KU LEUVEN

KU Leuven Technologiecampus Oostende

'Techniek die werkt'

Mechatronics and Embedded Software @ KU Leuven Oostende/Brugge

prof. dr. ing. Boydens Jeroen

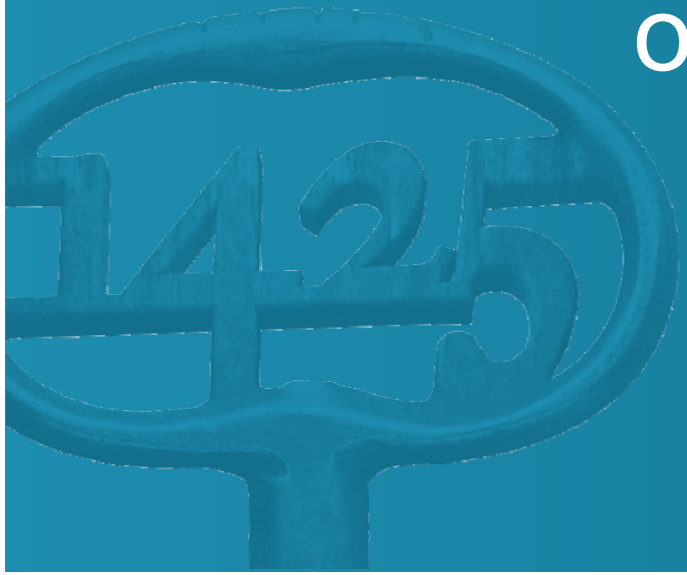
24/09/2015 AmiEs-2015

Ambient Intelligence and Embedded Systems

abstract

- Mechatronics is concerned with the development of as well intelligent production lines as intelligent products. In our current society they take an ever more increasing role as these systems can be found anywhere. Ranging from home environments such as digital cameras, to industrial environments such as smart agriculture machines.
- The software part of these Cyber Physical Systems is becoming more important as more and more components are programmable. This talk will focus on software engineering strategies that support the development of embedded software for CPS's. Focusing on current research in agile methodologies such as test-driven development and techniques to build resilient embedded software. This resilience copes with disturbances in software leading to dataflow or control flow errors. This resilience is positioned in relation to functional safety standards to which these systems must adhere.

Test-Driven Development of embedded software



KU LEUVEN

Problem statement

Two trends influence the way embedded software is developed

1) *Embedded system level*:

Embedded software plays a more important role

2) *Global level*:

Embedded systems are becoming more pervasive in our lives (even in critical domains)

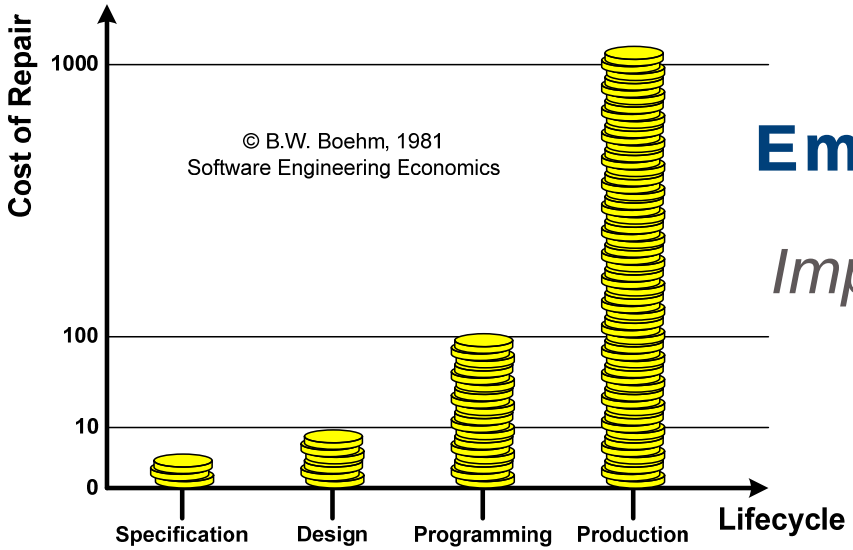
- Consumer electronics
- Automotive / transportation
- Medical applications

