MÁSTER UNIVERSITARIO EN SISTEMAS ESPACIALES
POR LA UNIVERSIDAD POLITÉCNICA DE MADRID

POLITÉCNICA

Verification and validation of Software Systems: Safety Critical Systems

Alejandro Alonso
alejandro.alonso@upm.es

STRAST

Objectives

- Characterize critical systems and the basic concepts for dealing with safety requirements
- Systematic approach to safety is based on qualification of hazards, risks and assignment of levels of integrity to system components
- Motivate and justify the special requirements of V & V activities for critical software
- Need for following a well-defined product assurance programme

Table of contents

1. Introduction

- As a society, we rely upon software systems
  - Safety systems: fly-by-wire on aircrafts
  - Security systems: protection of digital information
  - Financial systems: cash dispensers
- Then, high-integrity/critical applications should
  - be fully predictable
  - have all the properties required to them
- This can be only provided by a systematic and planned process of assessment
  - This activity can have high costs
  - Which type of assessment has to be made in a system?
- Aim: provide means for users trusting these systems

Introduction

- In order to deal with high integrity/critical systems:
  - Definition of conceptual framework, for better understanding
  - Methods for identification and analysis of potential problematic issues
  - Qualification of the software, according to its criticality
  - Use of process models and means for quality assurance
  - Specific techniques and methods for the development of these systems
- Validation and verification
  - Phases in software development
  - Try to demonstrate that software behaves as expected and according to the specification
  - The system will not fail in normal use

Critical Systems

- Safety-critical systems
  - Failure results in loss of life, injury or damage to the environment;
  - Chemical plant protection system; fly-by-wire system
- Mission-critical systems
  - Failure results in failure of some goal-directed activity
  - Spacecraft navigational system
- Business-critical systems
  - Failure results in high economic losses;
  - Cash dispensers; consumer electronics
- Infrastructure-critical systems
  - Failure implies of loss of infrastructure
  - Electric power distribution, telephone system

Other definitions (ECSS)

- Safety
  - system state where an acceptable level of risk with respect to
    - fatality,
    - injury or occupational illness,
    - damage to launcher hardware or launch site facilities,
    - damage to an element of an interfacing manned flight system,
    - the main functions of a flight system itself,
    - pollution of the environment, atmosphere or outer space,
    - damage to public or private property
  - is not exceeded

2. Safety analysis: Levels of integrity

- Importance of safety in different systems and situations depends on the risks involved
- Differing safety requirements btw projects in terms of the level of risk reduction required
- Safety Integrity:
  - is the likelihood of a safety-related system satisfactorily performing the required safety functions under all de stated conditions within a stated period of time
  - it is expressed as a number of safety integrity levels: community gradually converging to 4 levels

Levels of Integrity

- Example: IEC 61508
- Continuous mode
  - Also called continuous control systems

Safety Integrity Level	Probability of dangerous failure per hour (Continuous mode of operation)
SIL 4	>=10^{-9} to <10^{-8}
SIL 3	>=10^{-8} to <10^{-7}
SIL 2	>=10^{-7} to <10^{-6}
SIL 1	>=10^{-6} to <10^{-5}

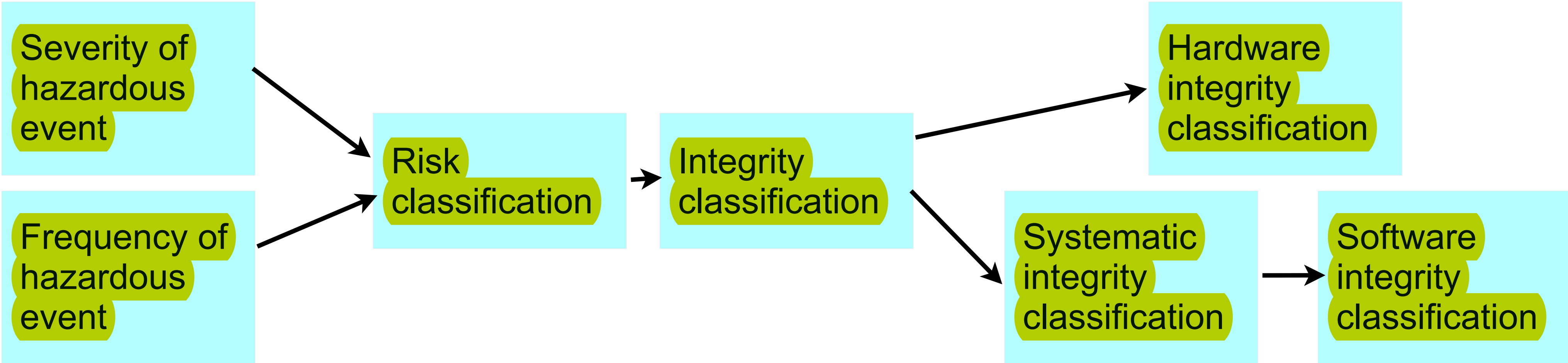
- Demand mode: application that is called upon only when needed.
  - Also called protection systems
  - Probability that the system will fail when called

