Rigorous Methods for Software Engineering (F21RS-F20RS)

Andrew Ireland
Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Course Aims

- To promote an understanding of the issues involved in building high integrity software intensive systems.
- To provide both practical and theoretical insights into industrial strength tools and techniques that promote the development of high integrity software intensive systems.

Course Road Map for Academic Year 2024-25

- The Teaching Team (Edinburgh & Dubai):
  - Andrew Ireland [a.ireland@hw.ac.uk] and Muhammad Najib [m.najib@hw.ac.uk]
  - Hind Zantout [h.zantout@hw.ac.uk]
- This course is being taught to both UG (F20RS) and PGT (F21RS) students.
- Note: the PGT coursework involves additional tasks compared to the UG coursework.

Course Road Map for Academic Year 2024-25

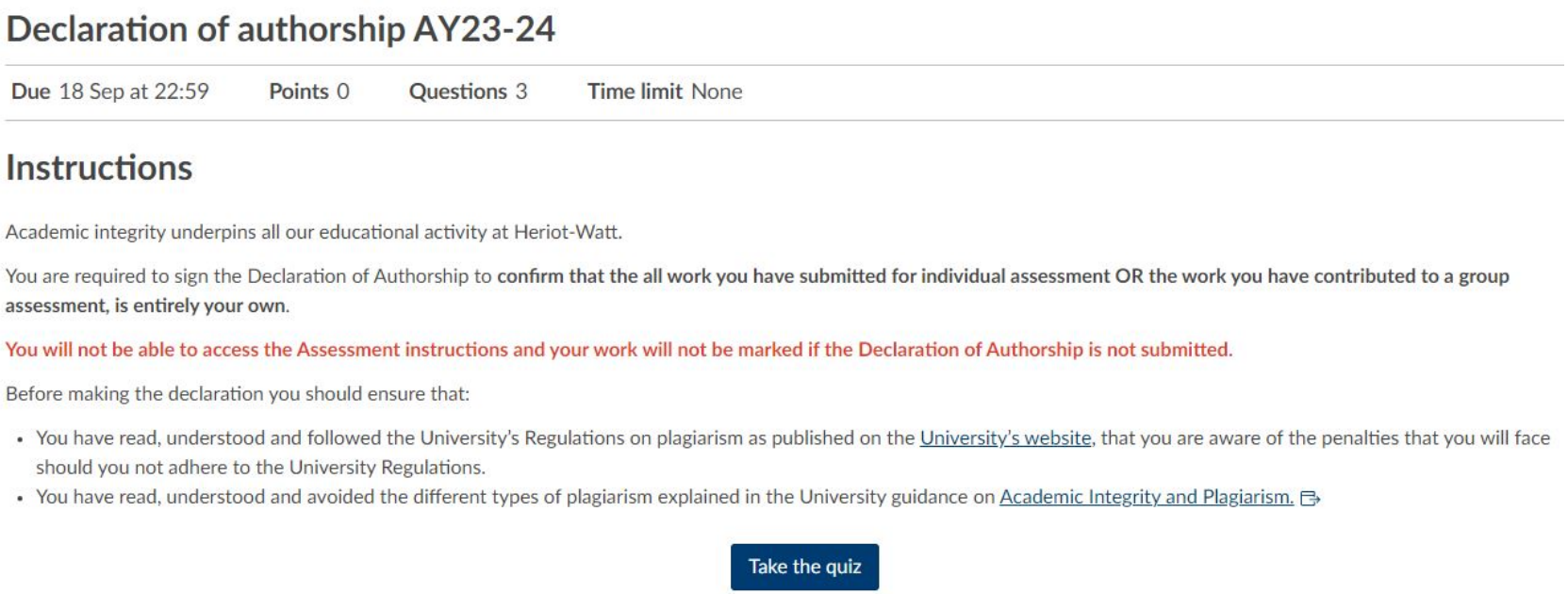
- High integrity software engineering (weeks 1-5):
  - Safe and secure code via structured programming
  - Analysis techniques, including verification via formal proof
  - SPARK programming language and toolkit (Linux and Windows versions)
- Consolidation Week (week 6)
- Design and reasoning (weeks 7-11):
  - Design level specification & analysis
  - Specification via Promela
  - Analysis via Spin model checker (access via Linux and Windows)
- Revision Week (week 12)