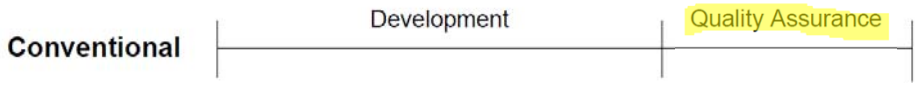
Risk = Importance x Chance of failure

2 1

Embedded quality assurance

Quality assurance in embedded software development is mostly limited to debugging and final testing, only focusing on the current issue…





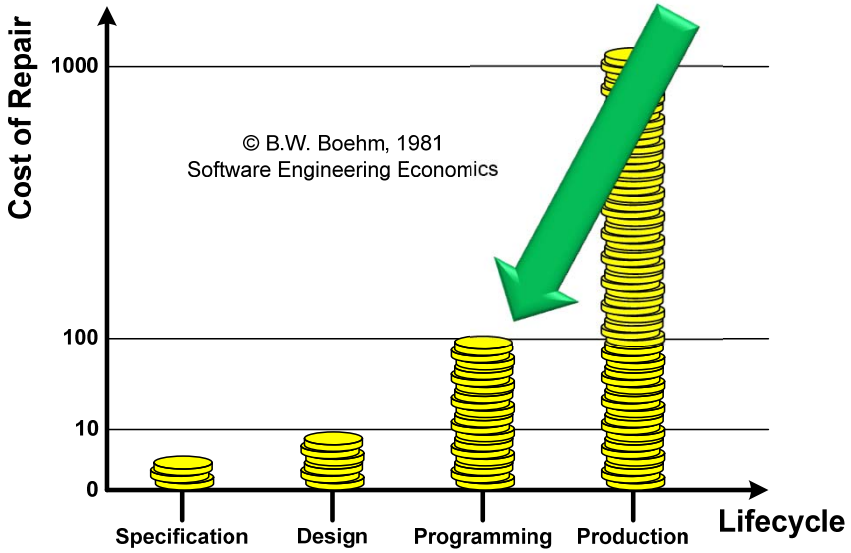
Embedded software testing gap

Importance versus *state* of testing

Automated testing

Timely detection of bugs by automated testing

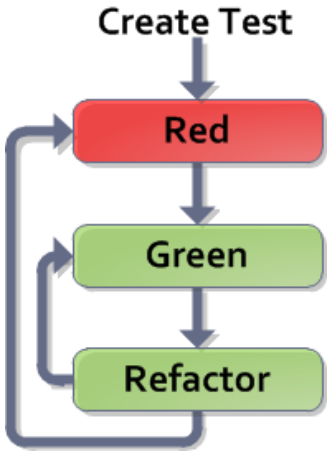
- Running test suite frequently during development
- Incrementally expanding the test suite
- Support from unit testing framework



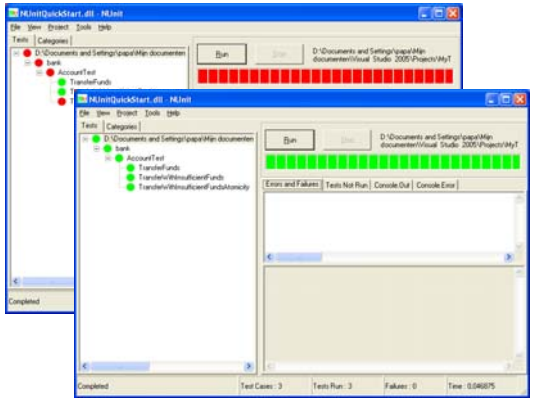
1. Find bugs early
2. Measurable quality
3. Increasing confidence
4. Detecting regression
5. Encapsulating third party code

Test-Driven Development

TDD cycle



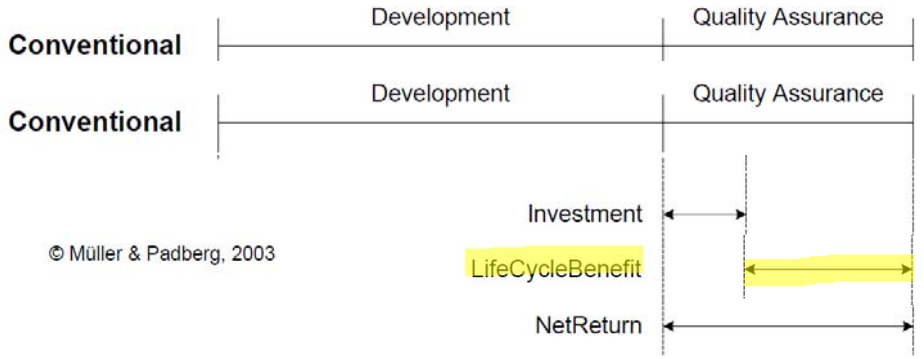
1. Write failing test (red)
   - New behavior
   - Minimal skeleton to get through compilation
2. Write code to pass test (green)
   - Minimal implementation
3. Refactor
   - No new behavior
   - Clean code
   - Keep tests passing



