Defect testing

testing per misurare dependability

Testing programs to establish the presence of system defects

Various kinds of testing

- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress
- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation
- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

(Defect) testing
- cos’è e cosa non è -

Defect testing

discrepanza di comportamento rispetto a quello previsto

- The goal of defect testing is to **discover defects** in programs: undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption, in other terms all possible failures
- **Can reveal the presence of failures and NOT their absence**

↳ l'attività del TESTING ha successo quando trovo errori
e se non trovo niente → o non ci sono bug (poco probabile) => non può garantire l'assenza di bug
→ o non sono riuscito a trovarli

- A successful test is a test which causes a program to behave in an anomalous way and which discovers one or more failures
- The **only** V & V technique for **non-functional** requirements (es. reliability, velocità)
- Should be used **in conjunction** with static verification to provide full V&V coverage

↳ il TESTING è dinamico

2 tipi di Testing a confronto
Defect T. vs. Statistical T.

- Defect testing → scoprire difetti
  - Tests designed to **discover** system defects.
  - A successful defect test is one which reveals the presence of defects in a system.

- Statistical testing →
  - tests designed to reflect the frequency of user inputs. Used for reliability **estimation**.

2 tipi di Testing a confronto
Defect T. vs. Statistical T.

- Defect testing

  casi di utilizzo ≠ INPUT specifici

  - Si progettano i test-case
  - Si eseguono i test e si verificano quali provocano quali failure
  - Si passano i dati raccolti sulle failure a colui che fa il debugging e corregge il codice per evitare l'insorgenza delle failure rilevate

- Statistical testing
  - Si definisce un profio operativo ed un ambiante d'esecuzio
  - Poi si esegue e si rilevano le misure di interesse (tempi, no. failure, ecc).
  - Poi si calcolano le metriche di interesse (MTBF, Avail, ROCOF, …)

2 tipi di Testing a confronto
Defect testing vs. validation testing

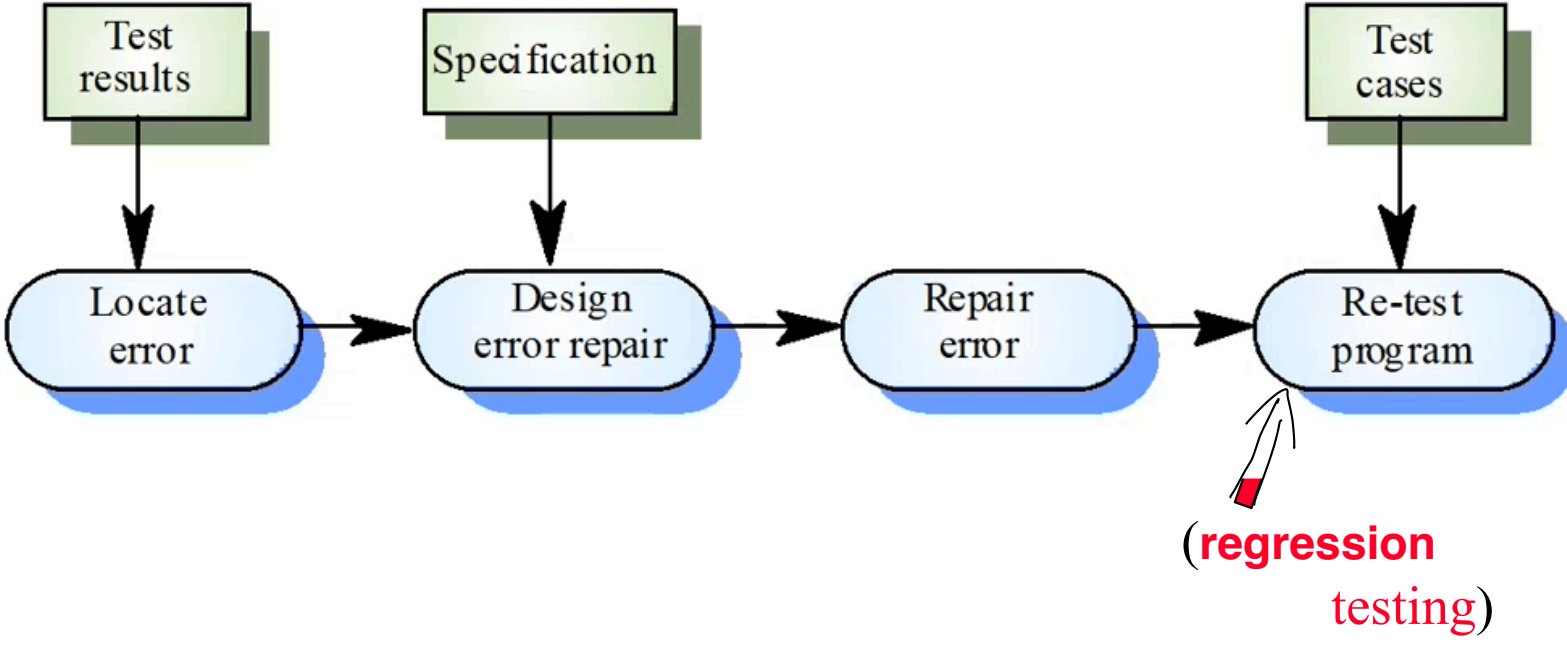
utilizzo il testing per fare validazione

- **Validation** testing (tecnica per la Validazione)
  - To demonstrate to the developer and the system customer that the software meets its requirements;
  - A successful test shows that the system operates as intended.
- **Defect** testing (tecnica per la Verifica)
  - To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification;
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

Testing ≠ debugging

- Defect testing and debugging are **distinct** processes
- Verification and validation is concerned with establishing the existence of defects in a program
- Debugging is concerned with locating and repairing these errors
- Debugging involves (i) formulating a hypothesis about program behaviour then (ii) testing these hypotheses to find the system error

