

Introduction

MEB: Prologue, Chapter 1



EECS3342 E: System
Specification and Refinement
Fall 2025

CHEN-WEI WANG

Learning Outcomes

This module is designed to help you understand:

- What a *safety-critical* system is
- Code of Ethics for Professional Engineers
- What a *Formal Method* Is
- *Verification* vs. *Validation*
- *Model*-Based System Development

What is a Safety-Critical System (SCS)?

- A **safety-critical system (SCS)** is a system whose **failure** or **malfunction** has one (or more) of the following consequences:
 - death or serious injury to **people**
 - loss or severe damage to **equipment/property**
 - harm to the **environment**
- Based on the above definition, do you know of any systems that are **safety-critical**?

Professional Engineers: Code of Ethics

- **Code of Ethics** is a basic guide for **professional conduct** and imposes duties on practitioners, with respect to **society**, **employers**, **clients**, **colleagues** (including employees and subordinates), the **engineering profession** and him or herself.
- It is the duty of a practitioner to act at all times with,
 1. **fairness** and **loyalty** to the practitioner's associates, employers, clients, subordinates and employees;
 2. **fidelity** (i.e., dedication, faithfulness) to public needs;
 3. devotion to **high ideals** of personal honour and professional integrity;
 4. **knowledge** of developments in the area of professional engineering relevant to any services that are undertaken; and
 5. **competence** in the performance of any professional engineering services that are undertaken.
- Consequence of misconduct?
 - **suspension** or **termination** of professional licenses
 - civil **law suits**

Developing Safety-Critical Systems

Industrial standards in various domains list **acceptance criteria** for **mission-** or **safety-**critical systems that practitioners need to comply with: e.g.,

Aviation Domain: **RTCA DO-178C** “Software Considerations in Airborne Systems and Equipment Certification”

Nuclear Domain: **IEEE 7-4.3.2** “Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations”

Two important criteria are:

1. System **requirements** are precise and complete
2. System **implementation** conforms to the requirements

But how do we accomplish these criteria?

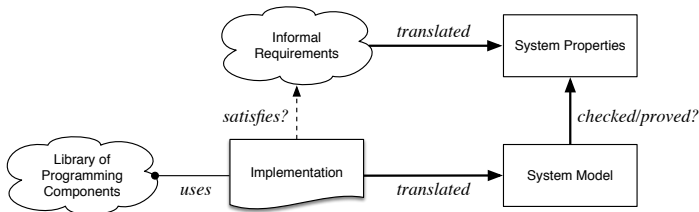
Safety-Critical vs. Mission-Critical?

- **Critical:**
A task whose successful completion ensures the success of a larger, more complex operation.
e.g., Success of a pacemaker $\Rightarrow$ Regulated heartbeats of a patient
 - **Safety:**
Being free from danger/injury to or loss of human lives.
 - **Mission:**
An operation or task assigned by a higher authority.
- Q. Formally relate being **safety**-critical and **mission**-critical.
- A.
- **safety**-critical $\Rightarrow$ **mission**-critical
 - **mission**-critical $\nRightarrow$ **safety**-critical
- Relevant industrial standard: **RTCA DO-178C** (replacing RTCA DO-178B in 2012) "*Software Considerations in Airborne Systems and Equipment Certification*"