Course Road Map for Academic Year 2024-25

- All course materials will be available via Canvas (Virtual Learning Environment):
  - Course Information
  - Additional Reading Materials
  - Core materials organized into weeks, e.g. *Week 1*, *Week 2*, etc.
  - Assignments: specifications and submission links
- Assessment is by 40% Coursework (CW) and 60% Exam:
  - CW1: Handout week 2 with deadline start of week 7 (20%)
  - CW2: Handout week 7 with deadline start of week 12 (20%)
  - An in-person invigilated Exam (60%)
- Reassessment for F21RS is by 100% Exam (there is no reassessment for F20RS).

IMPORTANT: Authorship Declaration (Canvas Quiz)

Deadline for completion: 1pm (BST) Tuesday 17 Sept



Declaration of authorship AY23-24

Due 18 Sep at 22:59 **Points** 0 **Questions** 3 **Time limit** None

Instructions

Academic integrity underpins all our educational activity at Heriot-Watt.

You are required to sign the Declaration of Authorship to **confirm that the all work you have submitted for individual assessment OR the work you have contributed to a group assessment, is entirely your own.**

You will not be able to access the Assessment instructions and your work will not be marked if the Declaration of Authorship is not submitted.

Before making the declaration you should ensure that:

- You have read, understood and followed the University's Regulations on plagiarism as published on the University's website, that you are aware of the penalties that you will face should you not adhere to the University Regulations.
- You have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism.

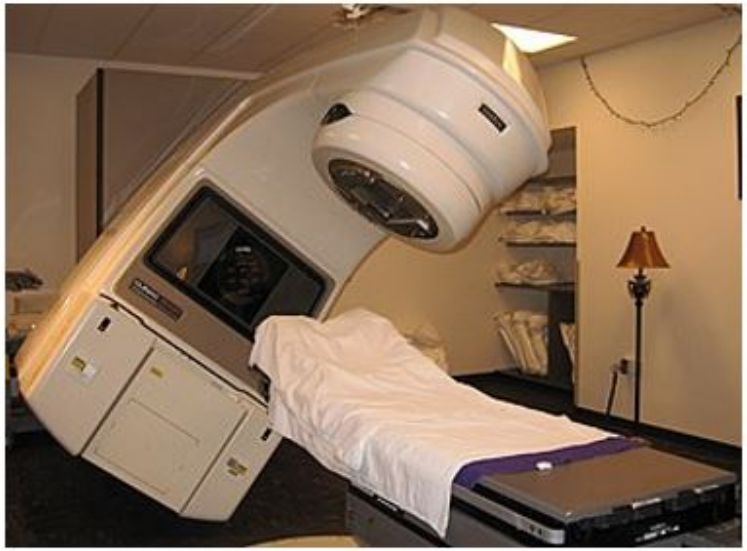
Take the quiz

Failure to complete the Authorship Declaration prevents you from submitting Coursework on Canvas.

Course Road Map for Academic Year 2024-25

- Delivery in Edinburgh:
  - A weekly in-person 2-hour **Class** (with an interval).
  - A weekly in-person 1-hour **Programming Lab**.
- Timetabled slots in Edinburgh:
  - **Class:**
    - Tuesday 2pm-4pm
    - Room 1.13 in the **David Brewster** (DB) building
  - **Programming Lab:**
    - Thursday 10am-11am
    - Either 2.50 (Linux) or 2.52 (Windows) in the **Earl Mountbatten** (EM) building.
- Exception – no Tuesday **Class** in week 6 although there will be an optional **Programming Lab** in week 6.