Safety Integrity Level	Probability of failure on demand, average (Low Demand mode of operation)	Risk Reduction Factor
SIL 4	>=10^{-5} to <10^{-4}	100000 to 10000
SIL 3	>=10^{-4} to <10^{-3}	10000 to 1000
SIL 2	>=10^{-3} to <10^{-2}	1000 to 100
SIL 1	>=10^{-2} to <10^{-1}	100 to 10

Software Criticality Categories-ECSS

Category	Definition
A	Software that if not executed, or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: → Catastrophic consequences
B	Software that if not executed, or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: → Critical consequences
C	Software that if not executed, or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: → Major consequences
D	Software that if not executed, or if not correctly executed, or whose anomalous behaviour can cause or contribute to a system failure resulting in: → Minor or Negligible consequences

Assignment of Integrity Levels

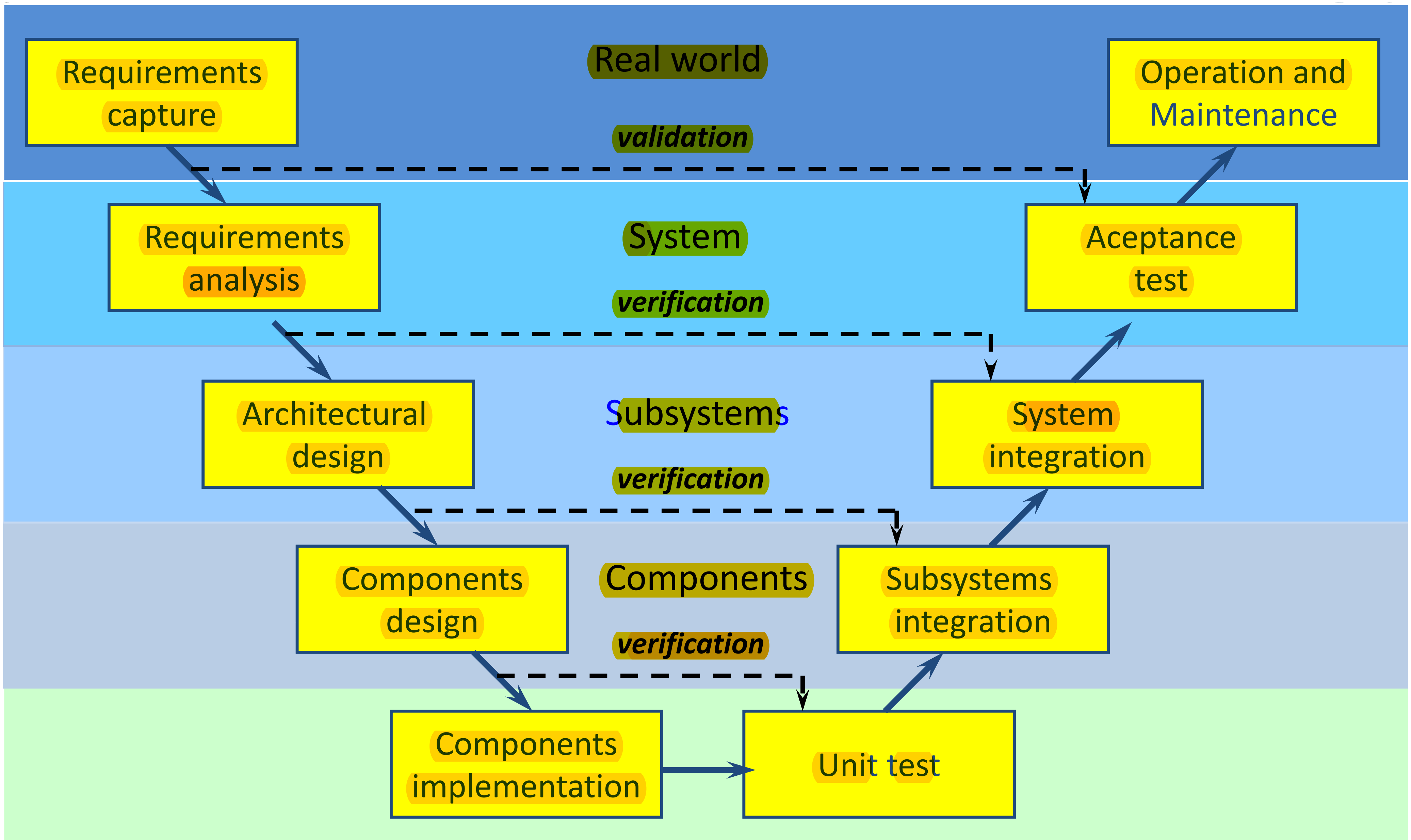


- Software Integrity is related to dangerous software failures
- Integrity levels are assigned in a recursive way.
- This will determine the development methods used and the level of testing performed