Test-Driven Development

Advantages

1. Code is tested while it's written
2. Fast feedback cycle
3. Extensive & safe refactoring
4. Focus on current functionality
5. Tests become living documentation



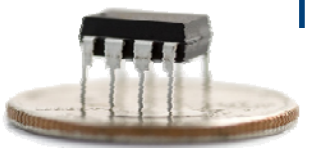
Disadvantages

1. 2x code

   increase development time

Embedded constraints

↔ Embedded TDD challenges

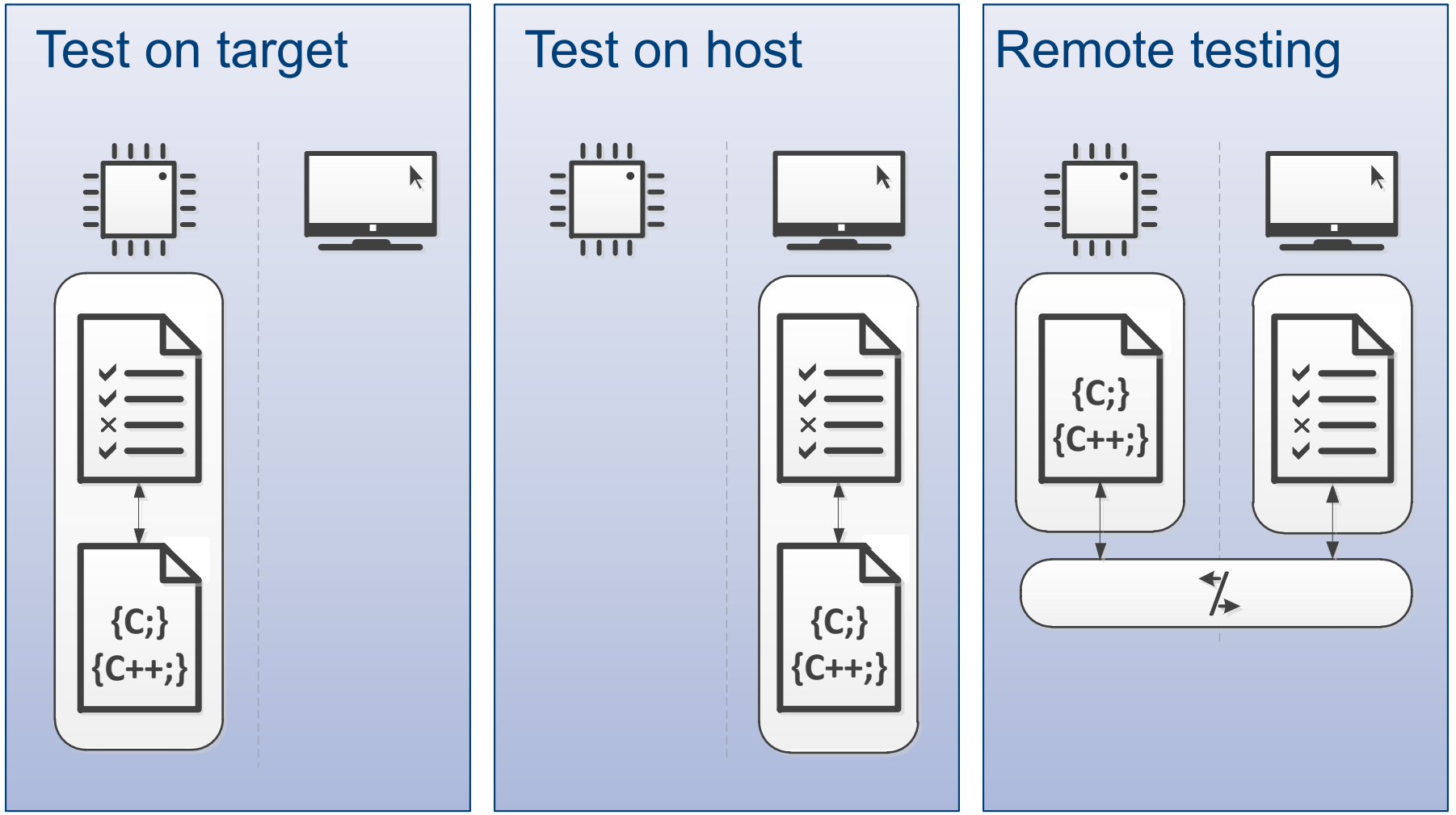


1. Limit the memory footprint needed for TDD on target hardware

2. Tests for
    1. Hardware independent code
    2. Hardware aware code
    3. Hardware specific code