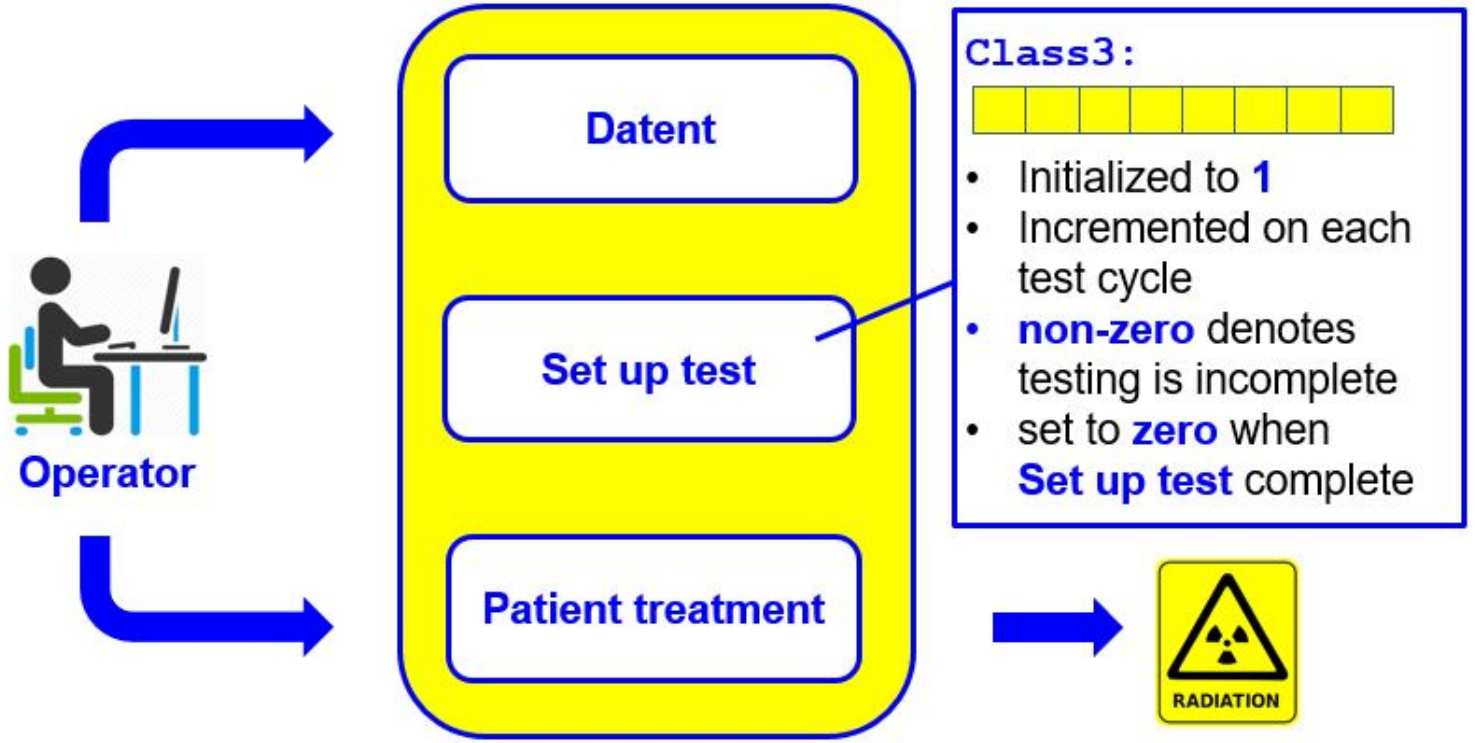
Motivation: Cost of Failure



Therac-25

- Computer-controlled radiation therapy machine
- Developed by Atomic Energy of Canada Limited (AECL) in the early 1980s
- Therac-25 relied heavily on software to ensure safety compared to its predecessors - Therac-6 and Therac-20
- Between 1985 and 1987 the Therac-25 massively overdosed six patients