3. Verification & Validation

- During and after the implementation process, the system must be checked to ensure it is correct and appropriate
- Verification & Validation
  - Is the name given to this checking and analysis processes
  - Verification: Are we building the product right?
  - Validation: Are we building the right product?
- Goal: establish confidence the system fits for purpose
- The level of required confidence depends on:
  - Software function, user expectations, marketing environment
- V&V of critical systems require additional and more demanding activities

Verification & Validation

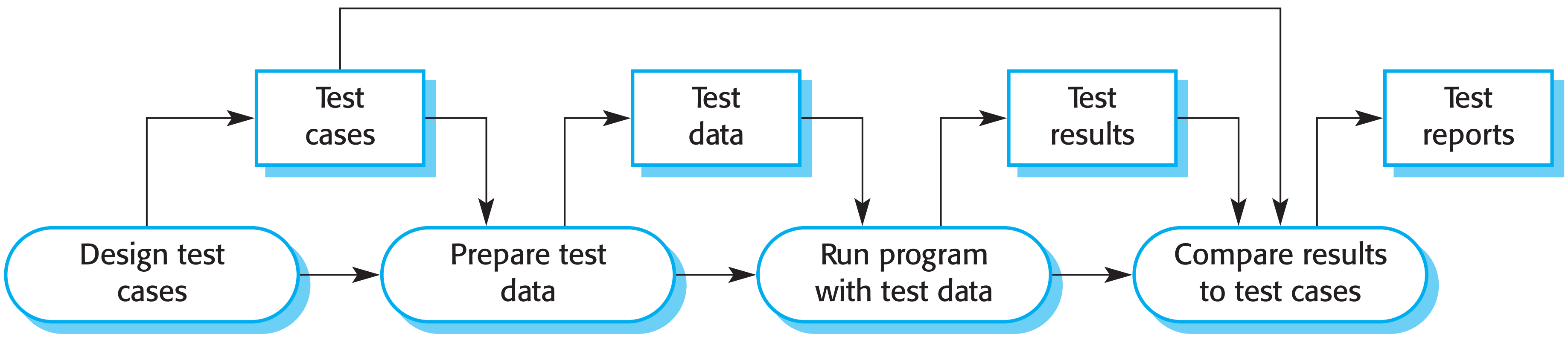


3.1 Validation

- Are we building the right product?
- Validation:
  - Ensure that the system meets the customer's expectations
- Validation (ECSS)
  - confirmation, through the provision of objective evidence that the requirements for a specific intended use or application have been fulfilled
  - The term "validated" is used to designate the corresponding status.
  - The use conditions for validation can be real or simulated.

