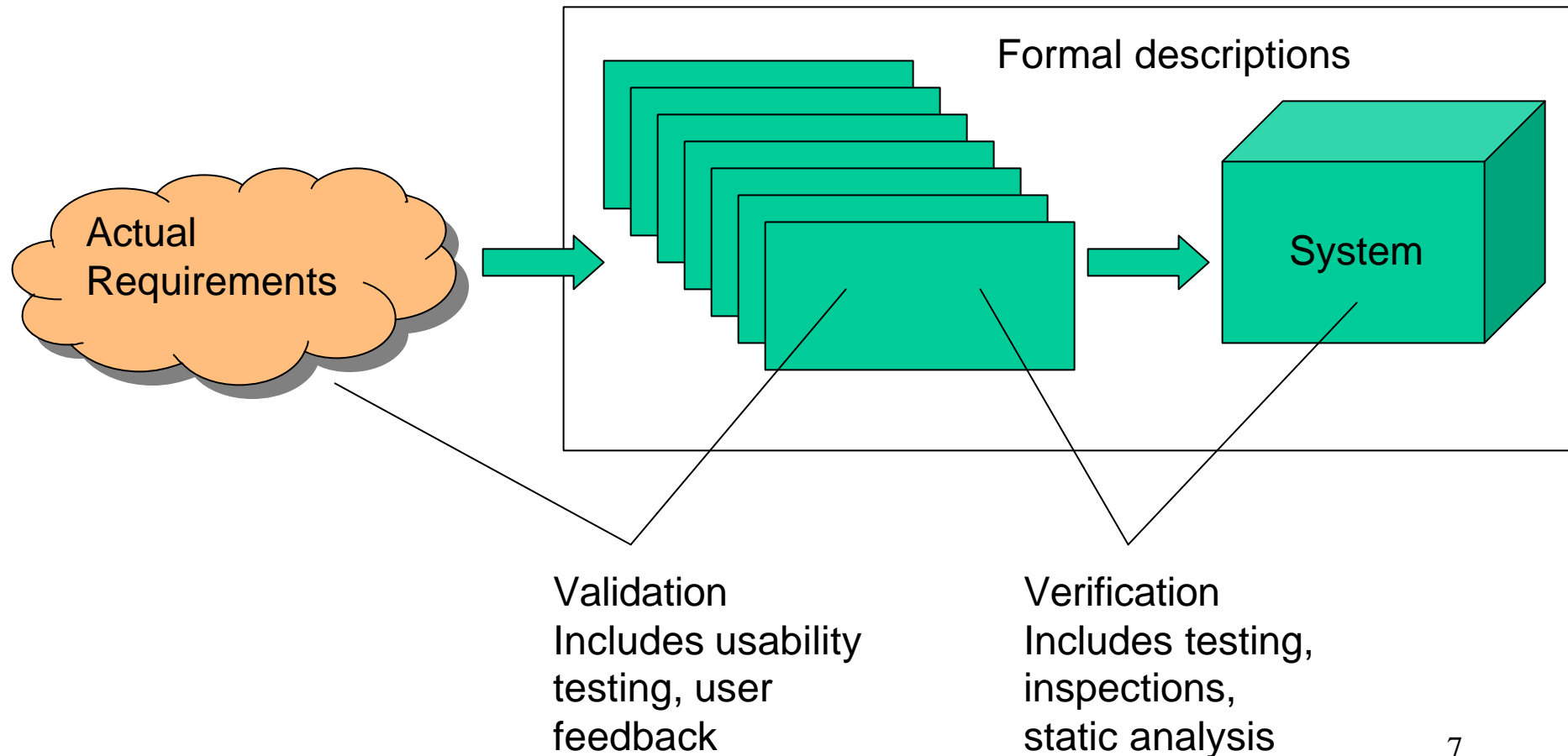


Validation vs. Verification

ABC, "Validation, Verification and Testing of Computer Software." *ACM Computing Surveys*, June 1982.