Motivation: Therac-25 A Closer Look



- If `Class3` reaches **255** then the next increment sets it to **zero**
- If the **Operator** selects the **Patient treatment** when this wraparound occurs then the **Set up test** will be incomplete

High Energy Beam + Test incomplete = Accident

Motivation: Software Weaknesses

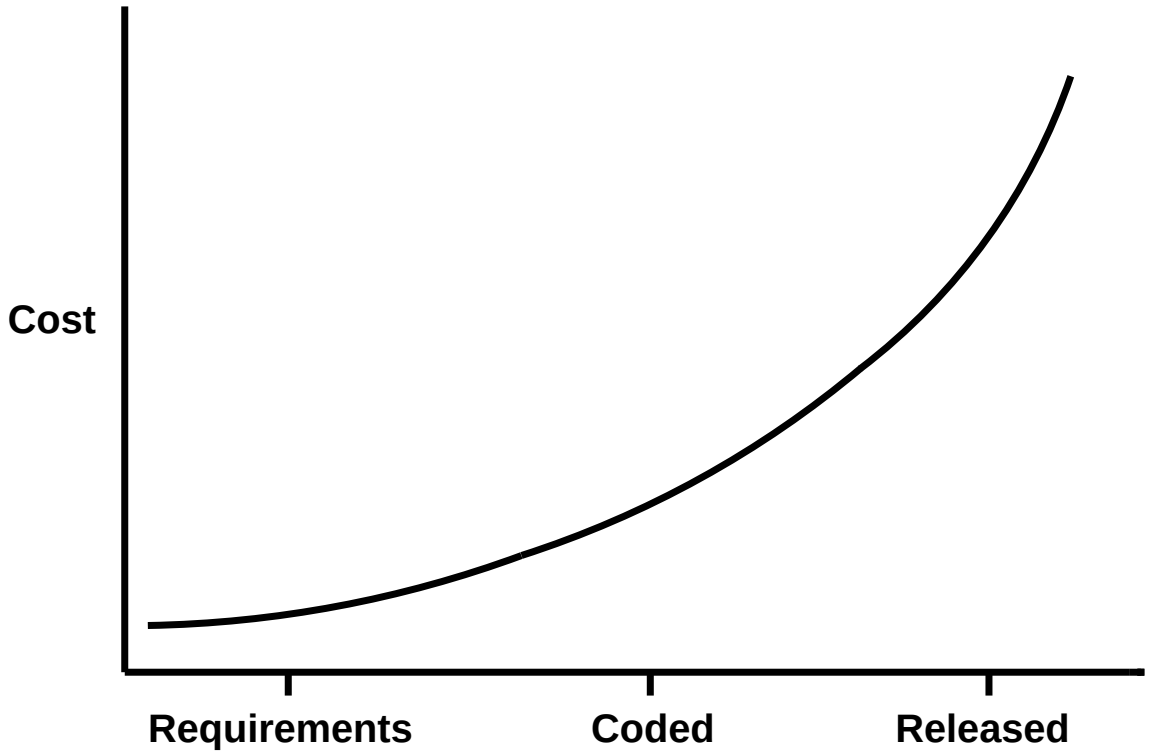
https://cwe.mitre.org/

Top 25 Weakness:

#5: *Improper Restriction of Operations within the Bounds of a Memory Buffer*

#11: *Integer Overflow or Wraparound*

The Economics of Defect Detection



(Boehm, 1976)
Late life-cycle fixes are generally costly, *i.e.* can range from 40% to 100% more expensive than corrections in the early phases.

