High Integrity Computing (F23PS2)

High Integrity Software Development (F29HD2)

Introduction

Andrew Ireland

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

High Integrity Computing?

- The goal of high integrity computing is to provide tools, techniques and methodologies that effectively support the development of highly reliable software:

$$\text{Reliability} = \text{Robustness} + \text{Correctness}$$

- Crucial for software systems that are:
  - safety critical: avionics, automotives, ...
  - security critical: telecommunications, smart cards, ...
  - business critical: databases, non-safety related embedded systems, ...

Note that there is increasing evidence that a high integrity approach means **getting the software right first time.**

Module Road Map

- High integrity software engineering:
  - Safe and secure code via structured programming
  - Program analysis techniques
  - Formal verification

  [`http://www.macs.hw.ac.uk/~air/hic-hisd/`]
- Delivery = Lectures + Labs + Workshops + Directed Reading

Module Road Map

- BSc Assessment:
  - Written examination week 10 [80%]
  - Coursework: Software development exercise [20%]
- MSc Assessment:
  - Written examination (start of Term 3)
  - Coursework: Software development exercise + comparative study
- Related modules include: Formal Specification; Prototype from Specification; Software Analysis & Testing; Logic & Proof; Automated Reasoning for Software Engineering.

Schedule

Week	Mon 14.15	Tue 14.15	Fri 13.15	Assignment
1	Lec 3.36	Lec 2.44	Lec 1.82	
2	Lab 2.50	Lec 2.44	Lec 1.82	Handout
3	Lab 2.50	Lec 2.44	Wor 1.82	
4	Lab 2.50	Lec 2.44	Lec 1.82	
5	Lab 2.50	Wor 2.44	Wor 1.82	
6	Lab 2.50 (BSc) Lec 3.36 (MSc)	Lec 2.44	Lec 1.82	
7	Lab 2.50	Wor 2.44	Wor 1.82	
8	Lab 2.50	Lec 2.44	Wor 1.82	
9	Lec 3.36	Lec 2.44		BSc Deadline
10				MSc Deadline

Overview

- Standards for system and software development
- The IEC 61508 standard in particular
- High integrity software and formal methods

Software Standards

- Software standards encapsulate the lessons learned by trail 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.

Examples of Software Standards

- **MoD 00-55:** *Requirements for the Procurement of Safety-Critical Software in Defence Equipment.* Produced by the UK MoD. The use of formal methods, in particular formal code level verification is mandatory.
- **DO-178B:** *Software Considerations in Airborne Systems and Equipment Certification.* Produced by the US Government and relates to civil aircraft. Key standard within the US and European aircraft industry. Places significant emphasis on software testing, in particular MC-DC testing.

More Examples of Software Standards

- **ITSEC:** *Information Technology Security Evaluation Criteria.* Produced by UK Government. Formal methods required for the highest levels of security, *e.g.* for banking and state security related applications (NSA, GCHQ).
- **IEC 61508:** *Functional Safety: Safety-Related Systems.* Produced by the International Electrotechnical Commission. Key feature of IEC 61508 is that it is a **generic standard**, that is, its scope is not limited to a particular industrial sector. The idea is that it provides a standard upon which sector-specific standards can be defined.

Safety Integrity Levels (SILs)

- SILs are a **qualitative** measure of the required protection against software or system failure.
- Based upon the results of hazard analysis and risk assessment, SILs are assigned to the safety functional requirements.
- SILs provide guidance in selecting appropriate techniques and measures for safety related software development.
- SILs are understandable across different industrial sectors and between customers and software vendors.

