Quentin Ochem
AdaCore

Introduction to Certification

Why Certify?

“Regular” software considerations

- Complex software are filled with bugs
  - OS (windows, linux, macosx…)
  - Webservers
  - Office suites
  - Video games
  - …
- And in most cases, bugs are OK
  - Reboot the system
  - Get an updated version
  - Workaround
  - Live with it

“Critical” software considerations

- In certain cases, bugs are not OK
  - They may kill people (aircraft, train, missile, medical devices…)
  - They may lose money (bank…)
  - They may fail to achieve a critical mission (secure a top secret facility…)
- When bugs are detected, it’s often too late
  - People die
  - Money is lost
  - Security is breached

Critical SW Standards

- Every Industry has its own standard
  - "Framework" IEC 61 508
  - Avionics ED 12B / DO 178B
  - Military DEF STAN 00-56
  - Railroad EN 50128
  - Automotive ISO 2626-2
  - Space ECCS-Q-ST-80C
  - Medical Devices IEC 62 304
- They rely on similar principles

Example of Levels of Criticality

- The activities to be performed depend on the SW criticality level
- DO-178C – level A (Catastrophic)
  - Failures will likely cause multiple casualties, or crash the airplane
- DO-178C – level B (Hazardous/Severe)
  - Failure will largely reduce the plane safety margin, or cause casualties to people other than the flight crew
- DO-178C – level C (Major)
  - Failure will significantly reduce the plane safety margin, or cause distress to people other than the flight crew
- DO-178C – level D (Minor)
  - Failure will slightly reduce the plane safety or discomfort to passengers of cabin crew
- DO-178C – Level E (No Effect)
  - Failure will have no effect for safety

Example of objective Organization

- The development is organized through processes
- Each process describes
  - Objectives
  - Activites

Objective	Activity	Applicability				Output	Control Category			
		A	B	C	D		A	B	C	D
Test coverage of Software Structure Is achieved	6.4.4.2.a 6.4.4.2.b 6.4.4.2.d	●	●	○		Software Verification Results	2	2	2	

How to achieve critical SW dev?

- "Just" reasonable development process…
  - Specify requirements
  - Implement only requirements
  - Test
  - Verify tests
  - Reviews
  - Control the development process
- … but now this process is checked and validated
- That's the certification process

Two certification schools

- Certify the process (e.g. DO-178B)
  - We can’t prove how good is the software
  - Let’s show how hard we tried
- Certify the product (e.g. DEF-STAN 00-56)
  - Through “safety cases”
  - Demonstrate absence of identified vulnerabilities

Cost of the certification process

- Certifying is expensive
- Proof must be written for all activities
- The software must be tested entirely with regards to
  - Functionalities
  - Robustness
- All development artifact must be traceable (justifiable, explainable)

Certification authorities

- Certification authorities are responsible for checking that the process is followed
- They're not checking directly the quality of the software
- The applicant and the authorities iterates and discuss various solutions followed to implement the standard
- Things are not fixed – new techniques can be used

Some considerations on critical SW

- The code is smaller and more expensive to write
  - A typical engineer write 1 line of code per day on average
- Not everything can be certified
  - Non-deteministic behaviors are out of the scope
- Not everything needs to be certified
  - On a system, certain parts of the software are critical, others aren't (e.g. entertainment system)

Beware of what's outside your development!

- Is the OS certified?
- Is the Run-Time certified?
- What guarantees on the compiler?
- What guarantees on the tools?
- What else runs on the platform?

Main Certified SW Development Activities

Requirements

- Defines and refines what the system should do
- High Level Requirements (close to the « human » understanding)
- Low Level Requirements (close to the code)
- As of today, this is the part that is the most error prone

