GEA Tianjin / 中国民航大学中欧航空工程师学院

CS41: TEST AND SIMULATION

V&V Methods: Test

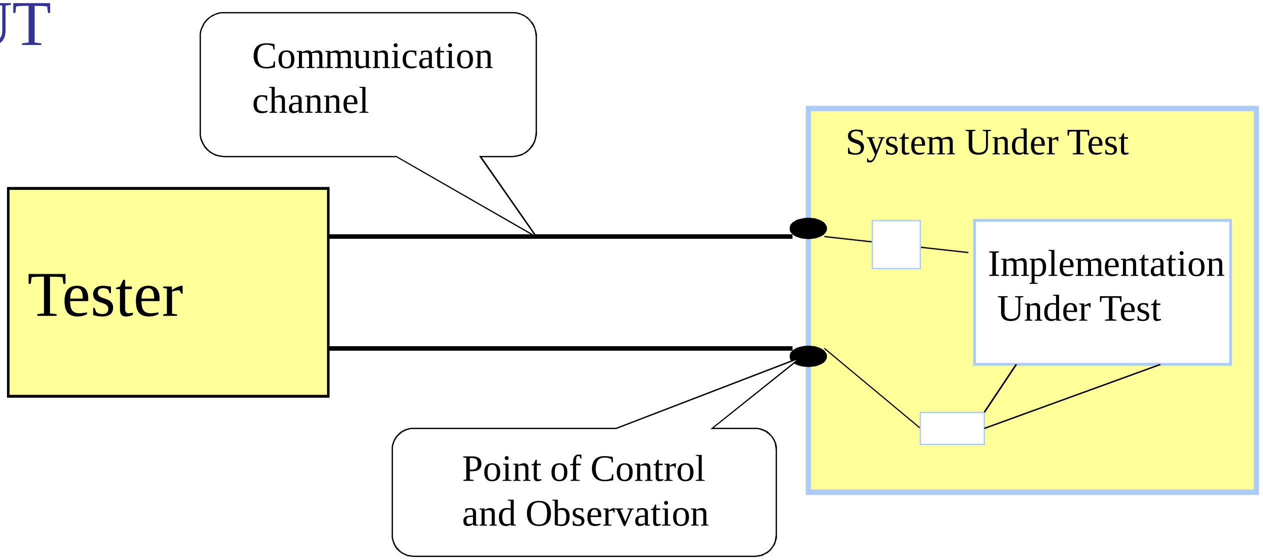
<u>Test</u>: *an action by which the operability, supportability, or performance capability of an item is verified when subjected to controlled conditions that are real or simulated.*

- often use special test equipment or instrumentation to obtain accurate quantitative data for analysis.

- (ARP4754) Provides repeatable evidence of correctness by exercising a system
  - To demonstrate that the implementation performs <u>its intended functions</u>;
  - To provide confidence that the implemented system does not perform <u>unintended functions</u> that impact safety.

Definitions

- System Under Test (SUT): System on which the tests are operated
- Implementation Under Test(IUT): real Target of the Tests
- Point of Control and Observation(PCO): Communication ports defined at System Level, and allowing access to the IUT



Test categories

- **Unit Testing: test the components**
  - Search for defects in system components items
  - Each component is tested separately
  - Can be done by people who developped the component

- **Integration testing: test the buildup of the system**
  - Search for defects in the interfaces between components
  - Performed by assembling the various components into working subparts

- **System testing: Test the system**
  - Exercise the whole system in an environment as close as possible to the final environment

Test categories

- Qualification testing: Test if the system can be used
  - *« To ensure that the system fulfills its specification and that it is ready to be used in the operational environment »* (ISO/IEC 12207)

- Acceptance testing: Assess the system readiness for Deployment/Delivery
  - Includes the customer feedback and agreement

- Alpha testing: Qualification testing in the supplier's sites

- Beta testing: Qualification testing performed by selected potentiel customers, in real conditions