Testing

- Validation is performed by test, in most systems
- The goal of these tests is to ensure that the system behaves according to requirements
- Testing activities affects the whole lifecycle



Testing

- Only exhaustive testing can show a program is free from defects.
  - However, **exhaustive testing is impossible** in complex systems

- In critical software, testing has to be more demanding and include a larger set of tests

- The exact required tests **depends on software severity level**
  - A number of coverage testing metrics are required

Testing

- Dynamic analysis tools: used for measuring the quality of the testing activities

- **Metrics of coverage** of the tests, such as:
  - Module Coverage
  - Statement Coverage
  - Loop Coverage
  - Decision Coverage
  - Condition Coverage
  - Basis Path Coverage

Testing - Coverage

- Module: foo is called at least one
- Statement: all instructions are exercised at least one time
- Decision:
  - Tests with condition true and false: foo(1,1), foo(1, 0)
- Condition:
  - Tests for the sub conditions: foo(1, 0), foo(0, 1)
- Loops:
  - In the case, exercise:
    - No times
    - One time
    - Several times

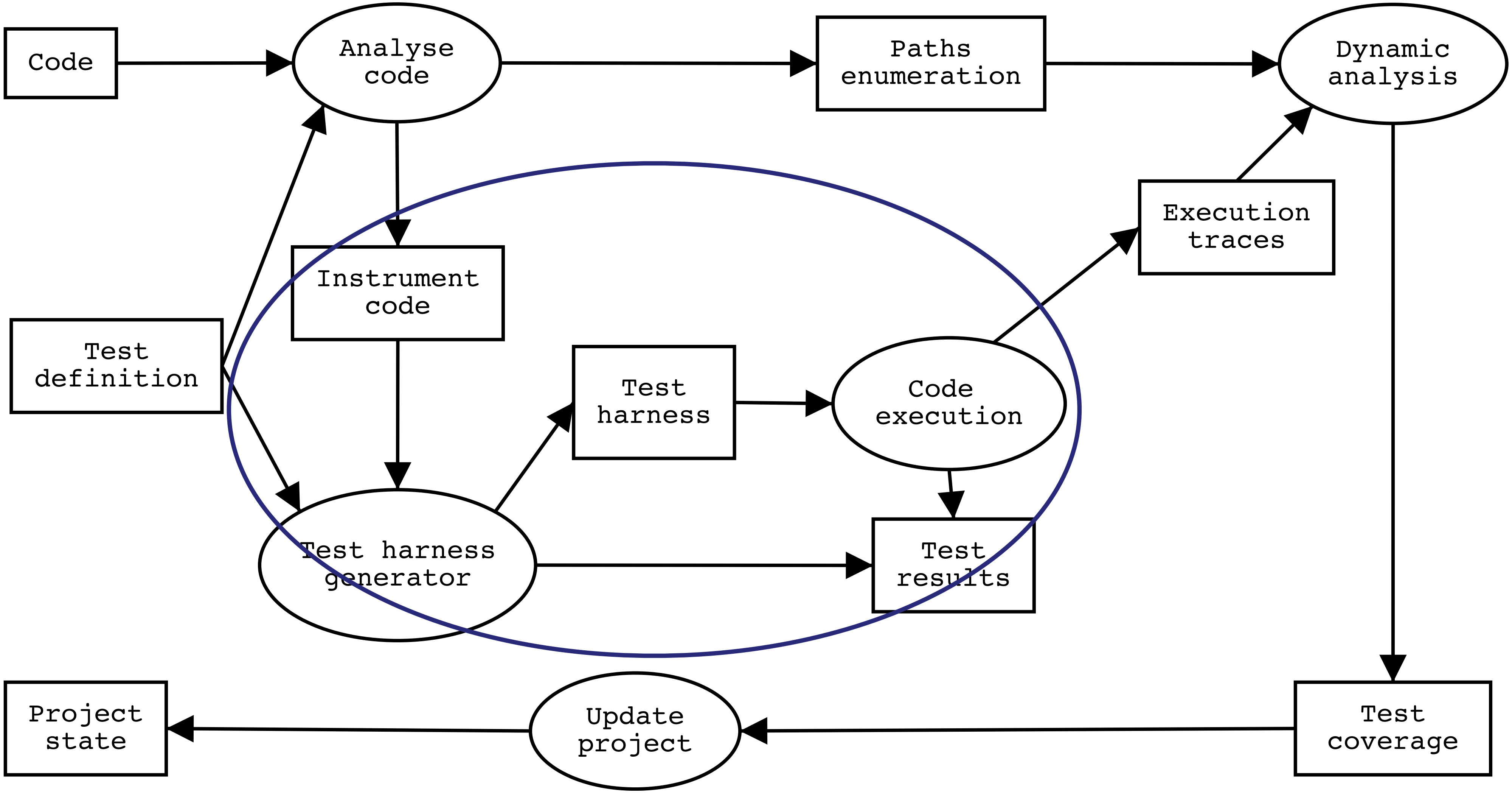
```
int foo (int x, int y)
{
        Probado0 = true
     int z = 0;
     if ((x > 0) && (y > 0))
     {
         Probado1 = true
         z = x;
     }
     return z;
}
```

```
int foo (int x, int y)
{
     int z = x;
     for(int i = 0; i<y, i++)
     {
         z = z + i;
     }
     return z;
}
```

