students learning.

We were particularly interested in what students had learned and perceived as important to safety-critical projects. Q4 asks them to describe the most important lessons they learned (if any) during Studio Course. The two authors then analyzed and encoded their responses. Some students mentioned multiple factors. Twelve students mentioned the value of learning to use **safety-related tools** and their associated processes. These included ASCE (Safety Cases) and FMECAs with comments such as "the most important thing I learned" was the "processes and tools used to validate the safety of a system." Five students stressed the importance of **considering safety early in the project**. For example, one said that "Safety Analysis should be performed at the early stage of the project before the system architecture is designed so that the preventive solutions or mitigating solutions are integrated into the system design for later implementations." Related to this, two students stressed the need to **balance functionality with safety**, for example by having "a less feature packed application" that is "secure and safe." The other primary category to emerge related directly to lessons learned from hardware integration. Four students commented on this with statements related to learning about hardware and leveraging "hard-set safety parameters in on-board systems". Other comments touched on learning about safety-critical software, requirements specifications, and building safety-assurance cases supported by diverse evidence catpured during the project.

VII. Related Work

While it is not uncommon for universities to include course offerings in Safety-Critical systems, the majority of Software Engineering and Computer Science students do not have the opportunity to engage in beginning to end development of a safety-critical system and are therefore not exposed to practices of safety critical development in a practical, hands-on setting. There are some exceptions. For example Koopman et al. described their experiences of teaching a series of courses in embedded software systems including safety critical ones at Carnegie Mellon University[11]. Sztipanovits et al described their effort in building an embedded software and systems course, including safety critical systems, at Vanderbilt University in [24]. But these courses did not include implementing a functioning product and students were not involved in all phases of the system development including safety analysis. Wolf et al has also describe lessons they learned from teaching the design of modern embedded computing systems at Princeton University and University of Denmark[26]; however, safety topics were only covered in the final weeks of the course.

VIII. Conclusion

In this paper we have shared our experiences in integrating physical computing experiences into graduate level Software Engineering projects. Our analysis of these experiences is based upon student presentations, inspections of the delivered products, and student comments. While we recognize the possibility for bias introduced as part of the survey, the positive responses were also reflected in the course evaluations which are fully anonymous and conducted by university administrators. In general, based on our analysis of the course outcomes and student surveys, we conclude that the Studio project created a productive and effective environment in which students were exposed to many aspects of a Safety Critical project. While this paper represents a case study, and not a controlled study, the evidence for course effectiveness in achieving our goals is quite strong. While our projects are clearly not at the criticality level of an avionics system or an automobile braking system; the experience was realistic enough to provide a really exciting and fun studio environment for both students and instructors, and to successfully expose students to many practices and concepts of safety-critical development.

IX. Acknowledgments

The work in this paper was partially funded by US National Science Foundation Grant CCF:1319680. We also thank Robert Cole for his invaluable assistance in setting up and testing hardware and software interfaces.

References

[1] K. Beck. Embracing change with extreme programming. *IEEE Computer*, 32(10):70–77, 1999.

[2] J. P. Bowen. The ethics of safety-critical systems. *Commun. ACM*, 43(4):91–97, 2000.

[3] M. Chechik, A. Gurfinkel, B. Devereux, A. Y. C. Lai, and S. M. Easterbrook. Data structures for symbolic multi-valued model-checking. *Formal Methods in System Design*, 29(3):295–344, 2006.

[4] Federal Aviation Authority (FAA). *DO-178B: Software Considerations in Airborne Systems and Equipment Certification*.

[5] A. Firm. *ASCAD: Adelard Safety Case Development Manual*. Adelard, 1998.

[6] K. Fowler. *Mission-critical and safety-critical systems handbook: Design and development for embedded applications*. Newnes, 2009.

[7] K. Gary, A. Enquobahrie, L. Ibáñez, P. Cheng, Z. Yaniv, K. Cleary, S. Kokoori, B. Muffih, and J. Heidenreich. Agile methods for open source safety-critical software. *Softw., Pract. Exper.*, 41(9):945–962, 2011.