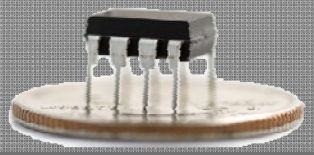
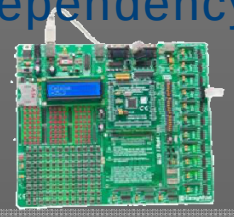
3. Maintain a fast programming cycle

TDD strategies for embedded

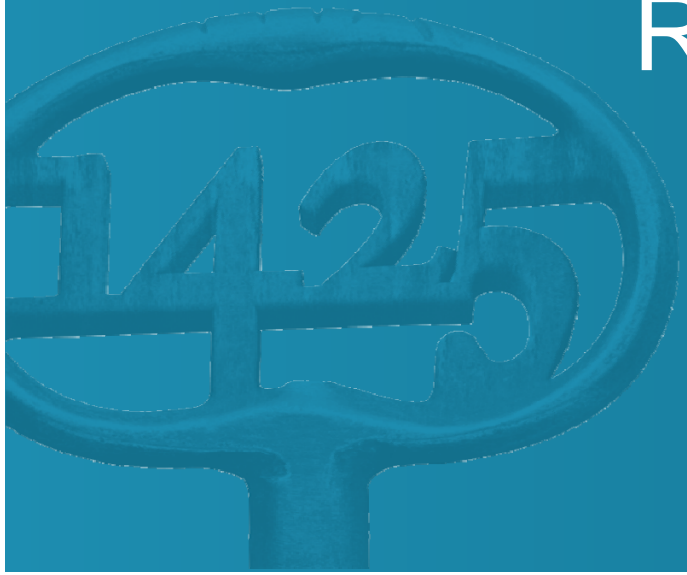


KU LEUVEN

TDD4ES strategy evaluation

	Test on target	Test on host	Remote testing
Resources 	--- Test & Program on target	+++ Test & Program on host	+/- Broker on target
Dependency 	+++ Real drivers	--- Only virtual drivers	+++ Real drivers
Slow 	--- Frequent flash programming	+++ Development limited to host	- Frequent flash programming*