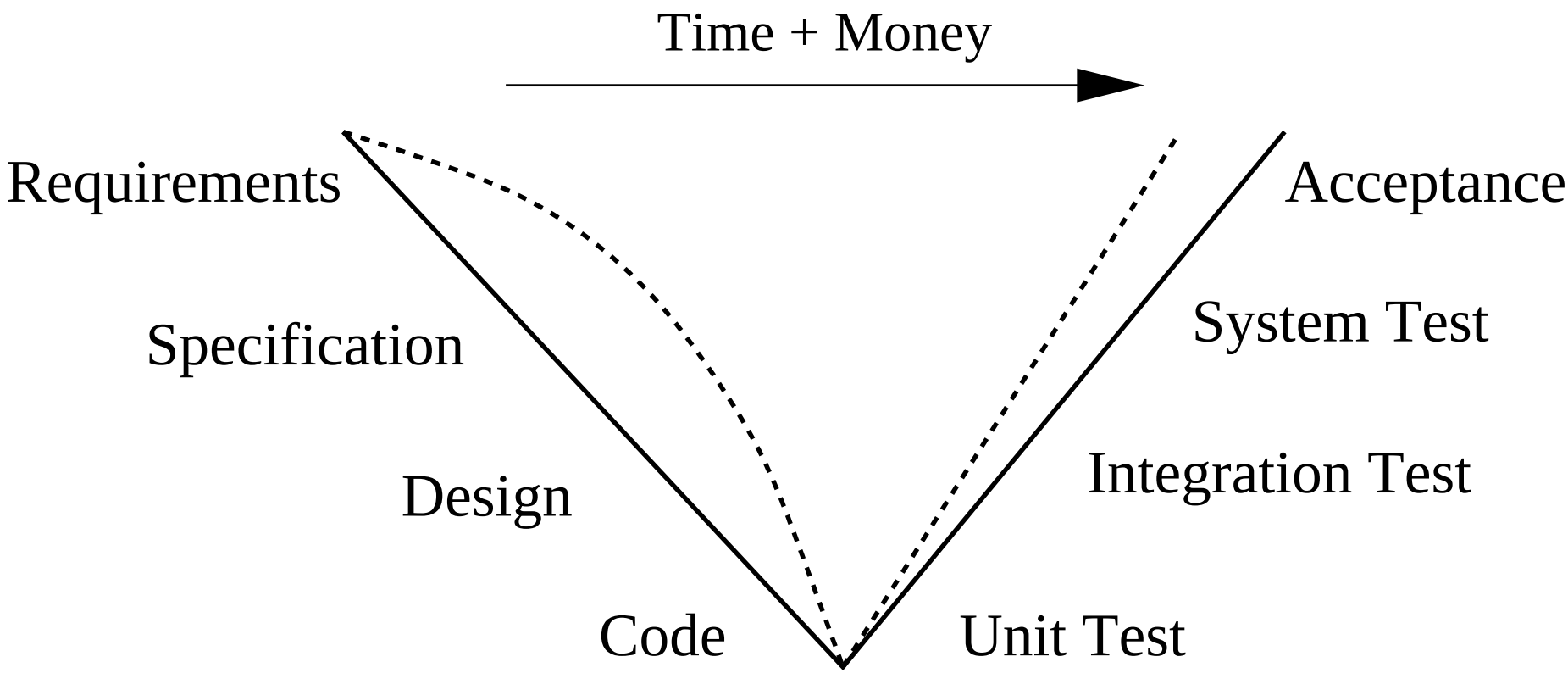
More of the Same?

- Dynamic analysis has been the main approach used by industry in establishing the correctness of software intensive systems.
- Dynamic analysis involves the execution of the system under test – so conventionally corresponds to software testing.
- However, exhaustive testing is not possible for realistic software intensive systems.
- Worse still, omissions and defects introduced early within the development life-cycle are the most expensive to rectify if they go undetected until testing and beyond ...

Complementary Methods: Rigorous and Formal

- Static analysis does not involve the execution of the system under test – such methods operate instead directly on the "representation" of the artifact under test, e.g. code, assertions, contracts, design models, etc.
- Static analysis methods can potentially be applied early within the development life-cycle.
- There exists a broad spectrum of static analysis methods, i.e. from Lint – *heuristic based checker for detecting generic coding errors in C programs* – to the use of mathematical proof in verifying properties of programs.
- This course will focus mostly on the **rigorous** end of the spectrum, and in particular on so called **formal methods** which rely heavily on formal notations and reasoning.
- The power of formal reasoning lies in its ability to establish evidence about a system's correctness in general, rather than in terms of specific test cases.