- Non regression testing
  - Selection and re-run of the tests

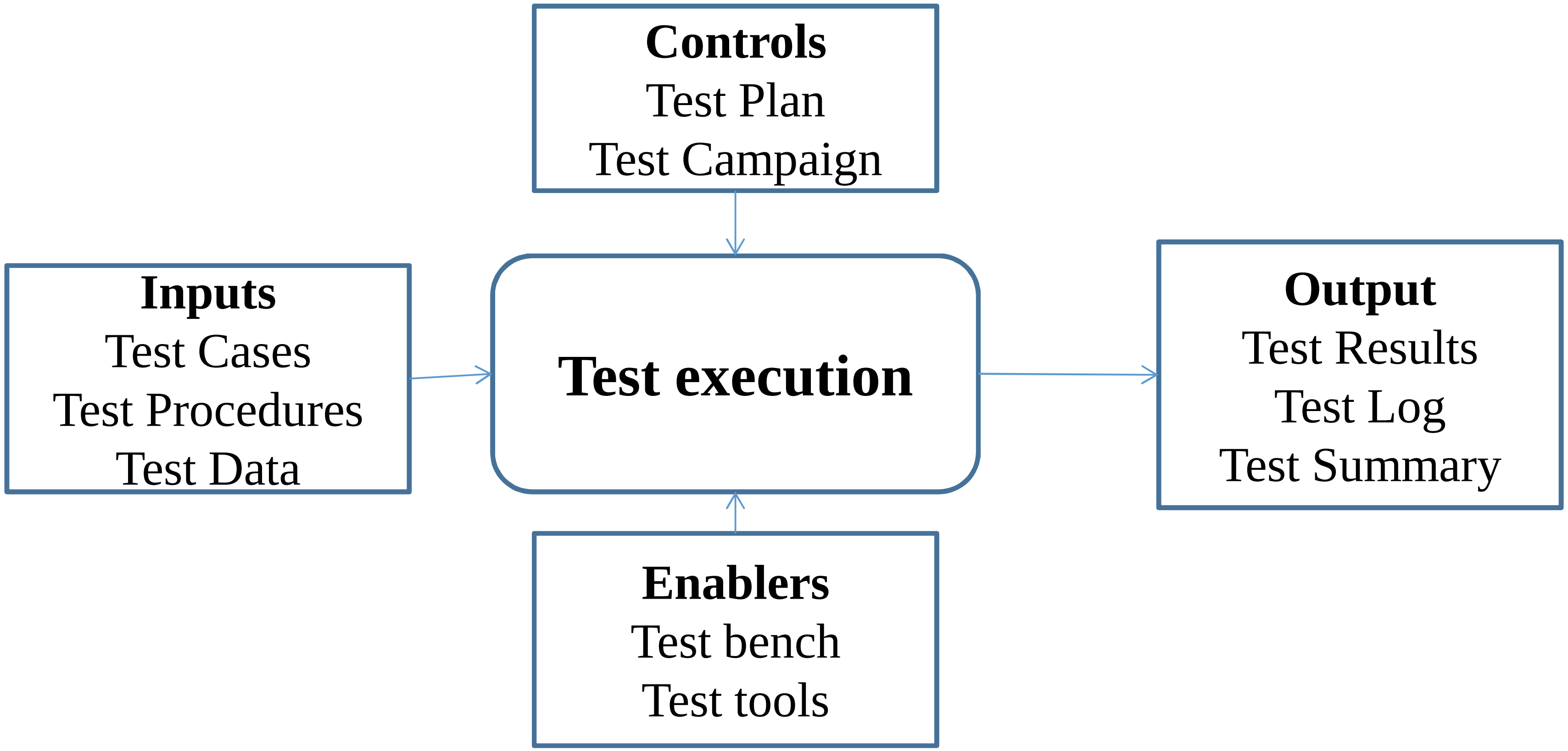
Test items

- **Test plan**: scope, approach, schedule of the testing activities
- **Test case**: identification and description of a test
- **Test objective**: Expression of what should be checked by the test case
- **Test procedure**: sequence of actions/reactions to be performed for a test case
- **Test campaign**: Execution of a selected set of test cases
- **Test result**: result of the execution of a test case

- **Test data**: input data required for the execution of a test case

- **Test log**: output data generated during the execution of a test case
- **Test Summary**: synthesis of all the test results
- **Test means**: test environment and test tools

Testing process



Test coverage

How many tests are good enough ?

