Overview

Course:	CS 152: Programming Languages
Course Level:	Upper-level undergraduate
Course Description:	Comprehensive introduction to the principal features and overall design of both traditional and modern programming languages, including syntax, formal semantics, abstraction mechanisms, modularity, type systems, naming, polymorphism, closures, continuations, and concurrency. Provides the intellectual tools needed to design, evaluate, choose, and use programming languages.[1]
Module Topic:	Managing Risks in Software Design
Module Author:	Michael Pope
Semesters Taught:	Spring 2023
Tags:	Risk [phil], Stakeholders [phil], Harm [phil], Software verification and validation [CS], Design [CS], Programming languages [CS]

Module Overview:	The goal of this module is to consider whether, and to what extent, software engineers have a responsibility to mathematically prove that their software is error-free. The module focuses on managing risks of error *before* any harm has occurred, relying on a distinction between *ex post* and *ex ante* risks. To assess risks that arise from trade-offs in efficiency, security, economy, and safety, the module introduces a stewardship model for software design. The model emphasizes the importance of stakeholder input and oversight. The module concludes with a sorting exercise in which students consider whether formal methods are required for a given software (e.g., mobile payment software, presentation software, wearable fitness technology, online dating website, etc.).	
Connection to Course Material:	This course includes an examination of formal methods for verifying that code is error-free. This module relates the utility of such methods to trade-offs between risk of harm and economic viability.	This module discusses two processes for checking that a software is ready for deployment. The first is validation, which confirms that the software fulfills design requirements. Students discuss how these requirements are generated and can be sensitive to stakeholder interests. The second process is verification, which formally shows that software is designed correctly (i.e., without error). This latter process is very costly (in time and money), introducing questions about economically viable development and potential harms.

[1] Harvard course catalog link. Course website link.

Goals

Module Goals:

1. Familiarize students with stakeholder analysis as a tractable framework for identifying ethical requirements on software performance.
2. Provide students with opportunities to practice devising requirements for applications.
3. Discuss the importance of validation and verification for meeting ethical and performance requirements.

Key Philosophical Questions:

1. What responsibility do software engineers have to prove their software is error-free?
2. How do *ex post* and *ex ante* risks differ and relate to software deployment?
3. How can attention to stakeholder interests in programming and software design promote goods and prevent harms?

Q1: The overarching goal of this module is to promote more responsible design choices by considering trade-offs around deployment in conditions of scarcity and uncertainty.

Q2 and Q3: It is often straightforward to identify potential risks when actual harms occur. This module focuses on the harder case of assessing risks of potential harm. Through sensitivity to stakeholder interests, software engineers can better formulate system requirements and determine when formal verification is required. However, this is not always straightforward, since stakeholders' interests can differ and conflict.

Materials

Key Philosophical Concepts:

- *Ex ante* and *ex post* risk
- Harm
- Stakeholder values and interests

The distinction between risks after some harm occurs (*ex post*) and risks before any harms occur (*ex ante*) helps to narrow the module's focus to risks that arise in software development prior to deployment.

When assessing risks in software development, the module explores advantages and limitations of incorporating stakeholder values and interests. In particular, student discussions invite reflection on ways that sensitivity to stakeholders' interests can (1)

		enhance design requirements utilized to validate software and (2) justify the cost of formal verification.
Assigned Readings:	● Jonathan Jacky (1989), "Programmed for Disaster: Software Errors That Imperil Lives," *The Sciences*. ● Barbara Fried (2018), "Facing Up to Risk"	Jacky's article succinctly introduces the Therac-25 case study as well as relevant considerations for assessing potential coding errors. Fried's article discusses the distinction between *ex post* and *ex ante* risk, as well as challenges for strategies that aim to avoid aggregation in managing potential risks.

Implementation

Class Agenda:	1) Introduction to risk in design: two case studies a) Case Study 1: Ford Pinto (*ex post* risk) b) Case Study 2: Therac-25 (*ex ante* risk) 2) Responsible Stewardship: Stakeholders, rights, and aggregate harms 3) Validation and design requirements 4) Verification and weighing competing concerns i) Case Study 3: Tesla Full Self-Driving System 5) Small-group sorting exercise and final debrief	
Sample Class Activity:	Having distinguished types of risk and introduced ways of integrating stakeholder interests into design requirements and tolerances for risk of error, the module concludes with a small-group sorting exercise. In the small groups, students determine whether a given software requires the use of costly formal methods. Examples include tax filing software, presentation software (e.g., PowerPoint), game apps for children ages 3+, online dating platforms, mobile payment services (e.g., Venmo), home security alarm system, wearable fitness technology, among other items. In addition to determining whether formal verification is required, students formulate reasons that justify meeting such a high threshold, especially in connection with potential harm to stakeholders. A debrief follows the exercise, wherein students from different groups share and discuss the rationale for their results.	Determining the appropriate level of risk tolerance is a matter of practical discernment. This exercise provides students with an opportunity to put the module's content into practice.
Module Assignment:	Within a homework subsection, students are asked to describe approaches to managing risks in deploying software, especially prior to any harms	The questions are designed to achieve two goals. First, the opening questions gauge student

	occurring. Then, supposing that stakeholder interests are relevant to system validation and verification, a set of open-response questions invites students to discuss additional considerations that would help them strike a balance between total risk aversion and reckless deployment.	understanding of the module material. Second, the open-response questions invite students to deeper reflection on responsible software design and deployment.
Lessons Learned:	Student engagement and feedback for this module was positive. A potential source of difficulty in delivering this module is that it abstracts from the logical and mathematical content of the course to focus on practical applications of that content. To serve this goal, it is important to ensure that there is sufficient time for the sorting activity, as well as student participation throughout the module.	