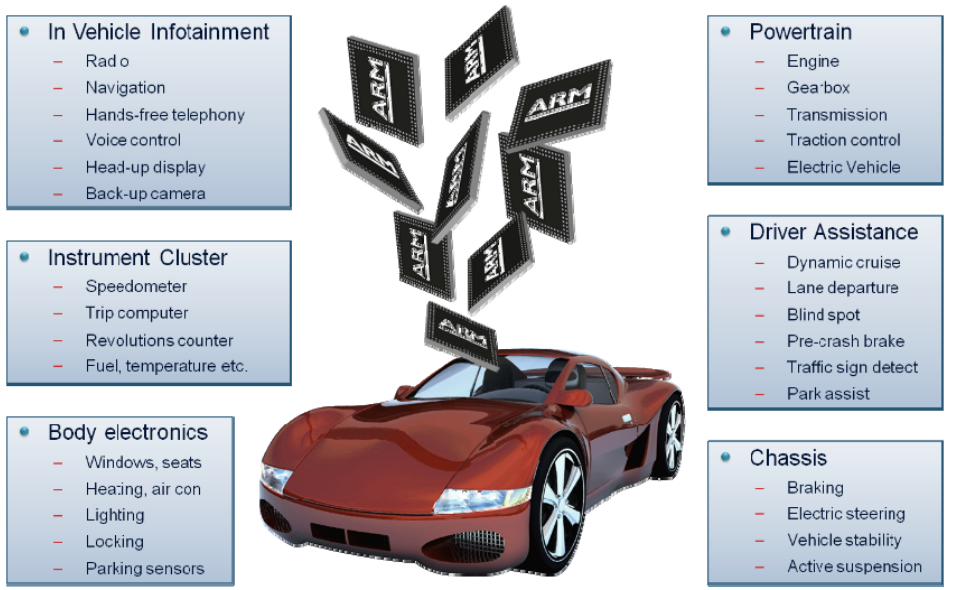
Embedded Software Resilience



KU LEUVEN

Introduction

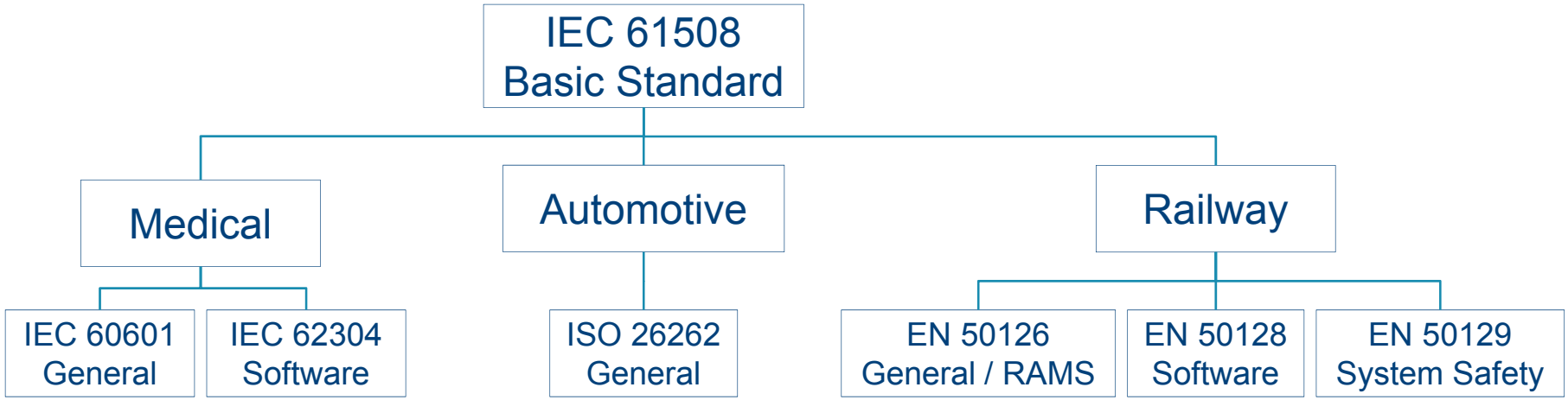
- Embedded systems are used more and more:
  - Home appliances
  - Avionics
  - Railway
  - Automotive
  - Medical
  - …



- Function can be grouped in two main categories
  1. Safety-critical: system can decide life or death
  2. Non-safety-critical: e.g. infotainment systems

Safety-critical

- Safety-critical products must comply with a standard



Functional Safety: IEC 61508

- The standard addresses two types of failures:
  1. Systematic Failures = design failures: these failures are addressed by imposing the Safety Life-Cycle.
     - The Safety Life-Cycle defines a number of steps, which must be followed, documented and verified to develop a compliant product.