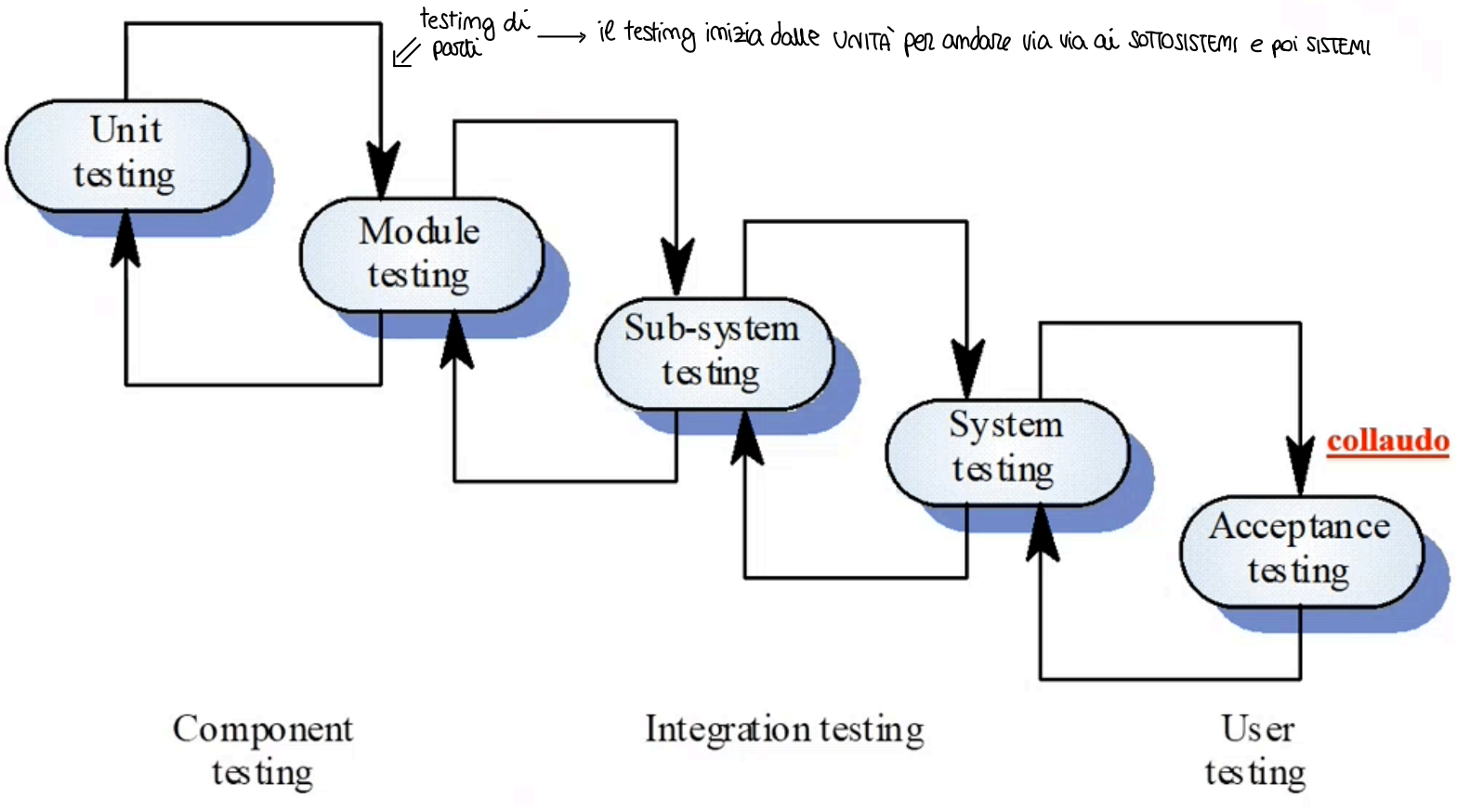
The debugging process



(Defect) Testing
quando e organizzato come

Le attività di V & V si fa il prima possibile, non quando finisco ie codice

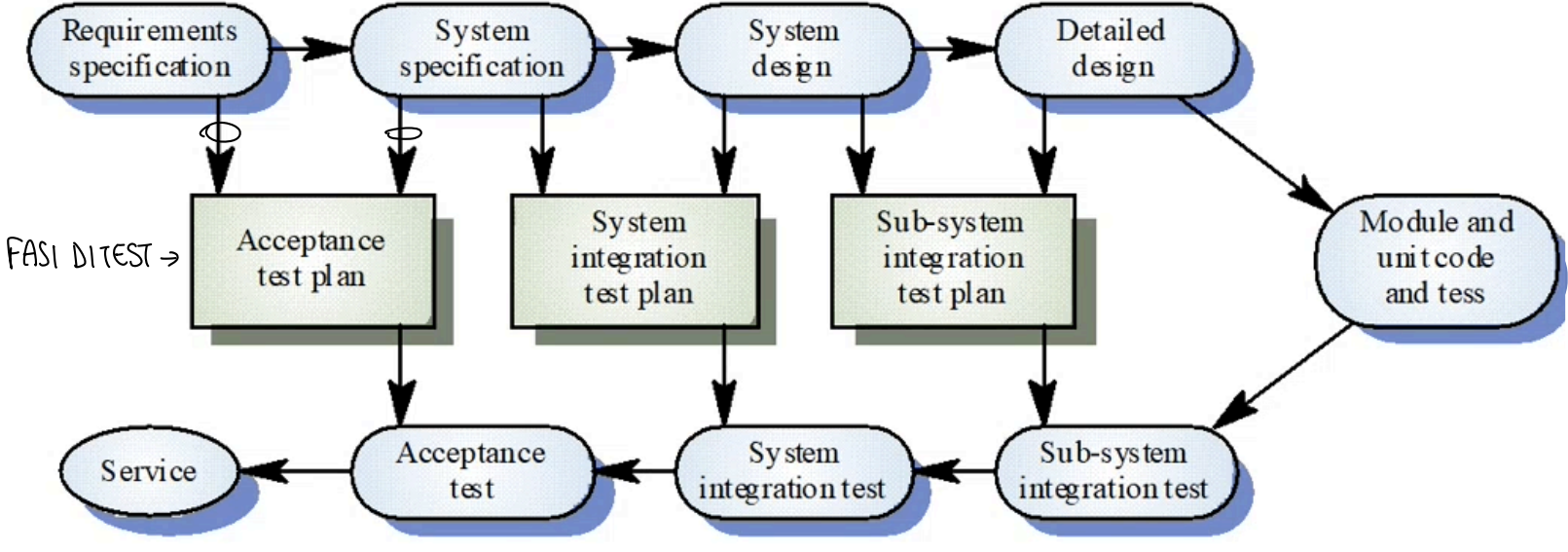
Modello dei processi del testing



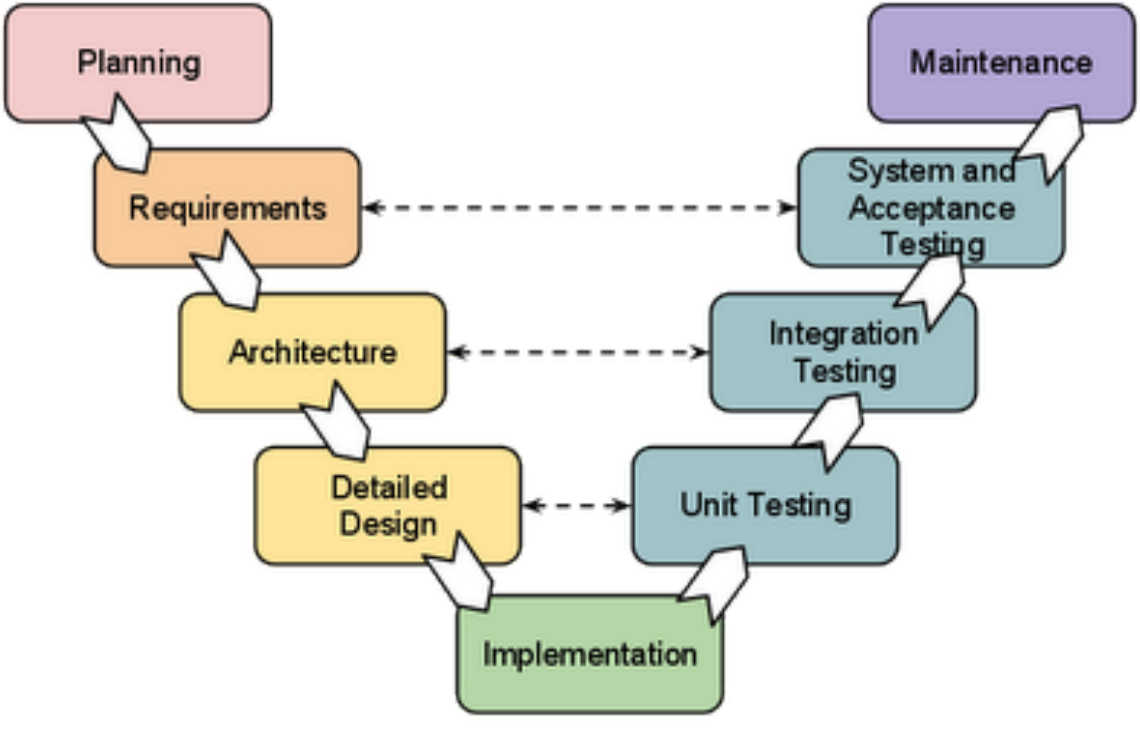
Testing stages

- Unit testing
  - Individual components are tested
- Module testing
  - Related collections of dependent components are tested
- Sub-system testing
  - Modules are integrated into sub-systems and tested. The focus here should be on interface testing
- System testing
  - Testing of the system as a whole. Testing of emergent properties
- Acceptance testing (Collaudo)
  - Testing with customer data to check that it is acceptable

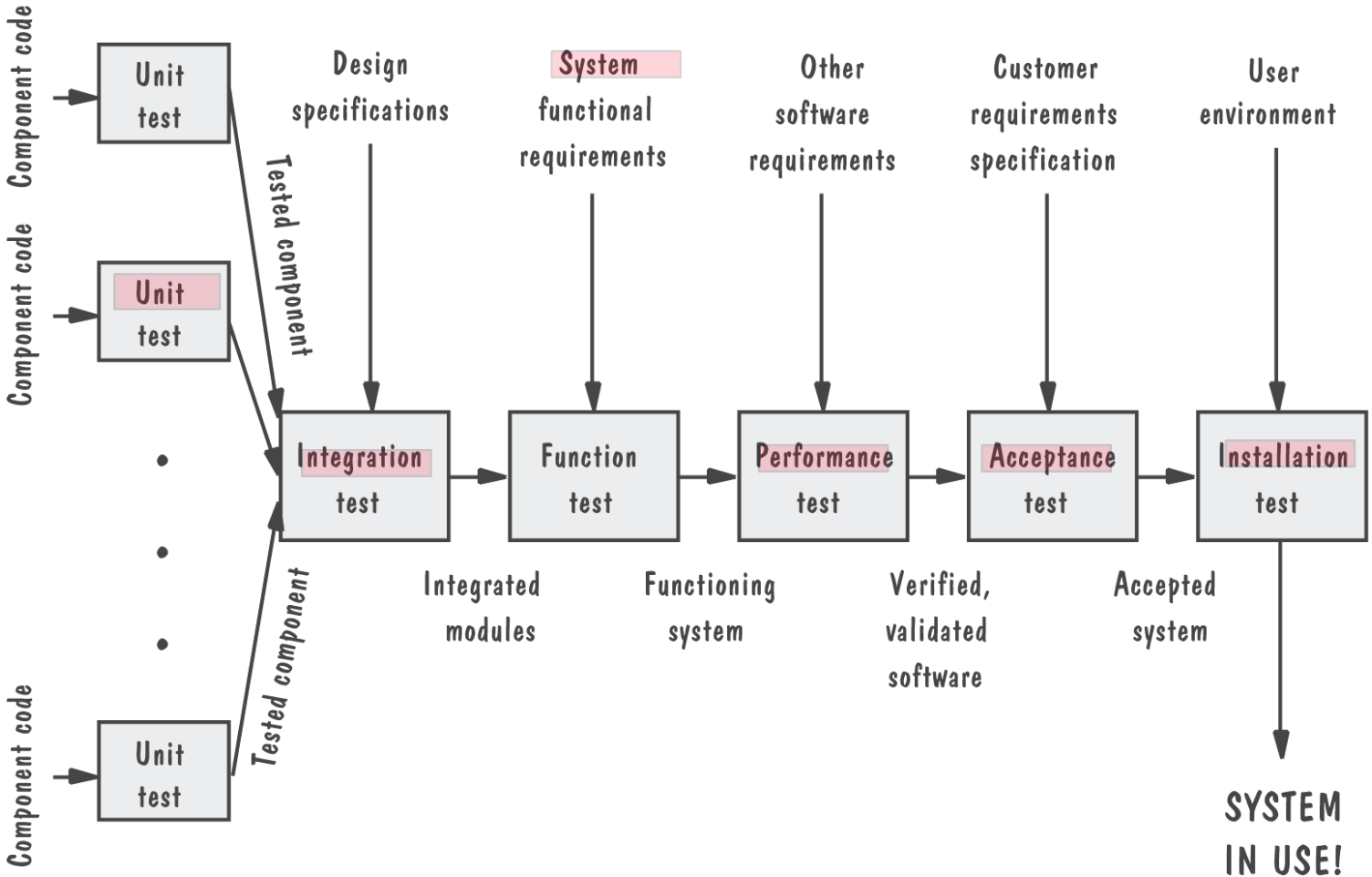
Testing phases (V-Model)