- Most common used criteria: **test coverage**
- Coverage ?: percentage of *items* for which at least one test is identified and run
- Items can be:
  - Functions, requirements, use cases → functional coverage
  - Degraded modes, failure recovery
  - Values of variables or input data → domain coverage
  - States or transitions or code → structural coverage
- For software, code coverage is often considered

Functional coverage

- Black-box type of testing

- Functional testing typically involves five steps:
  - The identification of functions that the system is expected to perform
  - The creation of input data based on the function's specifications
  - The determination of output based on the function's specifications
  - The execution of the test case
  - The comparison of actual and expected outputs

Domain coverage

- Domain = range of possible values for a data

- Exhaustive (100%) coverage can be extremely costly
  - Example: Testing a function $Y=X^2$ on a 32 bits hardware → 2^31 tests (68 years of testing time if one test per second …)

- Focus on some key values:
  - Boundaries: min value, max value
  - Equivalence partitioning: one test per partition
    - Example: partition between negative values and positive values
  - Random pick of values, according to some distribution law
  - Values governing decisions (see Decision coverage)

Structural coverage

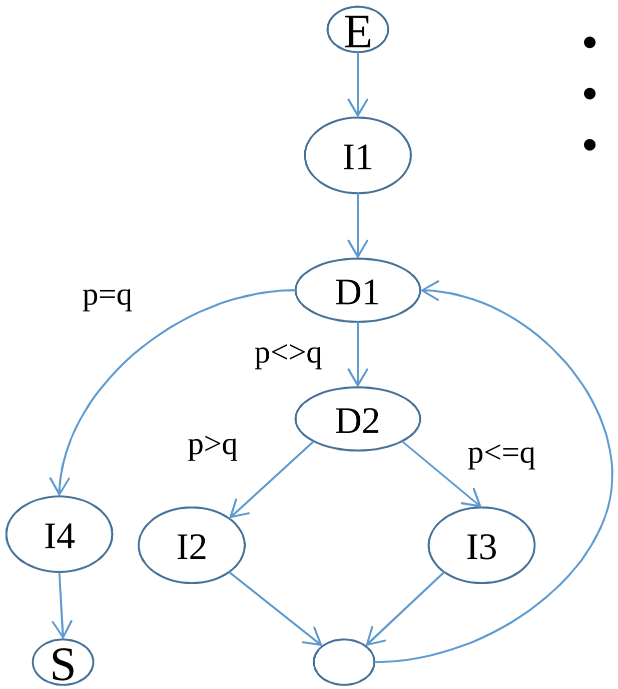
- 3 main types
  - Coverage of the Control Flow graph
  - Coverage of the Data Flow graph
  - Coverage of the Decisions

Control flow graph

```
integer PGCD
{
  integer p,q

  Read(p,q)            I1

  while (p<>q){        D1
    If (p>q)           D2
      p = p-q          I2
    else
      q = q-p          I3
  }
  return p             I4
}
```



- Nodes are statements
- Transitions are execution steps
- 2 special nodes, (E)ntry, (S)top

Decision coverage

- 3 levels of coverage related to Decision

  Example: *if (A and (B or C)) then …*

- **Decision Coverage(DC)**: one test for each possible outcome (true or false)

A	B	C	Result
true	true	false	true
False	False	False	false

→ **2 tests**

Decision coverage

- 3 levels of coverage related to Decision

  Example: *if (A and (B or C)) then …*

- **Condition Coverage(CC):** all possible values for individual conditions

A	B	C	(B or C)	Result
True	True	False	True	True
False	False	False	false	False
False	False	True	True	false

→ 3 tests

Decision coverage

- 3 levels of coverage related to Decision

  Example: *if (A and (B or C)) then …*

- **Multiple Condition/Decision Coverage:** cover all values of conditions which independently affects the result

A	B	C	Result
false	True	True	false
true	True	True	true
True	True	False	True
True	False	False	False
True	False	True	True
True	False	False	false

→ **6 tests**

DO178B

- Aeronautical software certification is governed by DO178B
- Criticity of software is defined from the severity of failure

Failure Condition	Software Level
Catastrophic	Level A
Hazardous/Severe - Major	Level B
Major	Level C
Minor	Level D
No Effect	Level E