Verification

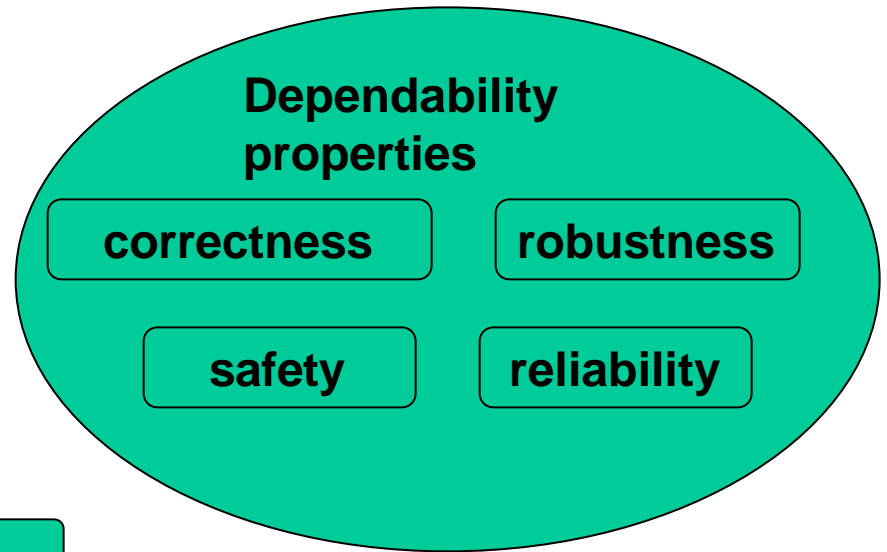
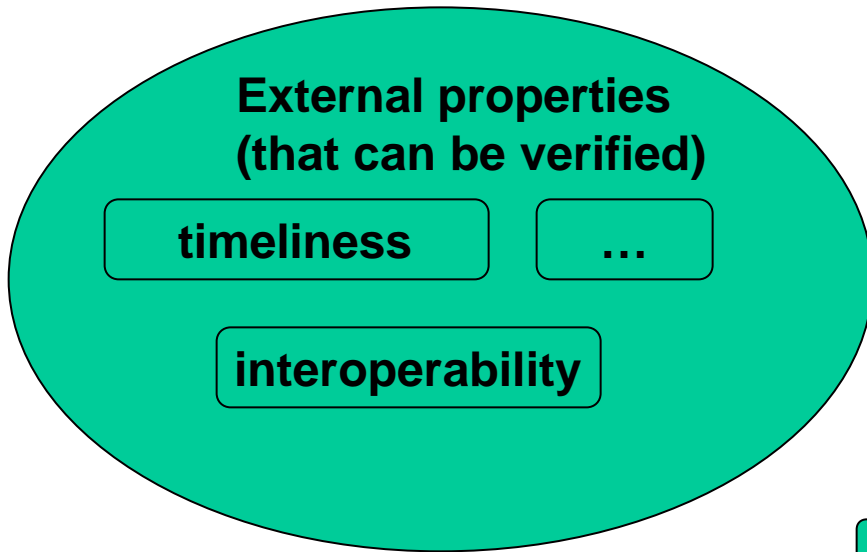
An activity whose primary purpose and effect is to help demonstrate the integrity or self-consistency of the object being verified

- Detect Verification Errors (Defects)
 - Recall definition (“object was built right - Veritas”)
 - Example 1: inconsistent requirements
 - Example 2: design defects found during ROOM/SDL walktroughs (controlled simulations)
 - Example 3: code defects found in code inspection
 - Example 4: Poor code coverage in design test
 - Example 5: integration makes build “insane”

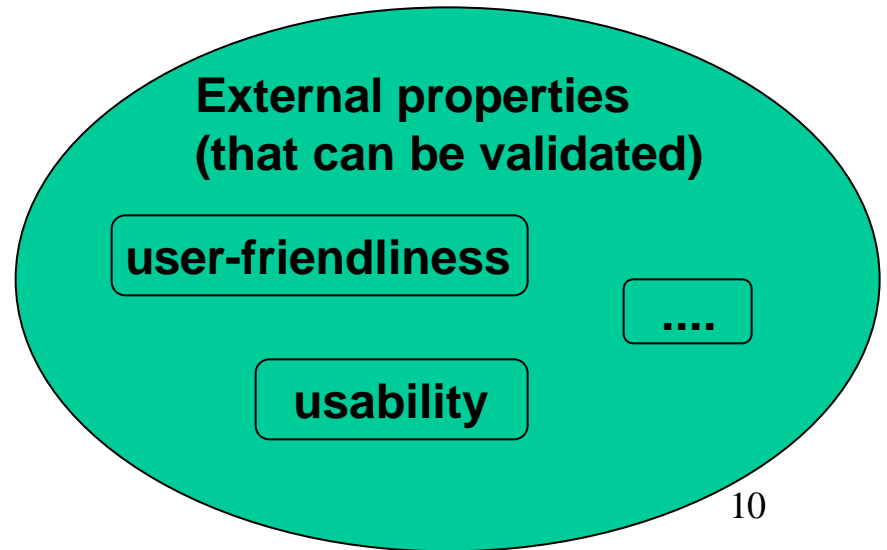
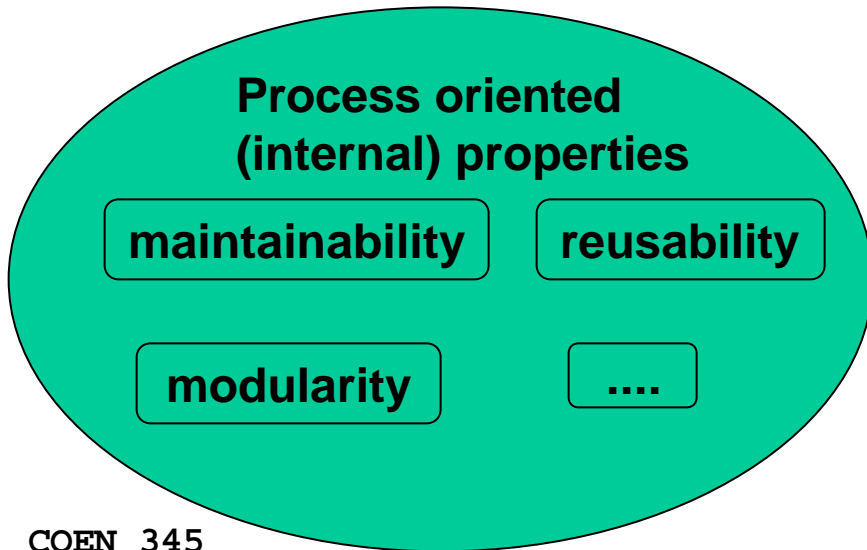
Validation