V-Model

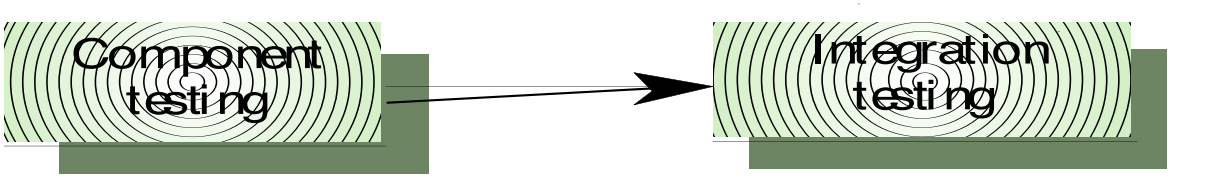

Planning
Maintenance
Requirements
System and Acceptance Testing
Architecture
Integration Testing
Detailed Design
Unit Testing
Implementation

Testing steps (Pfleeger & Atlee, 2006)



Testing phases and executors

Il TESTING sulle UNITÀ lo fa il PROGRAMMATORE (siamo a liv. tecnico)
Più si sale di livello (SOTTOSISTEMI) e il testing lo si fa fare a terzi



Stages of the testing process

1. Unit/Component testing
    1. Unit testing, where **individual** program units or object classes are tested in isolation. Unit testing should focus on testing the **functionality** of objects or methods. It is a Defect Testing of individual functions or methods, classes, ...
    2. Component testing, where several individual units are integrated to create composite components. Component testing should focus on testing component **interfaces**, which should conform to their specification.
    - Usually the responsibility of the component/unit **developer** (except sometimes for critical systems)
    - Tests are derived from the developer's experience/**minispec**/design **spec/**
2. Integration/System/User testing

    l'interfaccia che fa "parlare" (collaborare) assieme tutti i componenti funziona?

    tutte le singole componenti sono già testate

    System testing, where some or all of the components in a system are **integrated** and the system is tested as a whole. System testing should focus on testing component interactions and overall functions/performance
    - Testing of **groups** of components integrated to create a system or sub-system
    - The responsibility of an independent **testing team**
    - Tests are based on a **system spec**ification

(Defect) Testing
- come si fa? -

Testing: generalità e priorità

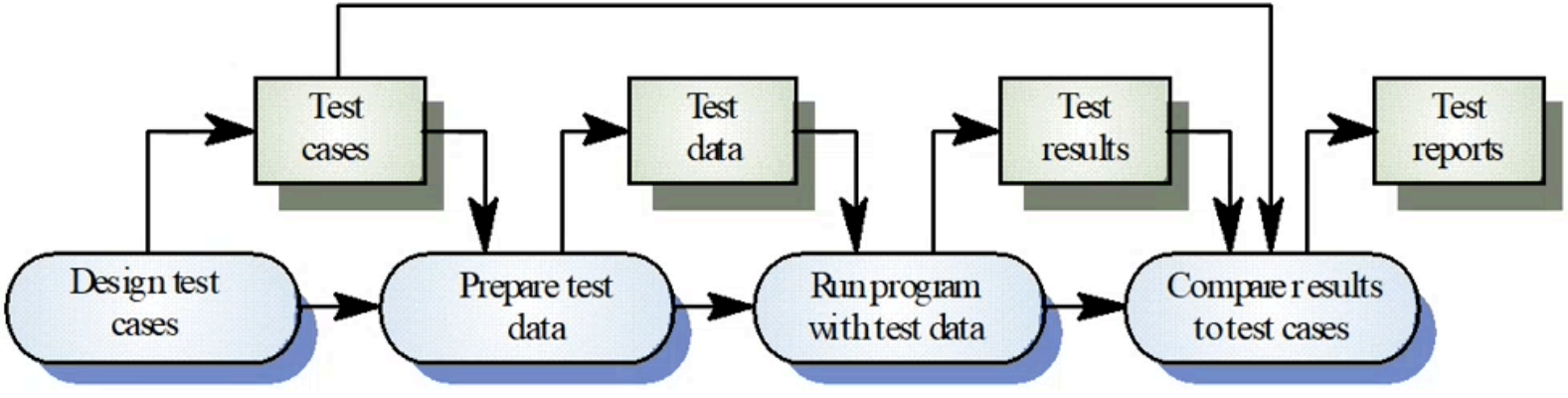
- **Basic Fact to keep in Mind**: **Only exhaustive (complete!!) testing could show that a program is free from defects. However, exhaustive testing is impossible**
- Tests should exercise a system's capabilities rather than its single components (e.g. all menus' commands, combinations, all correct - or incorrect - inputs)
- Testing old capabilities is more important than testing new capabilities: **sempre ri-testare** ciò che funzionave prima dell'ultimo aggiornamento
- Testing typical situations is more important than boundary value cases

Test data and test cases

- ***Test data***: inputs which have been devised to test the system
- ***Test case***: inputs to test the system and the predicted outputs from these inputs if the system operates according to its specification

- **LE VARIE TIPOLOGIE DI TESTING SI DIFFERENZIANO PER COME VENGONO PROGETTATI I CASI DI TEST!!**

Modello del processo per l'esecuzione del defect testing



Test cases for defect vs.validation testing

For Validation Testing: test cases reflect the system's expected use

For Defect Testing: test cases are designed to expose defects. The test cases in defect testing can be deliberately obscure and need not reflect how the system is normally used

Various kinds of testing

- 0. General Issues
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress

Development Testing

Development Testing

- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation

User Testing ……

- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

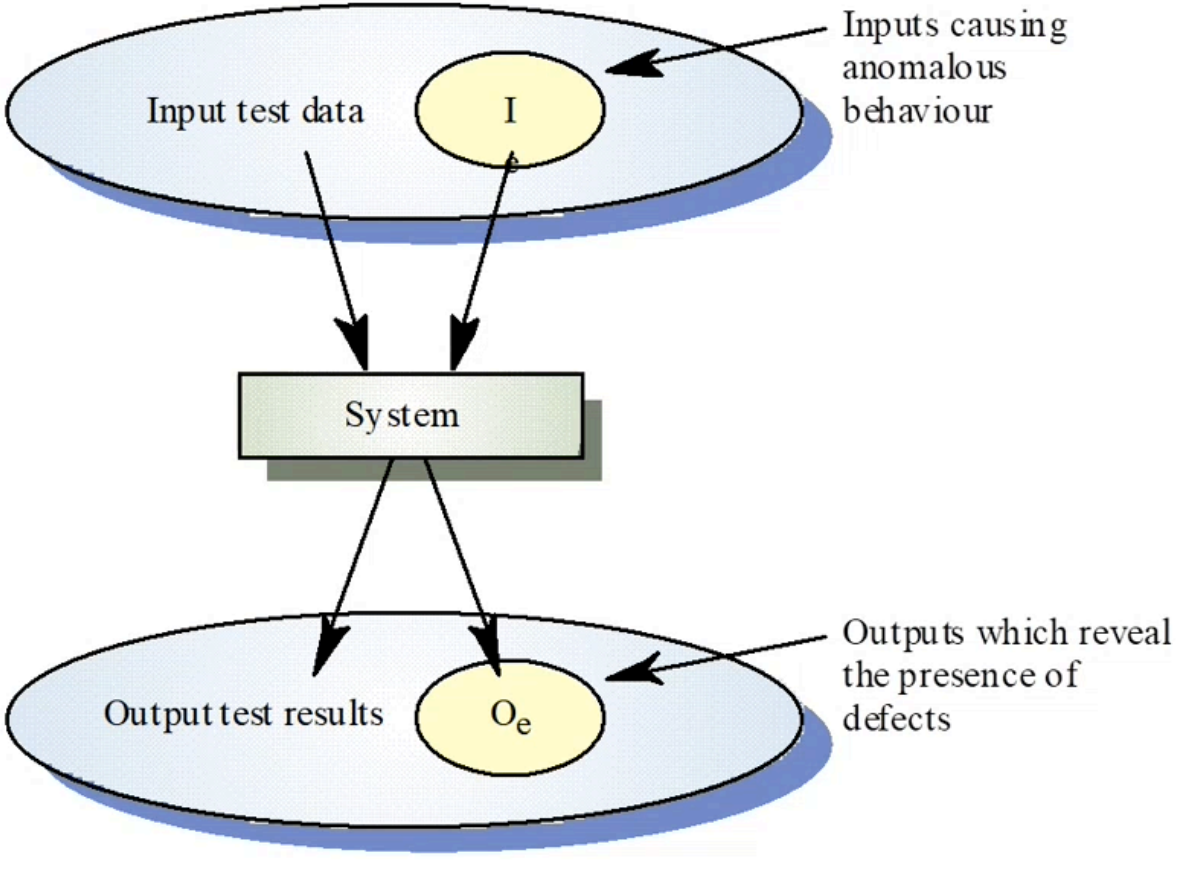
1. Component testing

1.A Black-box (closed-box) testing

- An approach to testing where the program is considered as a 'black-box'
- The program test cases are designed by analysing system specifications
  - so solo cosa fa il sistema MA NON SO COME È DENTRO
- Test planning can begin early in the software process
- [ricordare il V-Model….]