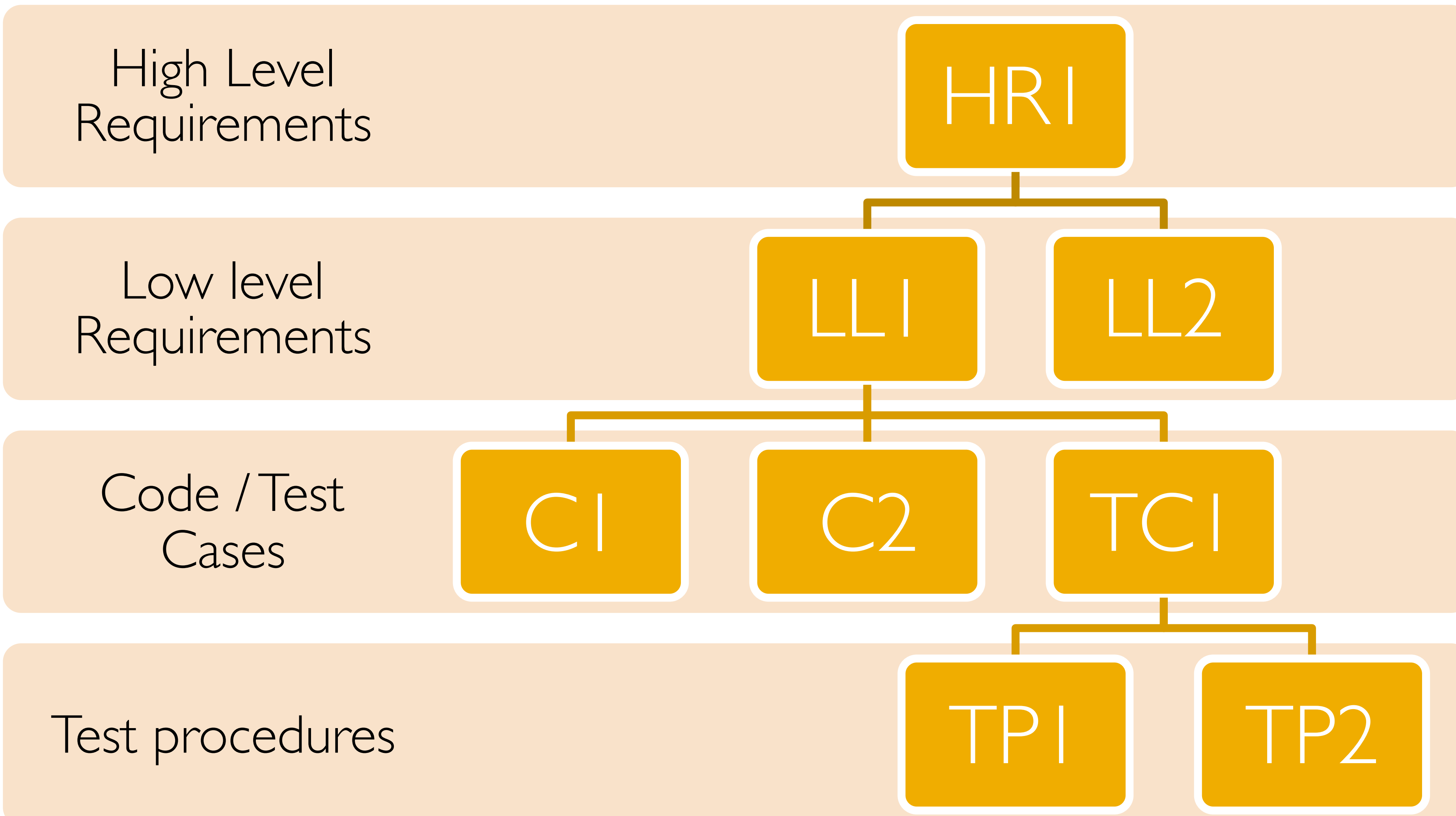
Code

- Implements requirements
- Must be verifiable
- “Easy” part
- Some (very rough) statistics
  - 1 line of code per day per developer
  - 1 line of code per 10 lines of test