Coverage requirements in ECSS

Code coverage vs criticality	A	B	C	D
Statement coverage	100 %	100 %	Agreed	Agreed
Decision coverage	100 %	100 %	Agreed	Agreed
Modified condition and decision coverage	100 %	Agreed	Agreed	Agreed

Dynamic Analysis Environment



3.2 Verification

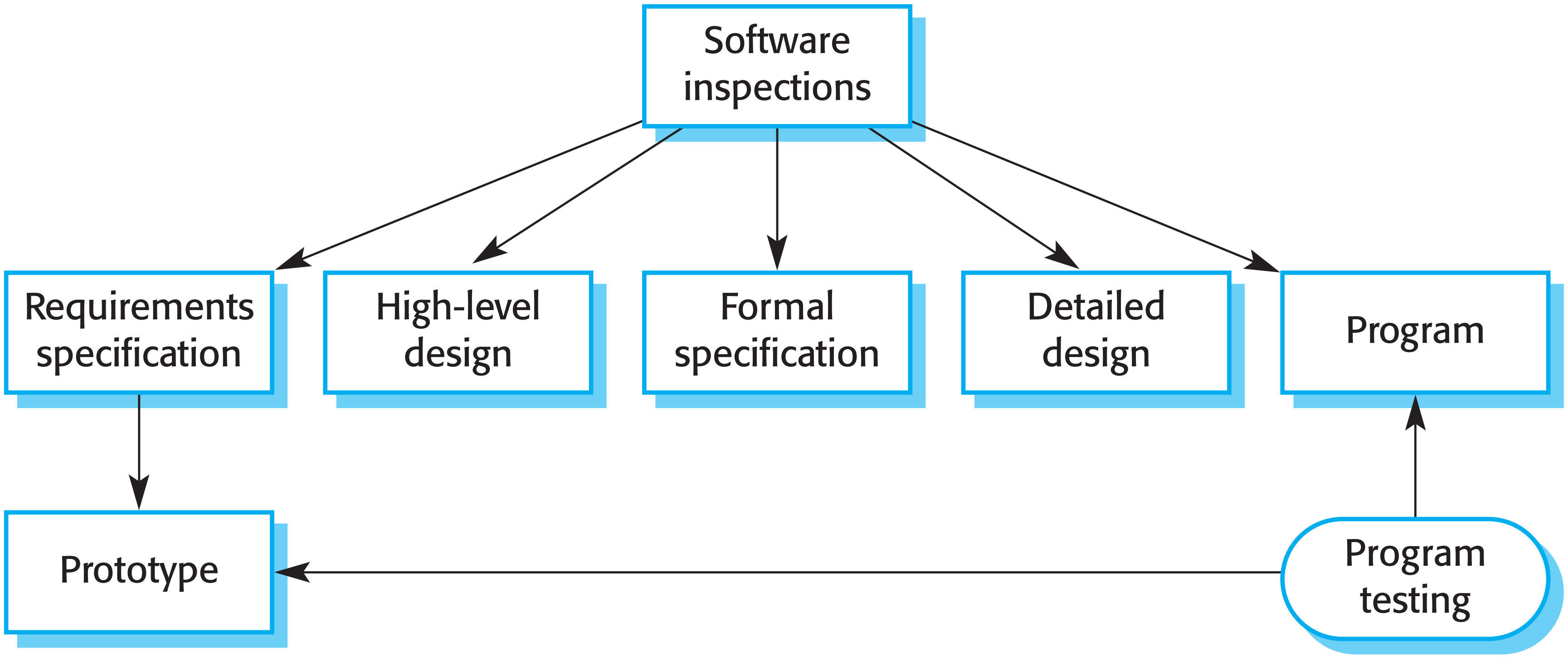
- Ensure that the system conforms to its specification
  - Software artefacts along the development process are correct
  - Appropriate testing has been performed
  - Requirements have been properly handled
- There are a wide range of verification techniques
  - The selection of those to be used in a component or system depends on the required level of confidence
- Verification techniques:
  - Traceability
  - Inspection and review
  - Static analysis
  - Testing