Code coverage and certification

One requirement of DO178B is linked to the test coverage

- Level A software: MCDC Testing
- Level B: Decision Coverage (MCDC optional)
- Level C: Statement Coverage

→Level A is much more costly to test

Instrumentation of SUT

- In white box testing, the System Under Test is usually instrumented
  - Instrumentation = integration of test related software or equipment within the system
  - Allows to
    - Observe internal values and behaviour
    - Control the system by forcing internal values or states
    - Measure coverage

Instrumentation

Pros

- Possibility to activate/cover paths impossible to activate by black-box testing
- Detailed log of execution paths, allowing for accurate error solving
- Allow to measure directly all types of coverage

Cons:

Instrumentation is intrusive

- Can modify the behaviour
- Can degrade the performances

Conclusion

- Test is the most common V&V method

- Pros
  - Most representative of real operational conditions
  - Necessary
  - Massive experience (used for mode than 40 years in software)

- Cons
  - Cannot be exhaustive (see Gödel incompleteness theorem)
  - Is performed on a concrete system, done very late
  - *“Testing can prove the presence of bugs, but never their absence* “ (E.W. Dijkstra)

GEA Tianjin / 中国民航大学中欧航空工程师学院

SIMULATION

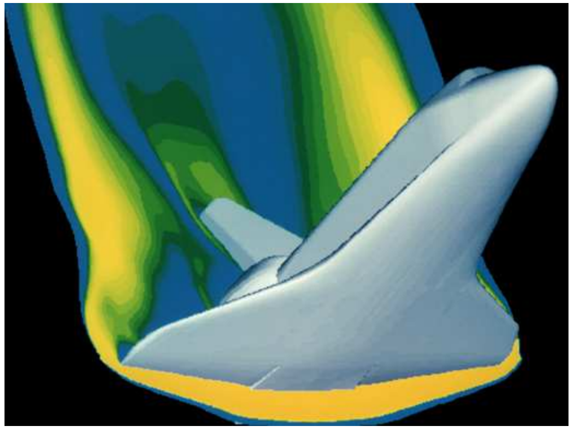
What is simulation ?

- Simulation includes necessarily modelling
  - In some disciplines, they are interchangeable

- A simulation is an exploitation of a model over time or space, in order to illustrate, compute or verify properties of the model.

Simulation example

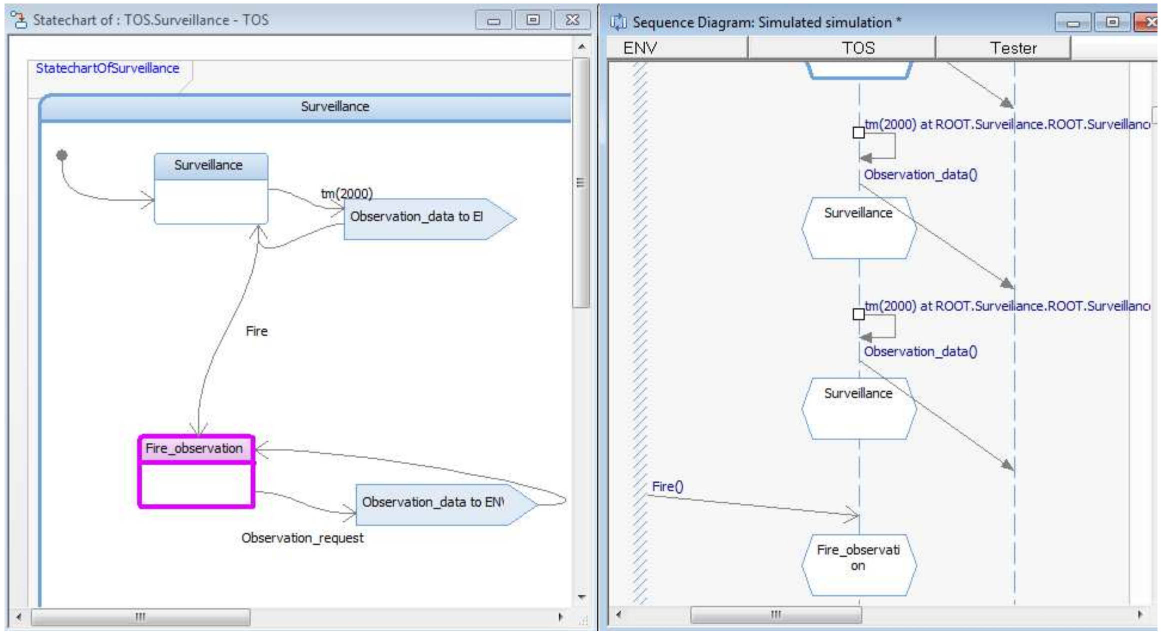
- Computation of general aerodynamical properties from a Computation Fluid Dynamics model



A computer simulation of high velocity air flow around the Space Shuttle during re-entry.