Example SILs Definition: IEC 61508

Safety Integrity Level	Low Demand Mode of Operation (Average probability of failure to perform its safety function on demand)
4	$\geq 10^{-5}$ **to** 10^{-4} ($> 99.99\%$ **reliable**)
3	$\geq 10^{-4}$ **to** 10^{-3} ($> 99.9\%$ **reliable**)
2	$\geq 10^{-3}$ **to** 10^{-2} ($> 99\%$ **reliable**)
1	$\geq 10^{-2}$ **to** 10^{-1} ($> 90\%$ **reliable**)

Software Specification: IEC 61508 Examples

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Structured Methodology	HR	HR	HR	HR
Computer-aided Tools	R	R	HR	HR
Semi-Formal Methods	R	R	HR	HR
Formal Methods	NC	R	R	HR

HR = Highly Recommended; R = Recommend;
NR = No Recommendation; NC = No Comment

Software Design: IEC 61508 Examples

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Fault detection & diagnosis	NR	R	HR	HR
Error detecting codes	R	R	R	HR
Programming with assertions	R	R	R	HR
Diverse programming	R	R	R	HR
Recovery blocks	R	R	R	R

A key goal should be to minimize the number and complexity of critical components. Particular attention should be paid to the boundaries between critical and non-critical components.

Software Implementation: IEC 61508 Examples

Technique	SIL 1	SIL 2	SIL 3	SIL 4
Modular approach	NR	R	HR	HR
Defence programming	NC	R	HR	HR
Code standards	R	HR	HR	HR
Analysable programs	R	HR	HR	HR
Suitable programming language	HR	HR	HR	HR
Language subset	NC	NC	HR	HR
Certified translator	R	HR	HR	HR
Verified library modules	R	HR	HR	HR

Suitable Programming Languages

- Programming languages typically contain features that increase the risk of software failures, *e.g.*
  - undefined behaviour (evaluation order in C)
  - behaviour hard to predict (dynamic binding in C++)
- For some programming languages there exists subsets that eliminate the unsafe features, *e.g.*
  - MISRA C: Motor Industry Software Reliability Association
  - SPARK: Ada subset, Praxis Critical Systems (more later)
- IEC 61508 highly recommends the use of Ada, Modula-2, Pascal and subsets of Fortran.
- IEC 61508 does not recommend the use of C without subsets.

Software Verification: IEC 61508 Examples

Technique	**SIL 1**	**SIL 2**	**SIL 3**	**SIL 4**
Formal Proof	**NC**	**R**	**R**	**HR**
Probabilistic testing	**NC**	**R**	**R**	**HR**
Static analysis	**R**	**HR**	**HR**	**HR**
Dynamic analysis & testing	**R**	**HR**	**HR**	**HR**
Software complexity	**R**	**R**	**R**	**R**

Formal Methods

Formal Methods = Formal Modelling + Formal Analysis

- Requirements capture
- Modelling
- Specification
- Analysis
- Documentation

Note that in reality some of these phases may overlap, in addition the overall process should be seen as iterative rather than sequential.

Rigour in the Application of Formal Methods

- **Level 0:** No use of formal methods.
- **Level 1:** Use of concepts and notations from discrete mathematics.
- **Level 2:** Use of formalized specification languages with some mechanized support tools.
- **Level 3:** Use of fully formal specification languages with comprehensive support environments, including mechanized theorem proving or proof checking.

Source: Rushby, *Formal Methods and the Certification of Critical Systems.* Technical Report CSL-93-7, SRI International, Menlo Park, CA.

Modelling Sequential Systems

- **Algebraic specification:** typically applied to abstract data types. A specification has 2 parts:
  - a "signature" - syntax and types of functions & relations
  - a set of axioms

  where the axioms define the meaning of the functions & relations. The best known example of algebraic specification is **OBJ**.
- **Model-based specification:** involves the development of a model of the system in terms on mathematical entities, *e.g.* sets, sequences, relations, ... *etc.* The best known model-based approaches are the **Vienna Development Method (VDM)**, **Z**, and the **B-Method**.

Summary

Learning outcomes:

- Understand the notion of SILs.
- Gain an understanding of the role of standards within the development of high integrity software, and where formal methods fit in.

Recommended reading:

- "Safety-Critical Computer Systems", Storey, N. Addison-Wesley, 1996.
- `http://www.afm.sbu.ac.uk/archive/` [formal methods]
- Formal Methods Specification and Analysis Guidebook for the Verification of Software Systems: Volume II A Practitioner's Companion, Published by NASA, see `http//eis.jpl.nasa.gov/quality/Formal_Methods/`