- Detect validation errors (failures)
 - Example 1: requirements errors (omissions, invalid constraints, erroneous requirements)
 - Example 2: non-conformance of design to key client scenarios, missing high risk exception handling
 - Example 3: functional test violations, improper or inadequate exception handling during function/ stress test
 - Example 4: regression test violations (at product test, system test and release process time)

Software Qualities



....



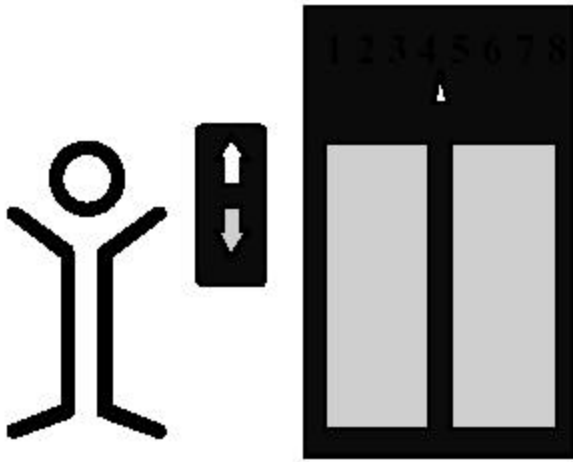
Verification or validation? depends on the property

- Example: elevator response
 - ... if a user press a request button at floor i , an available elevator must arrive at floor I soon...
 - ? this property can be validated, but NOT verified

(SOON is a subjective quantity)

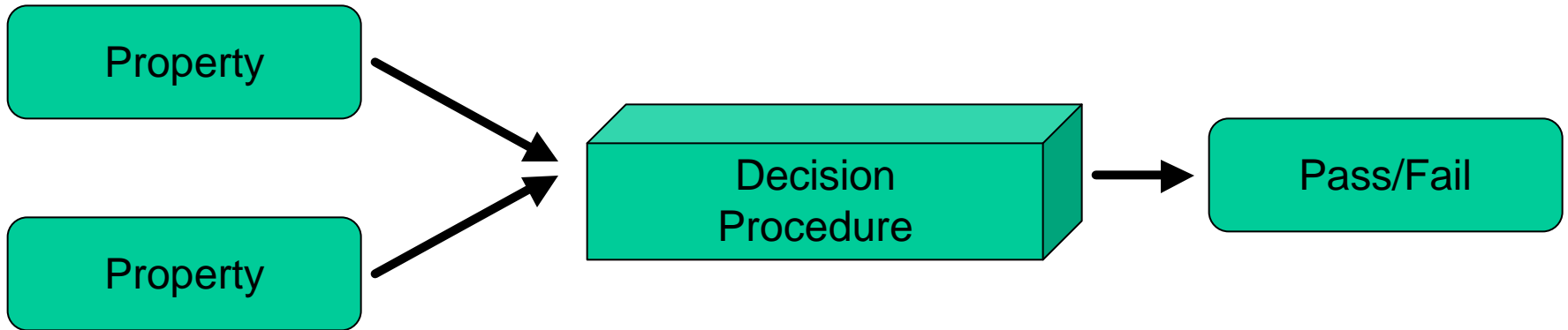
... if a user press a request button at floor i , an available elevator must arrive at floor i within 30 seconds...