  2. Random Hardware Failures: these failures are quantified and categorized

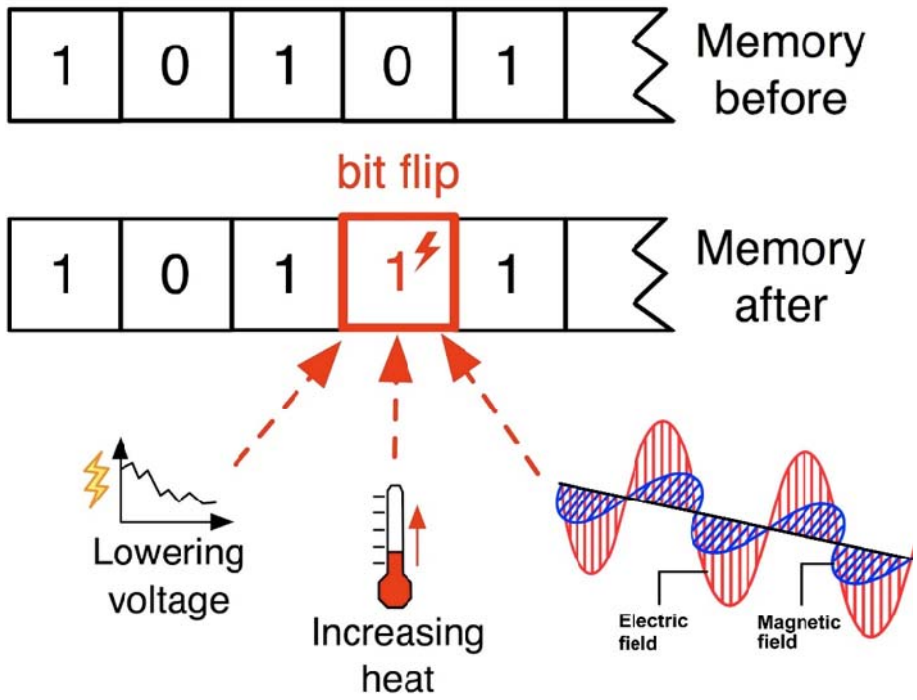
Functional Safety: IEC 61508

- Once all HW failure rates are known, a product can be given a Safety Integrity Level (SIL)
  - High demand = functioning more than 1 p.a.
    e.g. car brakes
  - Low demand = functioning less than 1 p.a.
    e.g. car airbag

SIL	High demand (dangerous failures / hr)	Low demand (probability of failing on demand)
4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 10^{-2}$ to $< 10^{-1}$

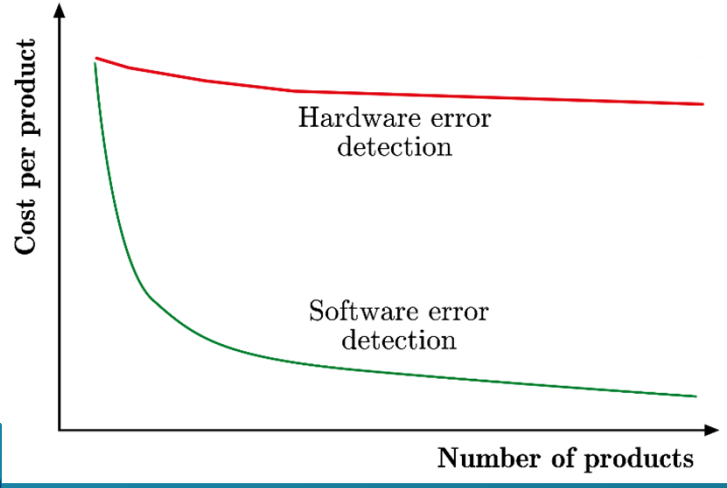
Problem: soft error

- A soft error is a disturbance on hardware level, caused by external factors.
- That disturbance translates to a bit-flip that corrupts memory.
- That corruption can affect the executing software and cause a system crash.