I test li faccio ragionando per testare le funzionalità

Black-box testing



SVILUPPARE

Designing test cases

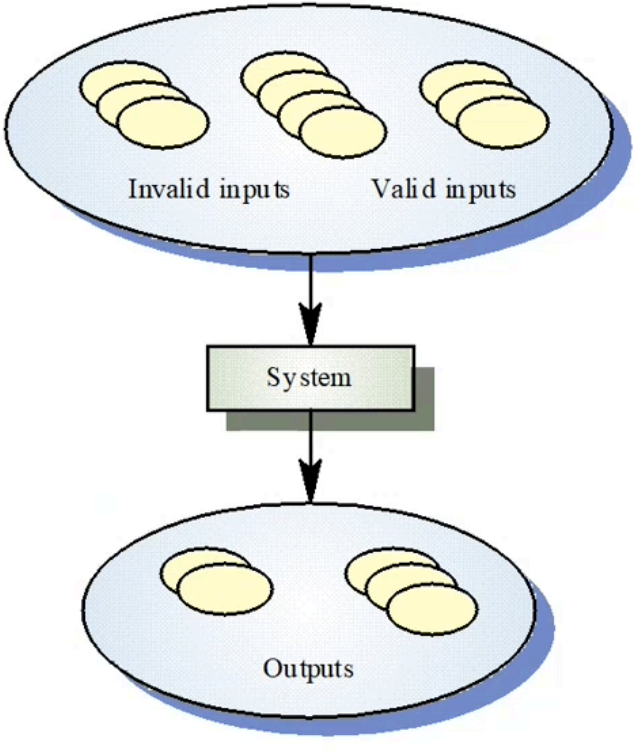
- The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.
- If there are defects in the component, these should be revealed by test cases (which will show *failures*).
- This leads to **2 types** of unit test case (che sono complementari):
  1. The first of these should reflect **normal operation** of a program and should show that the component works as expected. [?] **equivalence partitioning** → provo a vedere con semplici input
  2. The other kind of test case should be based on testing experience of where common problems arise. It should use **abnormal inputs** to check that these are properly processed and do not crash the component. [?] **general testing guidelines** → cerco di fare dei ragionamenti più "diabolici", provo ad inventarmi situazioni che porterebbero a bug → uso l'esperienza

1. Equivalence partitioning (or partition testing)

RAGGRUPPO IN CLASSI gli INPUT in modo da non doverli testare tutti (non testo due input equivalenti (della stessa classe))

- **Approccio utilizzato in tutte le tecniche di testing**
- Input data and output results often fall into different classes where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition
- La numerosità del testing esaustivo **viene ridotta** poichè si testano solo alcuni casi per partizione!!

Equivalence partitioning

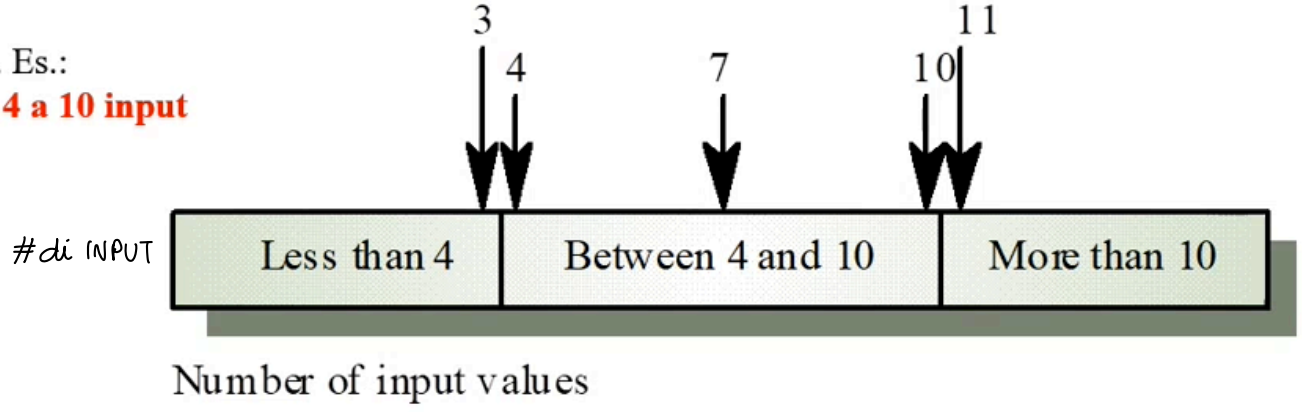


Equivalence partitioning - criteria and examples -

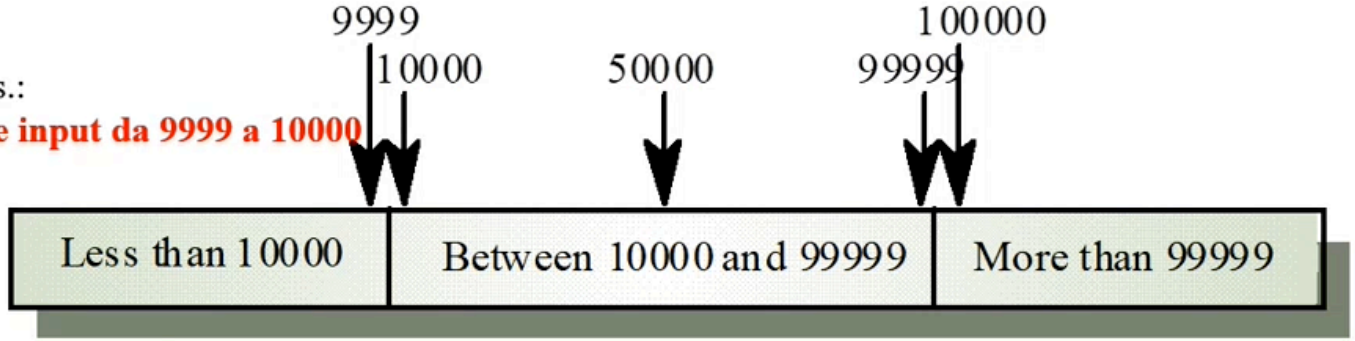
- Partitioning divides inputs and outputs into classes, 'equivalence sets'
  - If input is a 5-digit integer between 10,000 and 99,999, equivalence partitions are <10,000, 10,000- 99, 999 and >= 100,000
- Choose **typical** input of the class, in the '**middle**'
  - 5000, 4000, …
- Choose **atypical/borderline** input for the class, i.e. test cases at the boundary of these sets
  - 9999, 10000, 99999, 100000, 100001,

Equivalence partitions

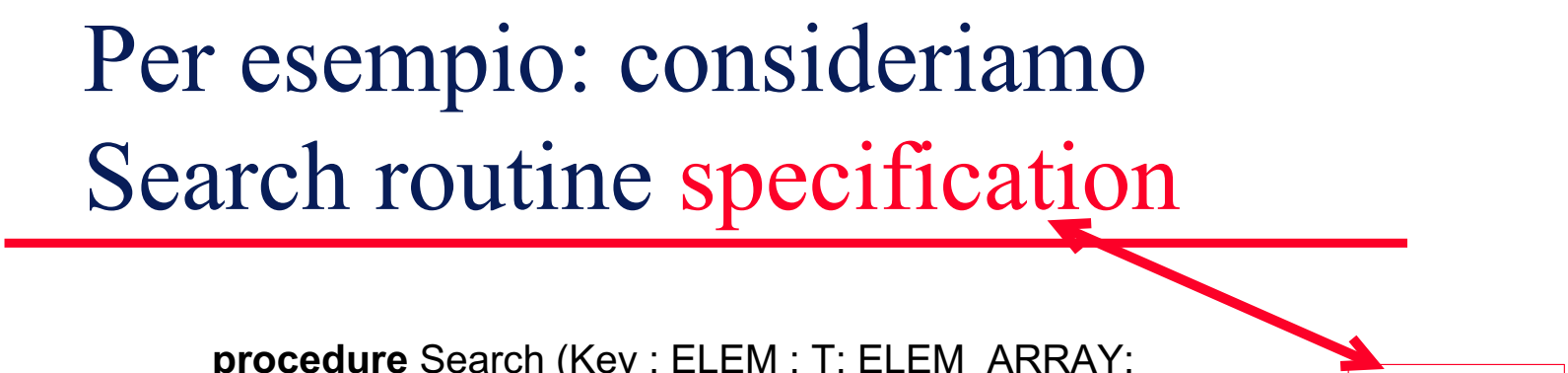
Ad. Es.:
Da 4 a 10 input



Ad. Es.:
Valore input da 9999 a 10000



Per esempio: consideriamo Search routine specification



procedure Search (Key : ELEM ; T: ELEM_ARRAY;
Found : **in out** BOOLEAN; L: **in out** ELEM_INDEX) ;

Pre-condition

-- the array has at least one element
T'FIRST <= T'LAST

Post-condition

-- the element is found and is referenced by L
(Found and T (L) = Key)

or

-- the element is not in the array
(**not** Found **and**
not (**exists** i, T'FIRST >= i <= T'LAST, T (i) = Key))

Search routine - input partitions

- Inputs which conform to the pre-conditions
- Inputs where a pre-condition does not hold
- Inputs where the key element is a member of the array
- Inputs where the key element is not a member of the array

2. General testing guidelines

- Choose inputs that **force** the system to generate all **error** messages
- Design inputs that cause input **buffers** to **overflow**
- **Repeat** the same input or series of inputs numerous times, in **different orders**
- **Force invalid** outputs to be generated
- **Force** computation results to be too large or too small
- …

Testing guidelines (for sequences/vettori)

- Test software with sequences which have only a single value
- Use sequences of different sizes in different tests
- Derive tests so that the first, middle and last elements of the sequence are accessed
- Test with sequences of zero length

Search routine - input partitions

Array	Element
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

li progetto

Test Cases

e per ciascun caso di test osservo se gli output corrispondono

chiave da cercare

Input sequence (T)	Key (Key)	Output (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Test Data