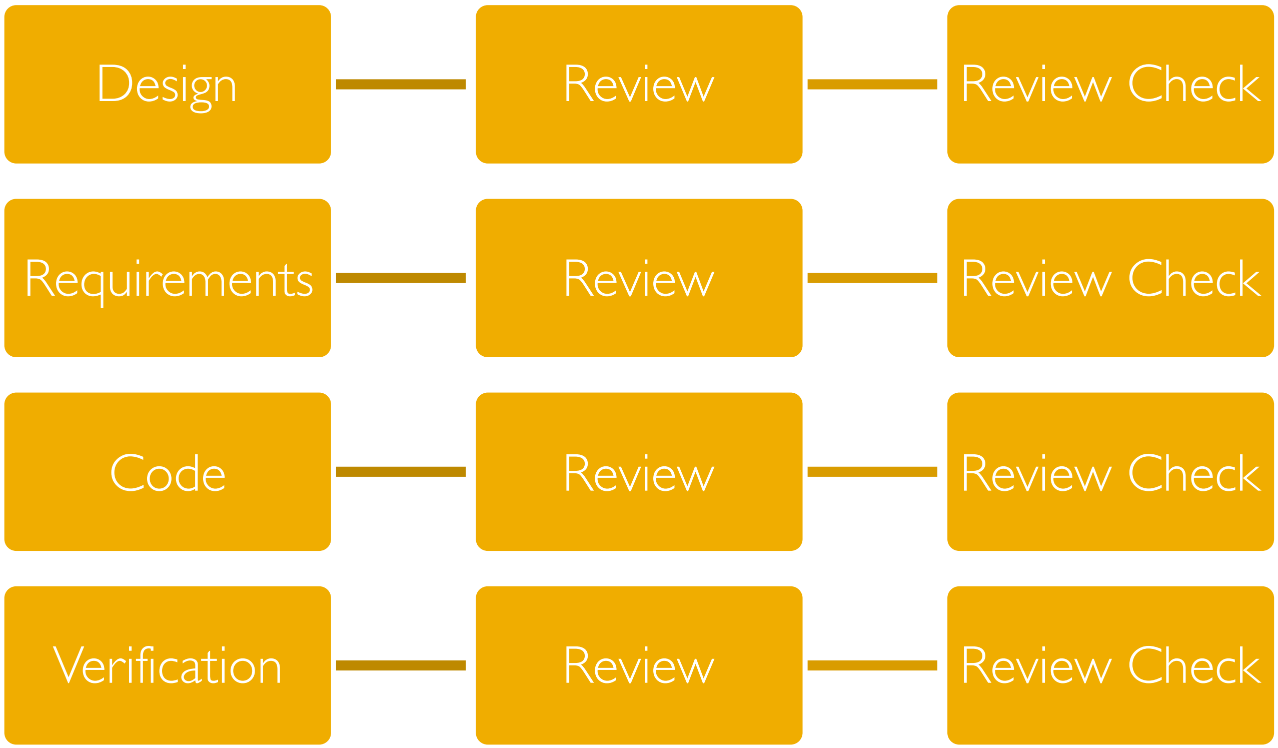
Verification

- Manual Reviews
- Unit and Functional Testing
- Dynamic analysis
- Static analysis