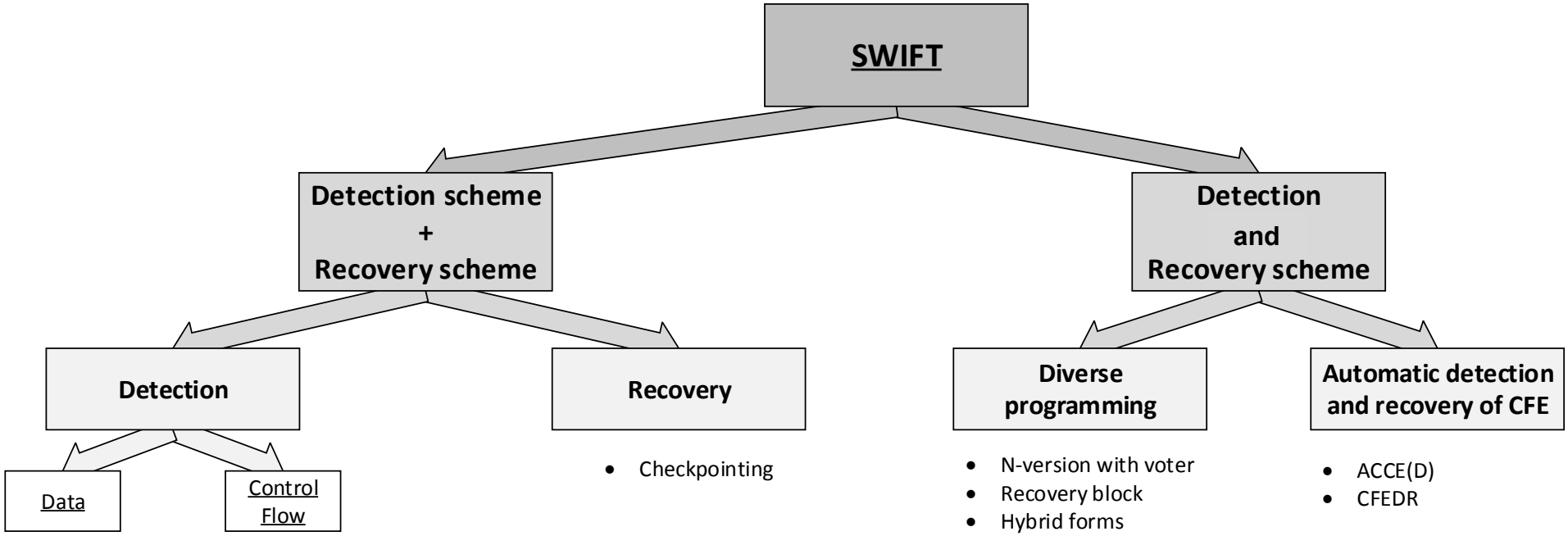
Cost Efficient Detection

Soft Error Detection		
	Hardware-based	Software-based
How	Add components to the PCB so radiation is blocked	Add a detection and recovery scheme
E.g.	Add shield	Implement Signature Monitoring
Cost	Constant extra cost per product	One-time cost only. Same software can be applied to each product



Protecting against Soft Errors: Overview

- The system can be protected against Soft Errors by Software Implemented Fault Tolerance (SWIFT)



Data flow error detection

- Corruption of input, intermediary and output values must be detected.
- This can be done via
  - Duplication and comparison:
    - Values are calculated multiple times and compared to each other.
    - Can be executed at different levels:
      - Variable and Parameter
      - Function…
  - Defensive programming:
    - Inputs and outputs of functions are submitted to reasonability checks
      - Pre- and postconditions

Control flow error detection

- Control Flow Errors (CFE) corrupt the execution order of the program.
  - Skip instructions
  - Re-execute instructions
  - Take wrong branch…

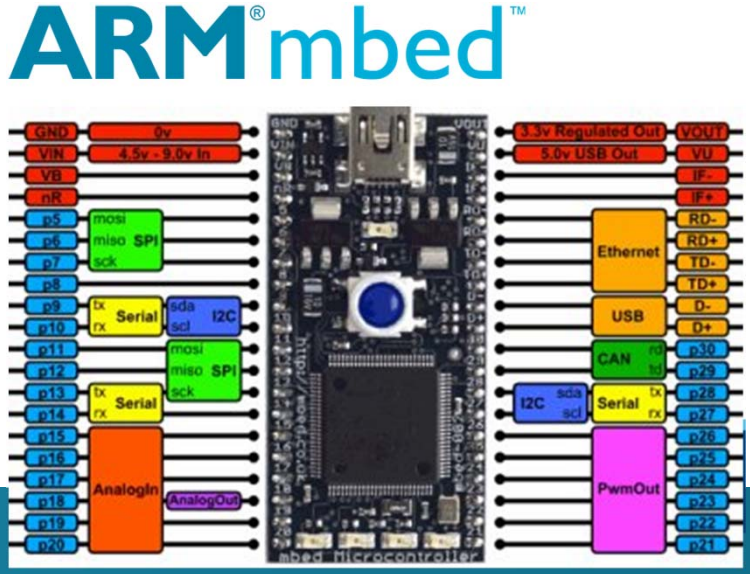
- To detect CFEs, the correct control flow of the program must be known.
  1. Divide program in Basic Blocks
  2. Construct the Control Flow Graph (CFG)

Validating the detection

- No matter what detection technique is used, once implemented it must be evaluated for its detection capability.
- Validating the detection technique means soft errors have to occur in the system. This can be arranged via fault injection.
- Fault Injection is, as the name indicates, the process of injecting, introducing soft errors in a system.