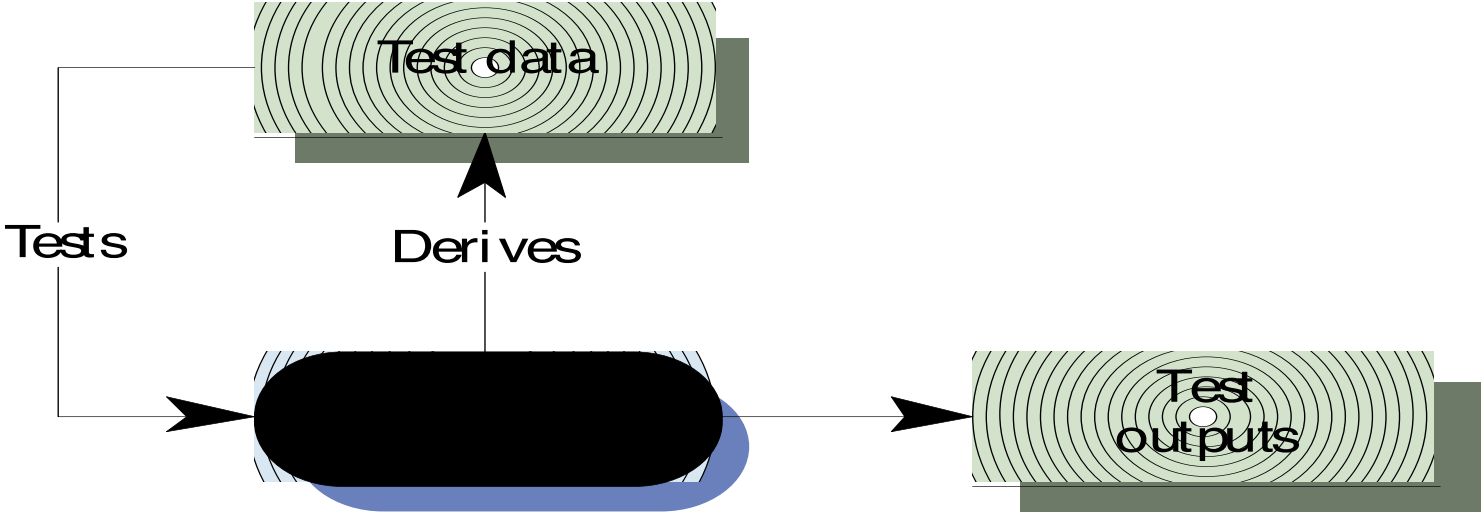
Various kinds of testing

- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress
- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation
- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

1.B Structural testing

- Sometime called white-box (or glass-box or clear-box) testing
  ↳ guardo come è fatto il programma al suo interno
- Derivation of test cases according to program structure. Knowledge of the program is used to identify/design **additional** test cases (ulteriori rispetto a quelli già progettati con Black-Box)
  ↳ prima faccio il test BLACK BOX e poi gli aggiungo case test con il WHITE BOX
- **Objective** is to exercise **all** program statements (not all possible path combinations)

1.B.1 White-box testing



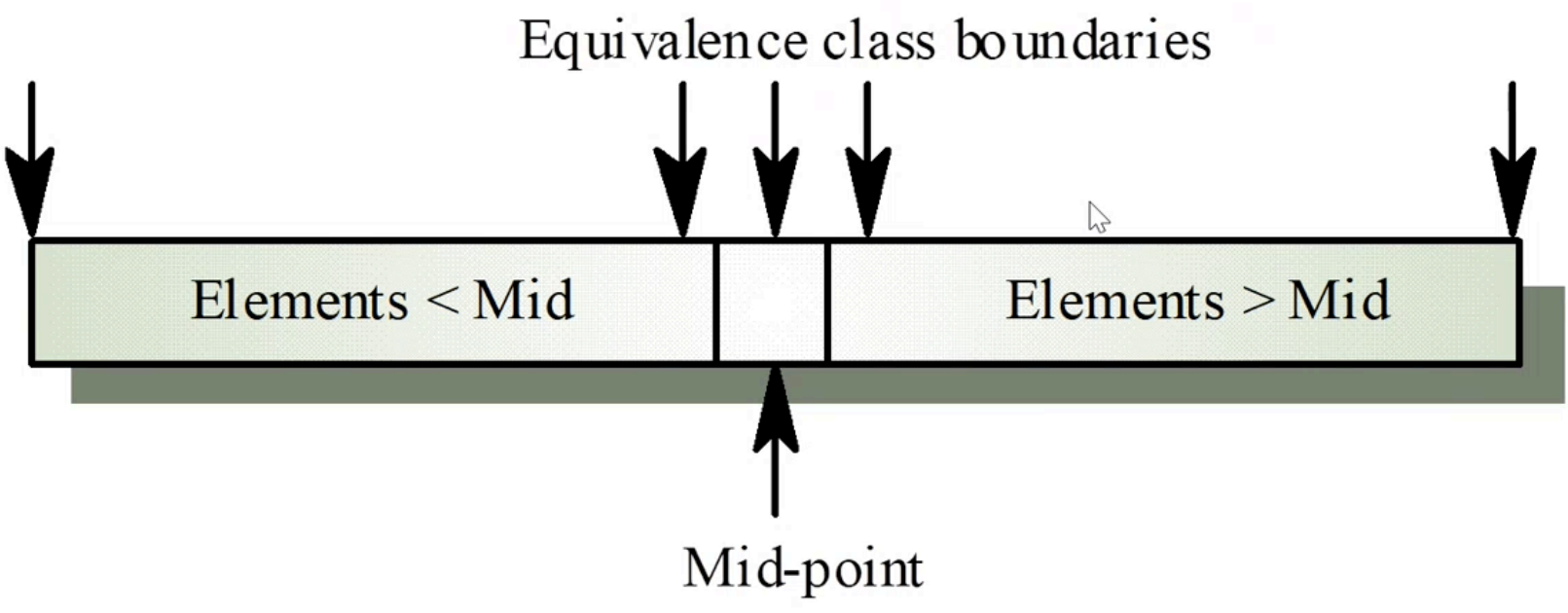
```
class BinSearch {

// This is an encapsulation of a binary search function that takes an array of
// ordered objects and a key and returns an object with 2 attributes namely
// index - the value of the array index
// found - a boolean indicating whether or not the key is in the array
// An object is returned because it is not possible in Java to pass basic types by
// reference to a function and so return two values
// the key is -1 if the element is not found

        public static void search ( int key, int [] elemArray, Result r )
        {
                int bottom = 0 ;
                int top = elemArray.length - 1 ;
                int mid ;
                r.found = false ; r.index = -1 ;
                while ( bottom <= top )
                {
                        mid = (top + bottom) / 2 ;
                        if (elemArray [mid] == key)
                        {
                                r.index = mid ;
                                r.found = true ;
                                return ;
                        } // if part
                        else
                        {
                                if (elemArray [mid] < key)
                                        bottom = mid + 1 ;
                                else
                                        top = mid - 1 ;
                        }
                } //while loop
        } // search
} //BinSearch
```

Binary search (Java)

Binary search - equiv. partitions

- Pre-conditions satisfied, key element in array
- Pre-conditions satisfied, key element not in array
- Pre-conditions unsatisfied, key element in array
- Pre-conditions unsatisfied, key element not in array
- Input array has a single value
- Input array has an even number of values
- Input array has an odd number of values

Binary search equiv. partitions: where the key lies



Binary search – adding further test cases

Input array (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

Various kinds of testing

- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress
- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation
- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

1.B.2 Path testing

CAMMINI

→ provo ad eseguire tutte le strade almeno una volta

- The objective of path testing is to ensure that the set of test cases is such that **each path** through the program is executed **at least once**
- The starting point for path testing is a **program flow graph** that shows nodes representing program decisions and arcs representing the flow of control
- Statements with conditions (if, …) are therefore nodes in the flow graph

Program flow graphs

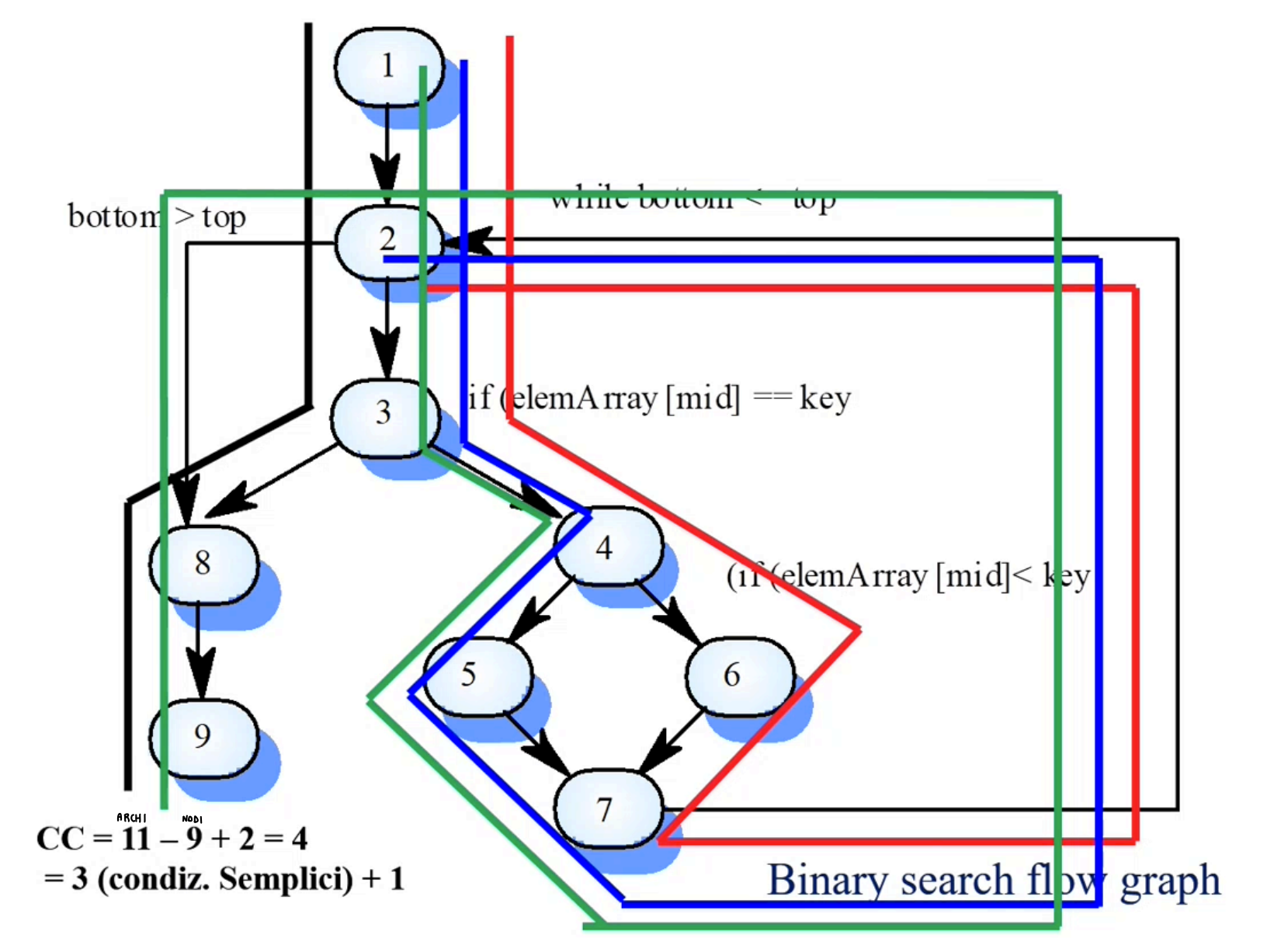
- Describes the program control flow. Each branch is shown as a separate path and loops are shown by arrows looping back to the loop condition node
- Used as a basis for computing the **cyclomatic** complexity (o complessità di **McCabe**), uno dei vari tipi di Metrica dells complessità procedurale
- Cyclomatic complexity = Number of edges - Number of nodes +2

(vedi lezioni/testo Dr. Baruzzo)

Cyclomatic complexity

- The number of tests needed to test **all control** statements equals the cyclomatic complexity
- Cyclomatic complexity **equals** number of (simple, no AND, OR or NOT) conditions in a program + 1 (with no goto)
- Useful if used with care. Does not imply adequacy of testing.