Simulation Example

- ➢ Animation of a behavioral model to show execution paths
  - ❑ Better explain to other people how the model works
  - ❑ Check that intented behaviours exist
  - ❑ Identify causes of wrong behaviours



Simulation strong points

- Pros
  - Can be performed very early in the engineering cycle
  - Can be performed on partial models
  - Efficiency depends mostly on computing power, which improves continuously
  - Applicable to systems unreachable to testing
    - Simulation of satellites, of deep-water systems, of surface conditions on other planets
    - Simulation of nuclear explosion (nuclear testing being forbidden)
    - Simulation of weather events
  - Time can be completely controlled in simulation
    - Simulate faster than testing (real-time), to get quicker and cheaper results
    - Simualte slower than real-time, to understand what is going on

Simulation Improvement points

- Cons
  - Models for simulation usually don’t cover all the real operational conditions
  - The semantics of the models used must be precisely known
  - Multi-domains is still difficult/out-of-reach
  - Simulation of big, complex systems requires big, complex models
  - Some domains are complex to model (e.g. human factors)
  - Performing modelling & simulation requires experts and deeply trained persons

Simulation categories

- Animation
  - Better understanding/Debugging of the model
  - Show behaviours and internal information about the system

- Trace production
  - Production of test procedures skeleton
  - Comparison to reference trace

- Random simulation
  - Exercise the system with random inputs (« Monkey testing »)

Simulation categories

- Systematic simulation (« *Exhaustive*» simulation)
  - Explores all the potential behaviours of the system
  - Search for unwanted situations and bugs
  - Can be done with respect to predefined properties, to check if they are valid in all cases

Formal verification

Formal verification

- Formal verification consists in comparing a mathematically defined model of the system, with properties defined also mathematically

In software engineering, this requires formal languages

- Formal languages= languages with a precise, <u>exhaustive</u> mathematical definition
  - All basic operators and constructions have a precise meaning
  - The combination of operators and constructions is defined mathematically

→A model has a predictable, computable behaviour

Formal languages examples

- Continuous domains
  - B, Z, VDM, Matlab, ...
- Synchronous data-flow
  - Lustre/Scade, Signal, Esterel
- Distributed, parallel systems
  - SDL, LOTOS, Estelle
- Proposals exist for a formal definition of some subsets of UML and SysML
  - Not standardized yet

Key points

- Formal models complement informal specification techniques

- Formal models are precise and unambiguous. They remove areas of doubt in a specification.

- Formal models forces an analysis of the system requirements at an early stage. Correcting errors at this stage is cheaper than modifying a delivered system.

- Formal models are most applicable in the development of critical systems and standards.

Formal verification techniques

➢ Model checking

- Principles:
  - From the system's model, build a graph (states-transitions) from the system's states at execution time
  - Explore the graph,
    - Wanted Properties: compare sequences in the graph with correct sequences

      Example of property: after each fault, there is always an alarm
    - Error finding: by random exploration, and search for deadlocks
- Difficulty: the graph is usually huge ($10^{\wedge 20}$ states), up to infinite
- Example of model checking and corresponding languages
  - ObjectGeode with SDL(now defunct)
  - Some subset of UML/SysML with Ifx:Omega (http://www-if.imag.fr/)
  - UPPAAL (http://www.uppaal.com/)

Formal verification techniques

- Theorem proving
  - Principles:
    - Build a set of mathematical equations equivalent to the system's model
    - Demonstrate with a Theorem Prover that this set of equations respect predefined properties
  - Difficulty: needs strong experts for an application on real systems
  - Many tools, mostly academic (see wikipedia):
    - Coq (http://coq.inria.fr/)