Drivers: Business & Economic Related



Conventional methods profile: ——————

Formal methods profile: - - - - - - - - - -

Health Warning

- There are no absolute guarantees.
- When applied correctly, formal methods have been demonstrated to result in systems of the highest integrity.
- Correctness is only guaranteed with respect to a specification — you need to validate the assumptions which under-pin the specification.
- Formal methods complement rather than replace conventional approaches, *e.g.* testing, simulation and prototyping.
- But formal methods are applied by humans who are error prone — so tools are crucial.

Drivers: Safety Related Standards

- RTCA DO-178B and DO-178C (USA Civil Avionics)
- Def Stan 00-55 (UK MoD)
- ITSEC (IT Security Evaluation Criteria)
- IEC 61508 (Generic "Programmable Systems")
  - IEC 601 (Medical Equipment)
  - (Pr)EN 50128 (Railway Industry)
- IEC 880 (Nuclear Power Control)
- MISRA (Automotive Industry)
- FDA (Medical Equipment)

Software Standards

- Software standards encapsulate the lessons learned by trial and error on government and commercial projects.
- Standards will typically evolve and change over time.
- Standards are guidelines that can be tailored to the characteristics of a particular project.
- Standards assist in the development of high quality software while reducing time-to-market.
- Standards play a crucial role within the development of high integrity software.

Formal Methods for Design & Code Generation

- Z (Oxford) – model based formalism for specifying systems (set theory based).
- B-Method (ClearSy) – similar motivations to VDM, but automatically generates sequential code, i.e. C, Ada.
- Event-B (Systerel) – event based formalism for system-level modelling that supports refinement.
- VDM (IBM, IFAD, CSK) – model based formalism that supports refinement, i.e. top-down incremental development of systems.
- Alloy (MIT) – a formal modelling language & model checker.
- SPIN (NASA) – one of the most popular tools (model checker) for verifying (and detecting bugs) in concurrent/distributed system designs. (focus of weeks 7-11 – theory and practice).