Traceability

- Required to establish that **implementation is complete** and to identify new requirements
- The process models require to provide evidence that requirements are reflected in designs, implementation and testing
- It use to be mandatory to include this information in **documentation**:
  - Which software entities are related with each of the system requirements
  - Plan and perform tests to ensure proper software behaviour with respect to requirements
- Traceability of requirements should be complete

Inspection & reviews

- Analysis and check different system representations:
  - Requirements baseline, architectural design, detailed design, code, tests, risk analysis, etc.



Inspection & reviews

- Sometimes to be done by independent entities
- Some **verifications** to perform:
  - Software requirements are verifiable
  - Designs are feasible
  - Software requirements related with safety, security and criticality are correct
  - Hardware environment constraints are identified
- Verification of **designs**:
  - Design is consistent with previous phases
  - Next phase (detailed design or implementation) is feasible
  - dynamic features are provided & RT choices are justified
  - testing is feasible

Inspection & reviews

- Verification of **code**
  - Implements proper events sequences, consistent interfaces correct data and control flow, appropriate allocation of timing
  - Numerical protection mechanisms, such as code coverage, performance, robustness (resource sharing, division by zero pointers, ...)
  - Following of code standards
- Verification of **unit testing**
  - Tests are consistent with previous phases
  - Traceable to requirements, design and code
  - Test information is under configuration management

Static Analysis

- Analysis of code to assess that it **meets** a number of **proper characteristics**
- Static analysers are software tools for **source text processing**
  - Parse the code to detect potentially erroneous conditions
  - These conditions are to be dealt with by the V & V team
  - Complement inspection
- Specially useful with languages with weak typing
- Required for software with a certain criticality level

Static analysis checks

Fault class	Static analysis check
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Language Safe Subsets

- Some language constructions have **impact** on the use of **static analysis** techniques
- There are language features that **prevents** these tools from **assessing** that code has the desired properties
- Language safe subsets:
  - Define a **set of safe** language **constructs** that are adequate for the development of critical software and
  - **Allows** for static analysis tools to **validate code properties**
- Examples:
  - Misra C,
  - Guide for use of Ada in high integrity systems

3.3 Product Quality Assurance

- ECSS defines a set of requirements for software product assurance (Q-ST-80C)
- The objectives are:
  - Provide adequate confidence to the customer and supplier that the software product satisfies its requirements throughout the system lifetime
- These requirements deal with quality management and framework, life cycle activities and process definition and quality characteristics of products
  - Specific activities are identified for each phase

Software Process Assurance

- Detailed requirements to be followed in the development lifecycle to ensure the development quality
- With respect to **safety** software, defines:
  - How to analyze the system for criticality classification of software products based on the severity of failure consequences
  - Require the supplier to perform a safety analysis and to identify the techniques to be used
  - Apply measures to reduce the risks of the software product
  - Measures to avoid propagation of failures
  - Justify and apply measures to assure the safety critical software

4. Conclusions

- The failure of some systems can cause important losses
- Safety analysis allows for:
  - Identifying the sources of failures
  - Qualifying its consequences, severity and frequency
  - Accepting hazards when risks are acceptable
  - Identifying means for reducing hazards severity or frequency
- Process safety models defines a set of steps to apply these ideas in a systematic and rigorous way
- V & V activities are more demanding when dealing with critical software:
  - Inspection and analysis, coverage metrics, development requirements

Bibliography

- *Safety Critical Computer Systems*, Neil Storey, Addison Wesley, ISBN-10: 0201427877, 1996
- *Software Engineering*, Ian Sommerville, Addison Wesley; 10th ed., 2015
- *Safeware: System Safety and Computers*, Nancy G. Leveson, Addison-Wesley Professional, 1995
- *Practical Reliability Engineering*, P. O'Connor, Wiley, 2002
- ECSS standards:
  - E-ST-40C, Software general requirements
  - Q-ST-80C, Software product assurance
  - Q-ST-40C, Safety
  - Q-ST-40-02C, Hazard Analysis
  - Q-ST-40-12C, Fault tree analysis