- ? this property can be verified
 - (30 seconds is a precise quantity)



ever

You can't always get what you want



Correctness properties are undecidable
the halting problem can be embedded in almost
every property of interest

Why testing and analysis

- Software is never correct no matter which developing technique is used
- Any software must be verified
- Software testing and analysis are
 - important to control the quality of the product (and of the process)
 - very (often too) expensive
 - difficult and stimulating

Impact of software on testing and analysis

- The type of software and its characteristics impact in different ways the testing and analysis activities:
 - different emphasis may be given to the same properties
 - different (new) properties may be required
 - different (new) testing and analysis techniques may be needed

Different emphasis to the same properties

Dependability requirements

- they differ radically between
 - Safety-critical applications
 - flight control systems have strict safety requirements
 - telecommunication systems have strict robustness requirements
 - Mass-market products
 - dependability is less important than time to market
- can vary within the same class of products:
 - reliability and robustness are key issues for multi-user operating systems (e.g., UNIX) less important for single users operating systems (e.g., Windows or MacOS)

Different type of software may require different properties

- Timing properties
 - deadline satisfaction is a key issue for real time systems, but can be irrelevant for other systems
 - performance is important for many applications, but not the main issue for hard-real-time systems
- Synchronization properties
 - absence of deadlock is important for concurrent or distributed systems, not an issue for other systems
- External properties
 - user friendliness is an issue for GUI, irrelevant for embedded controllers

Different properties require different A&T techniques

- Performance can be analyzed using statistical techniques, but deadline satisfaction requires exact computation of execution times
- Reliability can be checked with statistical based testing techniques, correctness can be checked with test selection criteria based on structural coverage (to reveal failures) or weakest precondition computation (to prove the absence of faults)

Different A&T for checking the same properties for different software

- Test selection criteria based on structural coverage are different for
 - procedural software (statement, branch, path,...)
 - object oriented software (coverage of combination of polymorphic calls and dynamic bindings,...)
 - concurrent software (coverage of concurrent execution sequences,...)
 - mobile software (?)
- Absence of deadlock can be statically checked on some systems, require the construction of the reachability space for other systems

Principles

Principles underlying **effective** software testing and analysis techniques include:

- **Sensitivity**: better to fail every time than sometimes
- **Redundancy**: making intentions explicit
- **Partitioning**: divide and conquer
- **Restriction**: making the problem easier
- **Feedback**: tuning the development process

Sensitivity:

better to fail every time than sometimes

- Consistency helps:
 - a test selection criterion works better if every selected test provides the same result, i.e., if the program fails with one of the selected tests, it fails with all of them (reliable criteria)
 - run time deadlock analysis works better if it is machine independent, i.e., if the program deadlocks when analyzed on one machine, it deadlocks on every machine

Redundancy: making intentions explicit

- Redundant checks can increase the capabilities of catching specific faults early or more efficiently.
 - Static type checking is redundant with respect to dynamic type checking, but it can reveal many type mismatches earlier and more efficiently.
 - Validation of requirements is redundant with respect to validation of final software, but can reveal errors earlier and more efficiently.
 - Testing and proof of properties are redundant, but are often used together to increase confidence

Partitioning: divide and conquer

- Hard testing and verification problems can be handled by suitably partitioning the input space:
 - both structural and functional test selection criteria identify suitable partitions of code or specifications (partitions drive the sampling of the input space)
 - verification techniques fold the input space according to specific characteristics, thus grouping homogeneous data together and determining partitions

Restriction.

making the problem easier

- Suitable restrictions can reduce hard (unsolvable) problems to simpler (solvable) problems
 - A weaker spec may be easier to check: it is impossible (in general) to show that pointers are used correctly, but the simple Java requirement that pointers are initialized before use is simple to enforce.
 - A stronger spec may be easier to check: it is impossible (in general) to show that type errors do not occur at run-time in a dynamically typed language, but statically typed languages impose stronger restrictions that are easily checkable.

Feedback: tuning the process

- Learning from experience:
 - checklists are built on the basis of errors revealed in the past
 - error taxonomies can help in building better test selection criteria

Goals of Testing

Goodenough & Gerhart, "Toward a Theory of Test Data Selection." IEEE TSE, Jan 1985.

- Find faults (“Debug” Testing):
 - a test is successful if the program fails
- Provide confidence (Acceptance Testing)
 - of reliability
 - of (probable) correctness
 - of detection (therefore absence) of particular faults

Goals of Analysis

- Formal proof of software properties
 - restrict properties (“easier” properties) or programs (“structured” programs) to allow algorithmic proof
 - data flow analysis
 - necessary | sufficient conditions
 - compromise between
 - accuracy of the property
 - generality of the program to analyze
 - complexity of the analysis
 - accuracy of the result

Testing and Analysis are Creative

- Testing and analysis are important, difficult, and stimulating
 - Good testing requires as much skill and creativity as good design, because testing *is* design
- Testers should be chosen from the most talented employees
 - It is a competitive advantage to produce a high-quality product at acceptable, predictable cost
- Design the product and process for test
 - The process: for visibility, improvement
 - The product: for testability at every stage