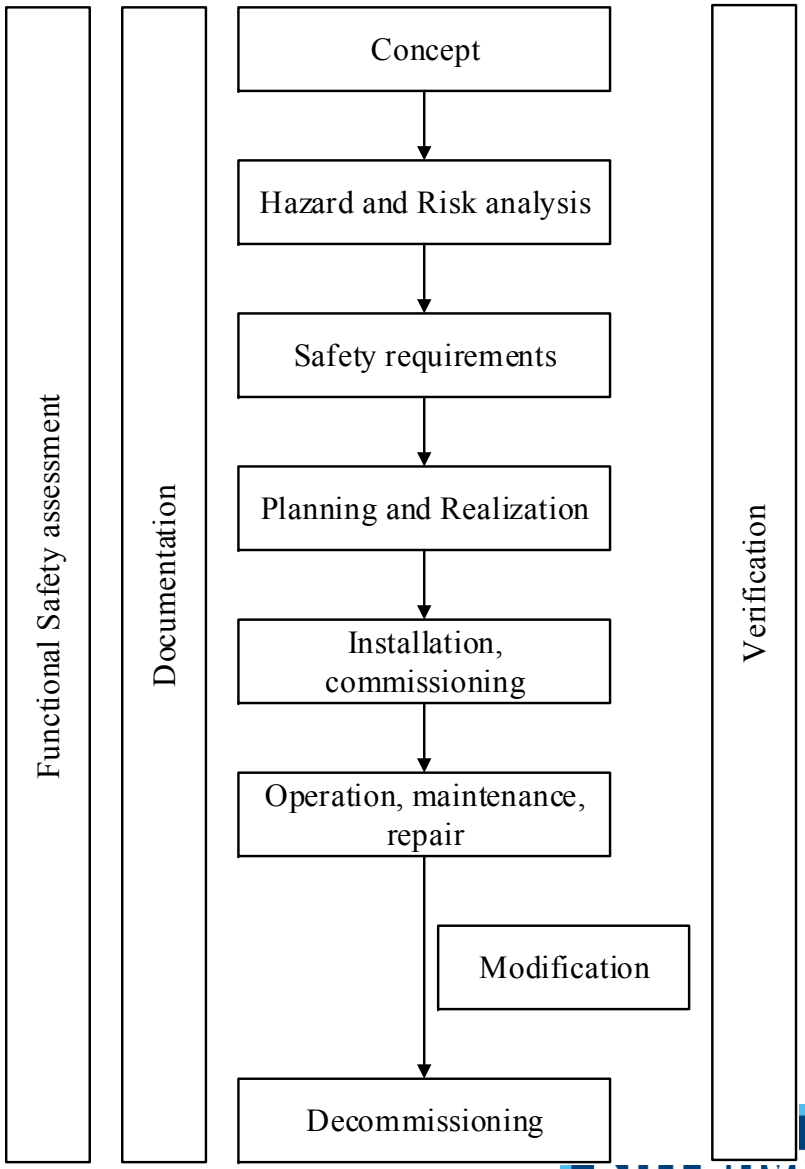
Case Study: Fault Tolerant Pick and Place Robot

- Parts:
  - Lynxmotion AL5B
    5 servo motors fully control the arm.
  - MBED NXP LPC1768
    the microcontroller, driven by Cortex-M3.
  - APDS-9960
    color, proximity and gesture sensor



Safety Life-Cycle

- The Safety Life-Cycle is a development process imposed by IEC 61508 to address Systematic Failures.



Safety Integrity Level (SIL)

- Once a function or system is developed and all protective measures have been implemented, it will have a failure rate.
- IEC 61508 uses SIL to express excellence, using the final failure rate. The range of failure rates is divided into four ranges. The higher the SIL the more stringent the requirements to meet that level.

SIL	Probability of dangerous failure per hour (High demand or Continuous operation)	Probability of failure on demand (low demand operation)
4	$\geq 10^{-9}$ to $< 10^{-8}$	$\geq 10^{-5}$ to $< 10^{-4}$
3	$\geq 10^{-8}$ to $< 10^{-7}$	$\geq 10^{-4}$ to $< 10^{-3}$
2	$\geq 10^{-7}$ to $< 10^{-6}$	$\geq 10^{-3}$ to $< 10^{-2}$
1	$\geq 10^{-6}$ to $< 10^{-5}$	$\geq 10^{-2}$ to $< 10^{-1}$

EN 50128/50129

EN 50129 (System Safety)

Safety Integrity Level	Tolerable Hazard Rate (THR), per hour, per function
4	$10^{-9} \leq THR < 10^{-8}$
3	$10^{-8} \leq THR < 10^{-7}$
2	$10^{-7} \leq THR < 10^{-6}$
1	$10^{-6} \leq THR < 10^{-5}$

EN 50128 (Software)

Software Safety Integrity Level (SSIL)	Description of Safety Integrity
4	Very High
3	High
2	Medium
1	Low
0	Non-Safety Related

Minimum required

- The SSIL is required to be at least the same as the system SIL