[8] W. S. Greenwell, E. A. Strunk, and J. C. Knight. Failure analysis and the safety-case lifecycle. In *Human Error, Safety and Systems Development, IFIP 18th World Computer Congress, TC13 / WG13.5 7th Working Conference on Human Error, Safety and Systems Development*, volume 152 of *IFIP*, pages 163–176. Kluwer/Springer, 2004.

[9] O. M. Group. *Structured Assurance Case Metamodel (SACM Version 2.0*. http://www.omg.org/spec/SACM/2.0/Beta1/PDF/ Accessed: 2016-08-11.

[10] T. Kelly and R. Weaver. The goal structuring notation - A safety argument notation. In *Proc. of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[11] P. Koopman, H. Choset, R. Gandhi, B. Krogh, D. Marculescu, P. Narasimhan, J. M. Paul, R. Rajkumar, D. Siewiorek, A. Smailagic, et al. Undergraduate embedded system education at carnegie mellon. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(3):500–528, 2005.

[12] N. G. Leveson. *Safeware, System Safety and Computers*. Addison Wesley, 1995.

[13] M. Lindvall, V. R. Basili, B. W. Boehm, P. Costa, K. C. Dangle, F. Shull, R. T. Tvedt, L. A. Williams, and M. V. Zelkowitz. Empirical findings in agile methods. In *Extreme Programming and Agile Methods - XP/Agile Universe 2002, Chicago, IL, USA, Aug 4-7, 2002*, pages 197–207, 2002.

[14] R. R. Lutz. Software engineering for safety: a roadmap. In *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000.*, pages 213–226, 2000.

[15] R. R. Lutz and R. M. Woodhouse. Requirements analysis using forward and backward search. *Ann. Software Eng.*, 3:459–475, 1997.

[16] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, 2013.

[17] P. Manhart and K. Schneider. Breaking the ice for agile development of embedded software: An industry experience report. In *26th International Conference on Software Engineering (ICSE 2004), 23-28 May 2004, Edinburgh, United Kingdom*, pages 378–386, 2004.

[18] A. Mavin, P. Wilkinson, A. Harwood, and M. Novak. Easy approach to requirements syntax (EARS). In *RE 2009, 17th IEEE International Requirements Engineering Conference, Atlanta, Georgia, USA, August 31 - September 4, 2009*, pages 317–322, 2009.

[19] D. J. Reifer. Software failure modes and effects analysis. *IEEE Trans. Reliability*, R-28,3:247–249, 1979.

[20] J. Rushby. Composing safe systems. In F. Arbab and P. lveczky, editors, *Formal Aspects of Component Software*, volume 7253 of *Lecture Notes in Computer Science*, pages 3–11. Springer Berlin Heidelberg, 2012.

[21] Z. R. Stephenson. *Change Management in Families of Safety-Critical Embedded Systems*. PhD thesis, 2002.

[22] N. R. Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[23] K. J. Sullivan, J. B. Dugan, and D. Coppit. The Galileo fault tree analysis tool. In *Digest of Papers: FTCS-29, The Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing, Madison, Wisconsin, USA, June 15-18, 1999* [23], pages 232–235.

[24] J. Sztipanovits, G. Biswas, K. Frampton, A. Gokhale, L. Howard, G. Karsai, T. J. Koo, X. Koutsoukos, and D. C. Schmidt. Introducing embedded software and systems education and advanced learning technology in an engineering curriculum. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(3):549–568, 2005.

[25] K. Weyrauch, M. Poppendieck, R. Morsicato, N. V. Schooenderwoert, and B. Pyritz. Agile methods for safety-critical software development. In *Extreme Programming and Agile Methods - XP/Agile Universe 2004, 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, 2004, Proceedings*, page 202, 2004.

[26] W. Wolf and J. Madsen. Embedded systems education for the future. *Proceedings of the IEEE*, 88(1):23–30, 2000.