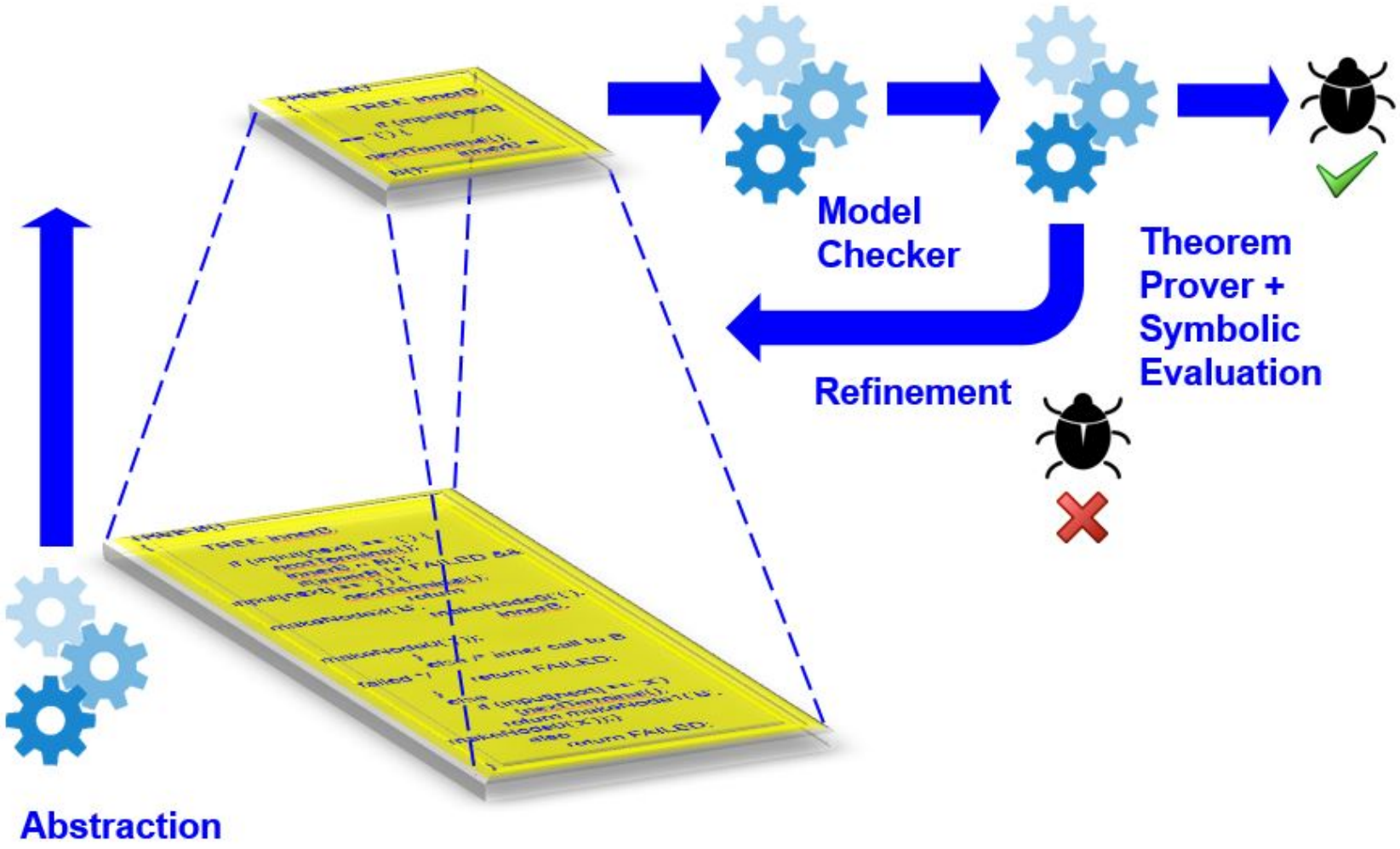
Formal Methods for Static Analysis of Code

- Frama-C: specification and analysis for a subset of C (CEA LIST & INRIA Saclay).
- eCv - specification and analysis for a subset of C (Eschertech).
- Spec#: specification and analysis of C# (Microsoft).
- SPARK - specification and analysis for a subset of Ada (Altran - *formally Praxis*) (focus of weeks 1-5 – theory and practice).

Formal Methods in Industry

- Microsoft Research - Static Device Verifier (SDV) automatically checks code of Windows device drivers (i.e. written in C).
- Ensures that a device driver interfaces correcting with the OS kernel and guards against security threats.
- SDV represents a tight integration of automated reasoning tools.

SDV: A Closer Look



Formal Methods in Industry

- SPARK projects: `https://www.adacore.com/industries`
- SPIN projects: `http://spinroot.com/spin/success.html`
- J. Woodcock and P.G. Larsen and J. Bicarregui and J.S. Fitzgerald, Formal methods: Practice and Experience, *ACM Computing Surveys*, Vol 41, No. 4, 2009. *(available on Canvas – see "Recommended Reading" folder with in the Learning Materials).*

Summary

Recommended reading:

- Spin http://spinroot.com/
- SPARK https://www.adacore.com/sparkpro
- Frama-C: https://frama-c.com
- eCv: http://eschertech.com/products/ecv.php
- Z: http://www.zuser.org
- B-Method http://www.systerel.fr
- Event-B http://www.event-b.org/
- VDM http://overturetool.org/
- Alloy https://alloytools.org
- Spec#: https://www.microsoft.com/en-us/research/project/spec/