Using Formal Methods for Certification

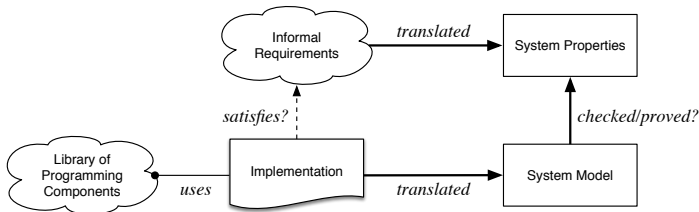
- A **formal method (FM)** is a **mathematically rigorous** technique for the specification, development, and verification of software and hardware systems.
- **DO-333** “Formal methods supplement to DO-178C and DO-278A” advocates the use of formal methods:
*The use of **formal methods** is motivated by the expectation that, as in other engineering disciplines, performing appropriate **mathematical analyses** can contribute to establishing the **correctness** and **robustness** of a design.*
- FMs, because of their mathematical basis, are capable of:
 - **Unambiguously** describing software system requirements.
 - Enabling **precise** communication between engineers.
 - Providing **verification (towards certification) evidence** of:
 - A **formal** representation of the system being **healthy**.
 - A **formal** representation of the system **satisfying safety properties**.

Verification: Building the Product Right?



- **Implementation** built via **reusable programming components**.
- **Goal** : **Implementation Satisfies Intended Requirements**
- To verify this, we **formalize** them as a **system model** and a set of (e.g., safety) **properties**, using the specification language of a theorem prover (EECS3342) or a model checker (EECS4315).
- Two Verification Issues:
 1. Library components may **not behave as intended**.
 2. Successful checks/proofs ensure that we **built the product right**, with respect to the informal requirements. **But...**

Validation: Building the Right Product?



- Successful checks/proofs $\nrightarrow$ We **built the right product**.
- The target of our checks/proofs may not be valid:
The requirements may be **ambiguous**, **incomplete**, or **contradictory**.
- Solution: **Precise Documentation** [EECS4312]

Catching Defects – When?

- To minimize **development costs**, minimize *software defects*.
- Software Development Cycle:

Requirements → *Design* → *Implementation* → Release

Q. Design or Implementation Phase?

Catch defects **as early as possible**.

Design and architecture	Implementation	Integration testing	Customer beta test	Postproduct release
1X*	5X	10X	15X	30X

∴ The cost of fixing defects *increases exponentially* as software progresses through the development lifecycle.

- Discovering *defects* after **release** costs up to 30 times more than catching them in the **design** phase.
- Choice of a **design language**, amenable to *formal verification*, is therefore critical for your project.

Source: IBM Report

Model-Based System Development

- **Modelling** and **formal reasoning** should be performed **before** implementing/coding a system.
 - A system's **model** is its **abstraction**, filtering irrelevant details.
A system **model** means as much to a software engineer as a **blueprint** means to an architect.
 - A system may have a list of **models**, "sorted" by **accuracy**:

$$\langle m_0, m_1, \dots, \boxed{m_i}, \boxed{m_j}, \dots, m_n \rangle$$
 - The list starts by the most **abstract** model with least details.
 - A more **abstract** model $\boxed{m_i}$ is said to be **refined by** its subsequent, more **concrete** model $\boxed{m_j}$.
 - The list ends with the most **concrete/refined** model with most details.
 - It is far easier to reason about:
 - a system's **abstract** models (rather than its full **implementation**)
 - **refinement steps** between subsequent models
- The final product is **correct by construction**.

Learning through Case Studies

- We will study example *models of programs/codes*, as well as *proofs* on them, drawn from various application domains:
 - REACTIVE Systems [sensors vs. actuators]
 - DISTRIBUTED Systems [(geographically) distributed parties]
- What you learn in this course will allow you to explore example in other application domains:
 - SEQUENTIAL Programs [single thread of control]
 - CONCURRENT Programs [interleaving processes]
- The **Rodin Platform** will be used to:
 - Construct system *models* using the Even-B notation.
 - Prove *properties* and *refinements* using *classical logic* (propositional and predicate calculus) and *set theory*.

Index (1)

Learning Outcomes

What is a Safety-Critical System (SCS)?

Professional Engineers: Code of Ethics

Developing Safety-Critical Systems

Safety-Critical vs. Mission-Critical?

Using Formal Methods to for Certification

Verification: Building the Product Right?

Validation: Building the Right Product?

Catching Defects – When?

Model-Based System Development

Learning through Case Studies