- **NB.** Although all paths are executed, all combinations of paths are **not** executed


1
2
3
4
5
6
7
8
9
bottom > top
while bottom <= top
if (elemArray [mid] == key
(if (elemArray [mid]< key
CC = 11 – 9 + 2 = 4
= 3 (condiz. Semplici) + 1
Binary search flow graph

Independent paths

- 1, 2, 3, 8, 9 (key trovato subito)
- 1, 2, 3, 4, 6, 7, 2 (non trovato, mi sposto e ricerco)
- 1, 2, 3, 4, 5, 7, 2 (non trovato mi sposto e ricerco)
- 1, 2, 3, 4, 6, 7, 2, 8, 9 (non trovato)
- Test cases should be derived so that all of these paths are executed
- A dynamic **program analyser** may be used to check that paths have been executed

Valori di Complessità Ciclomatica

- Valori di complessità per una classe* superiori a 15-20 indicano un codice molto complicato, che sarà più difficile da capire, da testare e da modificare. ? emerge quindi la necessità di un refactoring
- Esistono strumenti che permettono di visualizzare la complessità ciclomatica di un Sistema e delle sue varie classi.
- Monitorare periodicamente l'andamento della complessità ciclomatica delle varie classi del Sistema, permette di analizzare l'andamanto, i miglioramenti, i peggioramenti, ecc.

(vedi lezioni/testo Dr. Baruzzo)

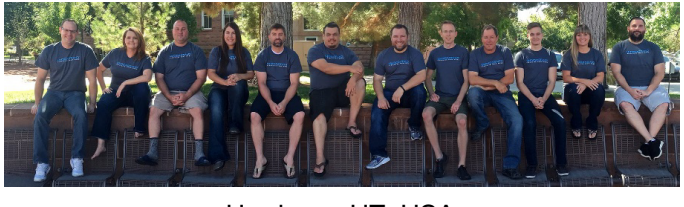
(*) La complessità di una classe è indicata dal massimo delle complessità calcolate per ciascuno dei suoi metodi.

Strumenti per visualizzare la struttura del codice

Qui visualizzato il Cyclomatic Treemap delle varie classi, con la complessità ciclomatica evidenziata da un codice colore.

(vedi lezioni Dr. Baruzzo)

scitools Understand



Hurricane UT, USA

Various kinds of testing

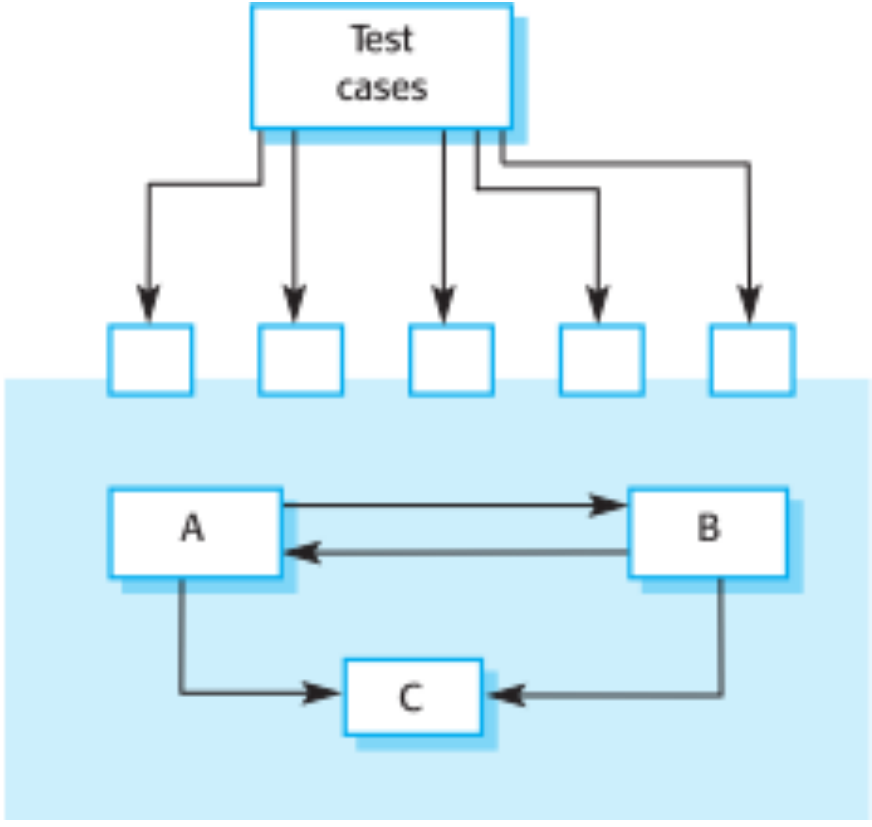
- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress
- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation
- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

1.B.3 Interface testing

Devo INTEGRARE i moduli (testati singolarmente) e testo se a loro INTEGRAZIONE funziona o meno

- It becomes important when modules or sub-systems have to be **integrated** to create larger systems (modules, components or subsystems)
- Testing the interface of a single component.
- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces
- Particularly important for object-oriented development as objects are defined by their interfaces

Interface testing



Vari Meccanismi di Interfacciamento

- Parameter interfaces
  - Data passed from one procedure to another
- Shared memory interfaces
  - Block of memory is shared between procedures or functions
- Procedural interfaces (API/Web Service)
  - Sub-system encapsulates a set of procedures to be called by other sub-systems
- Message passing interfaces
  - Sub-systems request services from other sub-systems

Perchè è importante testare l'interfaccia? Tipici errori …

- Interface misuse
  - A calling component calls another component and makes an error in its use of its interface e.g. parameters in the wrong order
- Interface misunderstanding
  - A calling component embeds assumptions about the behaviour of the called component which are incorrect
- Timing errors
  - The called and the calling component operate at different speeds and out-of-date information is accessed

Interface testing guidelines

- Design tests so that parameters to a called procedure are at the **extreme ends** of their ranges
- Always test pointer parameters with **null pointers**
- Design tests which **cause** the component to **fail**
- Use **stress** testing in message passing systems
- In shared memory systems, vary the order in which components are activated

Various kinds of testing

- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress
- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation
- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

1.B.4 Stress testing

- Exercises the system beyond its maximum design load. Stressing the system often causes defects to come to light
- Stressing the system test failure behaviour. Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data
- Particularly relevant to distributed (Web) systems which can exhibit severe degradation as a network becomes overloaded

Various kinds of testing

- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress

abbiamo testato le singole componenti, ora dobbiamo metterle assieme e testare

- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation

- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

2. Integration testing

Integration testing: **focalizza gli aspetti dell'integrazione di componenti**

- Tests complete systems or subsystems constituted by **integrated components**
- The focus in system testing is testing the **interactions** between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behaviour of a system.
- Integration testing should be **black-box** testing with tests derived from the specification
- Main difficulty is localising errors