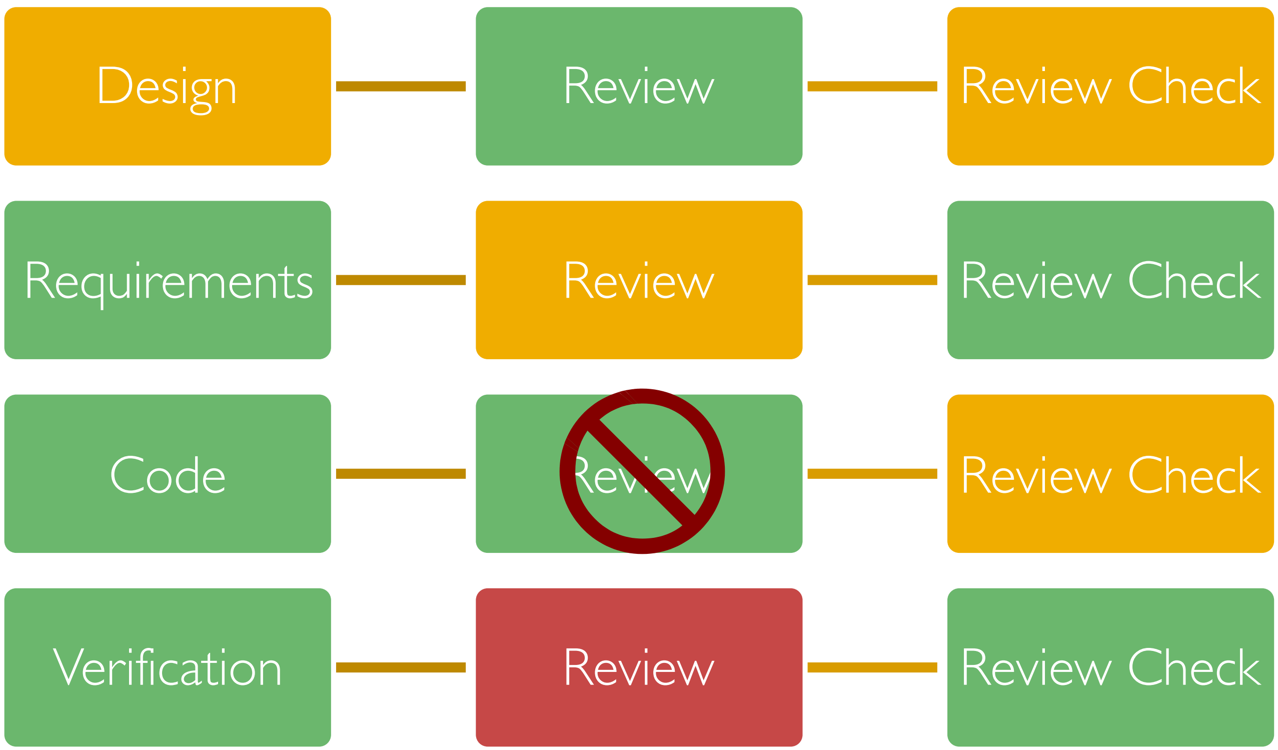
Overall software traceability



Overall review process



Overall independent review process



Examples of Verification Techniques

Testing

- Integration Testing
  - Test the software in the final environment
- Functional Testing - “Black Box”
  - Test high level functionalities
- Unit Testing “White Box”
  - Test software entities without considering the final purpose

Typical Failures to Look For

- “High level errors”
  - Design Errors
  - Algorithmic errors
- “Low level errors”
  - Non-initialized variables
  - Infinite loops
  - Dead code
  - Stack overflow
  - Race conditions
  - Any kind of Run-Time errors (exceptions)

Coverage (1/3)

- How to ensure that all the code is actually tested?
- How to ensure that all the code is testing the requirements?

- Coverage verifications checks that all the code is exercised, and that no unintended function is left

Coverage (2/3)

- Statement Coverage ✓
- Decision Coverage ✓
- Condition Coverage ✓

```
if A = 0 or else B = 0 then
   P1;
else
   null;
end if;
```

Coverage (3/3)

- Coverage by code instrumentation
  - The code tested is not the code deployed
  - Needs memory on the board to store results
- Coverage by target single-stepping
  - Very slow
- Coverage by emulator instrumentation
  - Do not test the real board

Stack Analysis

- Embedded software may have a limited amount of memory

- Need to check the appropriate stack usage
  - By testing (if it crashes, let's double it)
  - By post-mortem introspection (if it's close to crash, let's double it)
  - By static analysis

Static Stack Analysis

- Computes the tree of calls
  - Can’t handle recursively
  - Can’t handle external calls
  - Can’t handle indirect calls
- Computes the size of each frame
  - Can’t handle dynamic frames
- Combine both information for determining the worst stack consumption

Constraints on Timing and Concurrency

Timing issues

- Worst time execution timing must be computed …
- … but is extremely hard to prove
- Done by testing
- Done by model checking
- Requires predictable architecture (no cache, no branch heuristic…)

Concurrency Issues

- Concurrent behavior must be deterministic
- Concurrent programming tends to be non-deterministic
- Needs
  - Modeling technologies
  - Deterministic models (Ravenscar)

Constraints on Language Features

Improve readability

- Constant naming / formatting
- Avoid ambiguous features
- Force comments

Remove / control dynamic structures

- Pointers
- Recursivity
- Indirect calls
- Exceptions

Certain languages are harder to analyze…

```
float * compute (int * tab, int size) {

    float tab2 [size];
    float * result;

    for (int j = 0; j <= size; ++j) {
          tab [j] = tab2 [j] / 10;
    }

    result = tab2;
    return result;
}
```

… than others

```
type Int_Array is array (Integer range <>) of Integer;
type Float_Array is array (Integer range <>) of Float;

function Compute (Tab : Int_Array) return Float_Array is
   Tab2 : Float_Array (Tab'Range);
begin
   for J in Tab'Range loop
      Tab (J) := Tab2 (J) / 10;
   end loop;

   declare
      Result : Float_Array := Tab2;
   begin
      return Result;
   end;
end Compute;
```

Trends

Introduction of New Techniques

- Formal Methods
- Object Orientation
- Modeling
- Outsourcing

Emphasis on Tools

- Cover various areas
  - Static analysis
  - Dynamic analysis
  - Test support
  - Requirement management
  - Traceability management
  - Version control systems
  - Code generators
- Typically two different kind
  - Verification tools
  - Development tools
- Tool Qualification or certification often required

Selection of the Formalism(s)

- Determines the complexity of the tools to write
- Programming languages
  - Ada
  - Java
  - C/C++
- Domain specific languages (DSL)
  - SCADE
  - Simulink
  - MARTE

Conclusion

- Certifying SW is expensive
- … but Certifying SW is necessary
- Tools developed for certification can be pragmatically used for "regular" SW