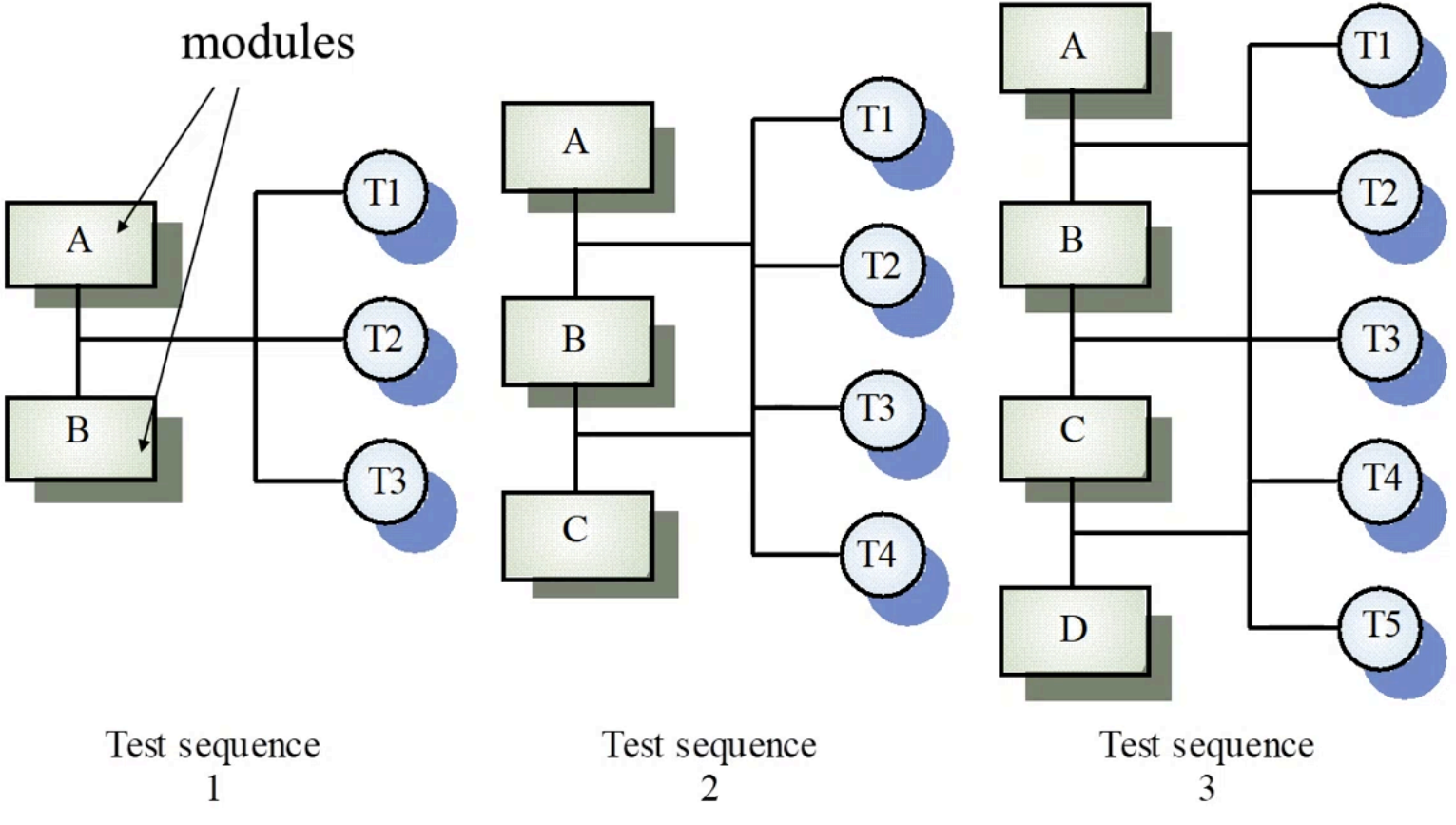
→ so che le singole componenti vanno a chi do la colpa allora? cerco di procedere facendo piccoli incrementi

? **Incremental integration** testing reduces this problem

Big Bang vs. Incremental testing

- Big Bang = integrare tutto e poi testare ….(.. e sperare..)
- … problema: dove è il bug?
- Approccio alternativo al tutto insieme….
- Incremental….

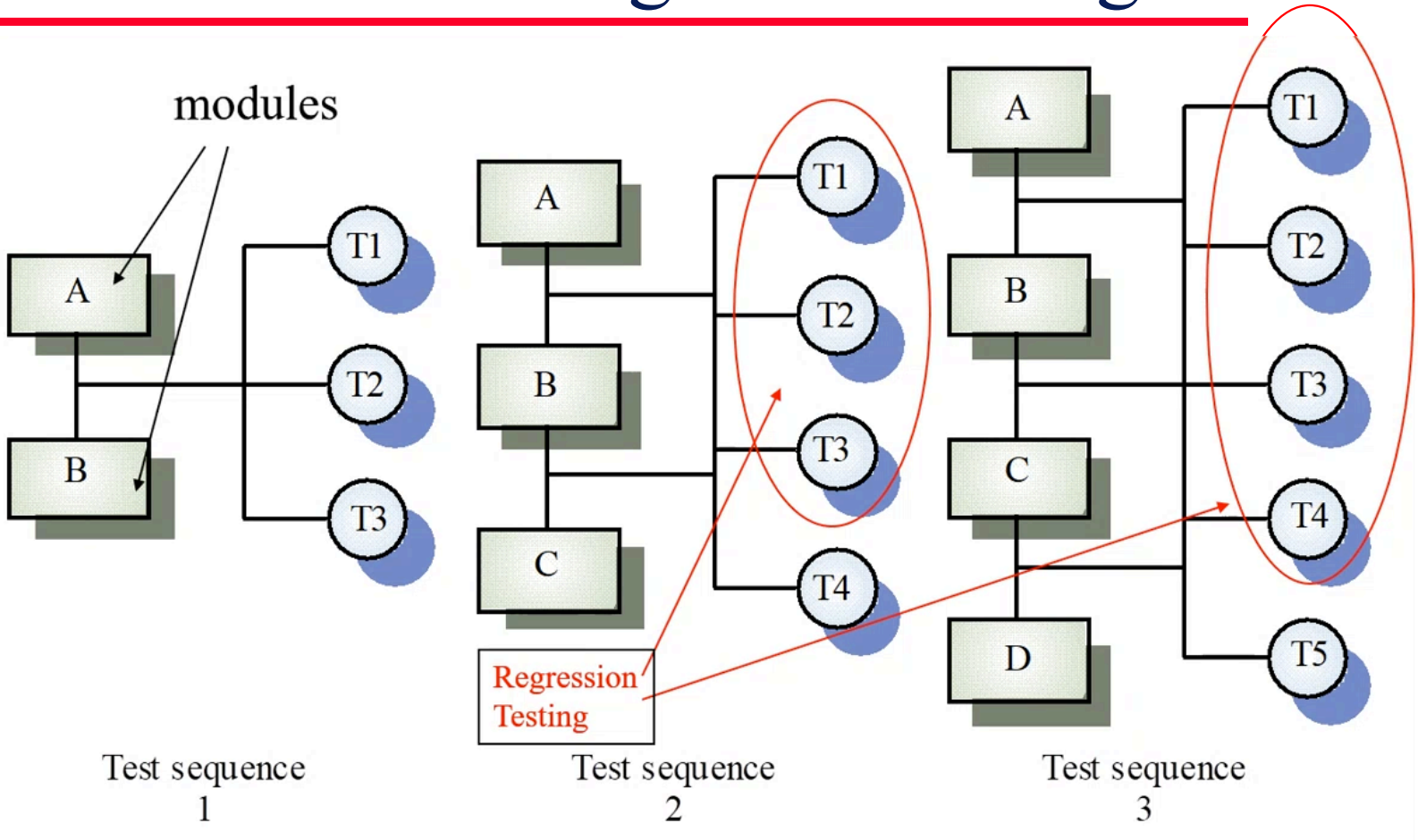
2.A Incremental integration testing



io provo i test T1 e T2 su A e B, poi aggiungo C e provo i test T1, T2, T3

↳ li riprovo perché aggiungendo C magari il comportamento di A e B potrebbe cambiare

Incremental integration testing



Various kinds of testing

- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress
- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation
- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

2.A.1 Regression testing

- Nell'incremental integration testing è necessario **ripetere**, ad ogni step incrementale, **tutti i test già eseguiti** al passo incrementale precedente.
- Ciò è necessario per verificare che i test già fatti senza portare a problemi (malfunzionamenti), si comportino ancora in questo modo. Altrimenti è necessario intervenire.
- In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are re-run every time a change is made to the program.
- Testing Case tools: **JUnit**, …

Approaches to integration testing

- Top-down testing
  - Start with **high-level** system and integrate from the top-down replacing individual components by **stubs** where appropriate

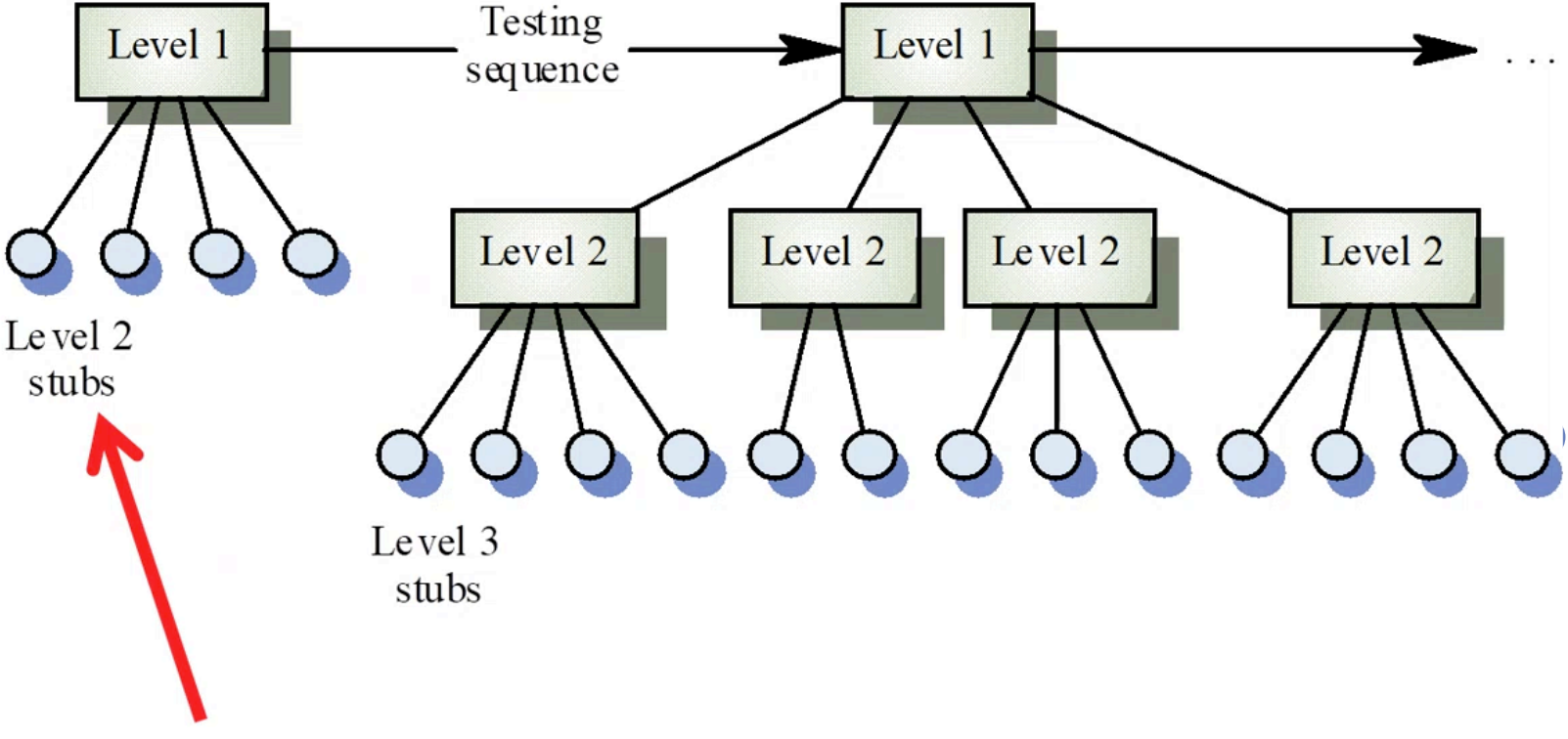
STUB is a special-purpose program aimed at simulating the missing component. The Stub answers the calling sequence and passes back output data that lets the testing process continue. No need to be complex or logically complete.

- Bottom-up testing
  - Start at the **low level**. Integrate individual components (by means of **test drivers**) in levels until the complete system is created

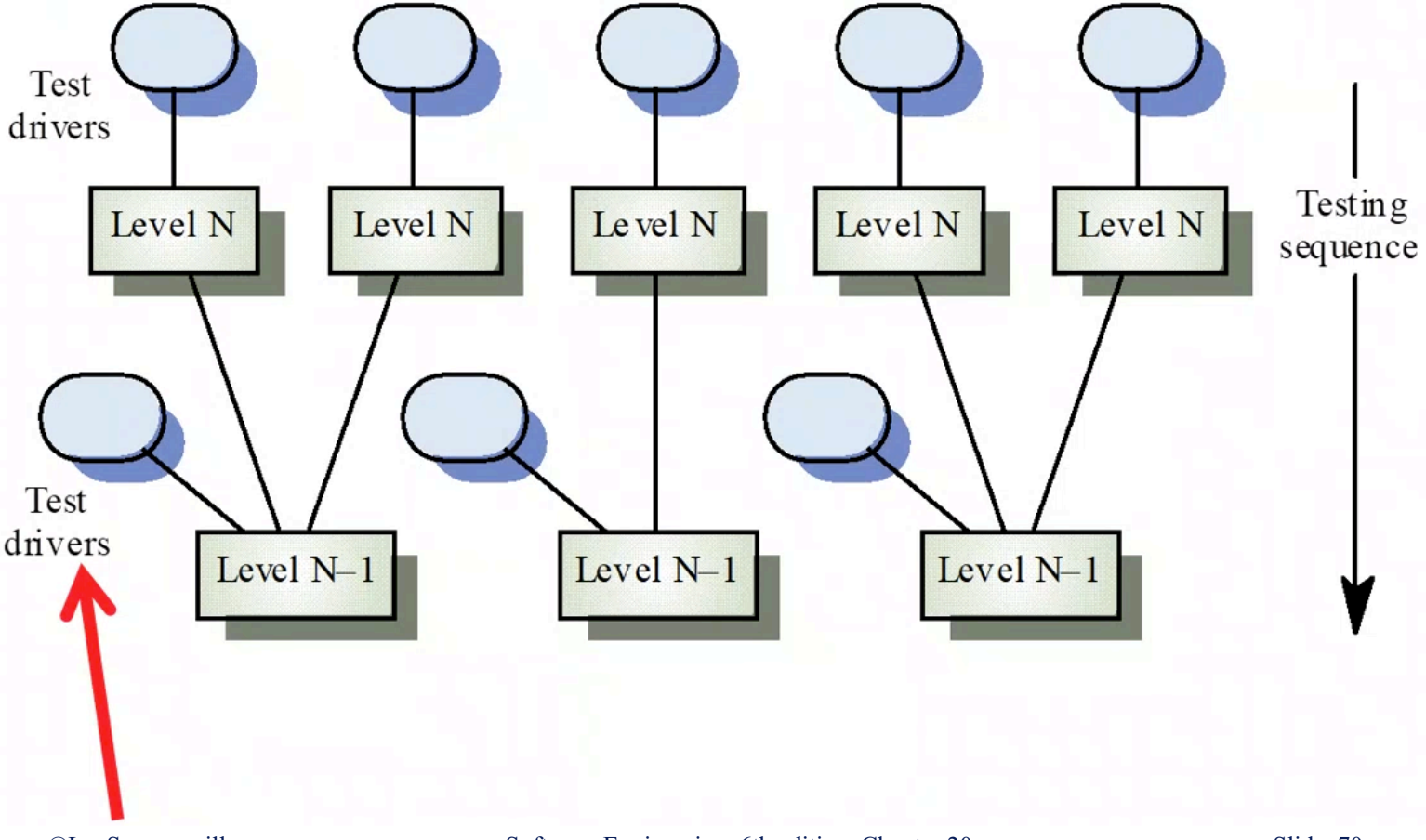
COMPONENT DRIVER is a routine that calls a particular component and passes a test case to it. Not difficult to code.

- In practice, most integration involves a combination of these strategies

2.A.2 Top-down testing

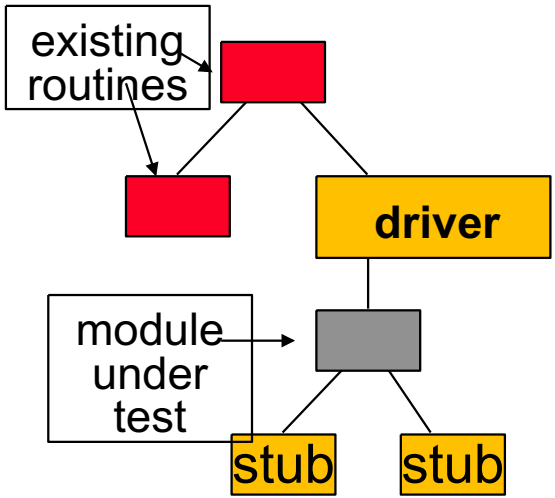


2.A.3 Bottom-up testing



Using Stubs and Drivers

- Stubs and drivers link modules to enable them to run in an environment close to the real one of the future.



Stubs: take the place of modules that are called but have not yet been coded may be invoked or receive or transmit data to the test module as required.

Drivers: call the module under test and pass it test data

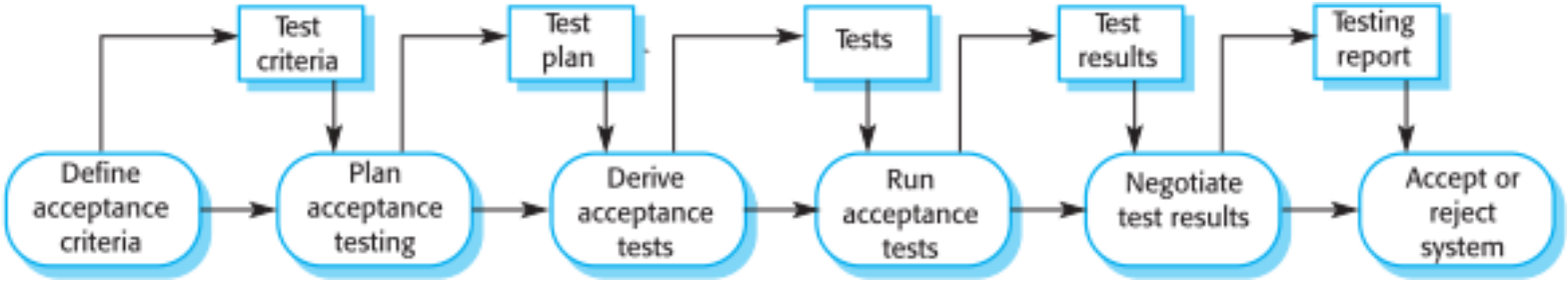
Various kinds of testing

- **0. General Issues**
- I. **STATISTICAL**
- II. **DEFECT**:
- 1. UNIT/COMPONENT
  - 1.A Black Box (Functional, …)
  - 1.B Structural
    - 1.B.1 White box
    - 1.B.2 Path
    - 1.B.3 Interface
    - 1.B.4 Stress
- 2. INTEGRATION/SYSTEM
  - 2.A Incremental
    - 2.A.1 Regression
    - 2.A.2 Top-down
    - 2.A.3 Bottom-up
  - 2.B Acceptance
    - 2.B.1 Release
    - 2.B.2 Performance
    - 2.B.3 Back-to-back testing
  - 2.C Installation
- III. **VALIDATION**
  - 3.A Requirements based
  - 3.B User testing
    - 3.B.1 Alpha Testing
    - 3.B.2 Beta Testing

2.B Acceptance Testing (Collaudo)

- Processo formale in cui lo sviluppatore dimostra di aver realizzato quanto richiesto, eseguendo una serie di test (solitamente definiti a priori; cfr V-Model)
- Si confronta con le specifiche inserite nel contratto
- Coinvolge gli stakeholder, gli utenti, …

The acceptance testing process



2.B.1 Release testing

→ vado a provare l'intero sistema prima di consegnarlo all'utente

- Can be part of the Acceptance process
- The process of **testing a release** of a system that will be distributed to customers.
- Primary goal is to increase the customer's confidence that the system meets its requirements and to convince that the system is good enough for use
- Release testing is usually black-box (also called functional) testing
  - Based on the system specification only;
  - Testers (usually a separate team) do not have knowledge of the system implementation.

2.B.2 Performance testing

- Part of release testing may involve testing the emergent (non-functional) properties of a system, such as performance and reliability.
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable (stress testing).

2.B.3 Back to back testing

Tipicamente utilizzato quando un vecchio sistema viene sostituito da un nuovo sistema, che ha alcune funzionalità simili o (anche solo parzialmente) coincidenti con il precedente. => uso il vecchio sistema di testing per testare solo le funzioni (del nuovo sw) che erano simili/le stesse di quelle del vecchio sw

- Si confrontano i risultati ottenuti con il nuovo Sistema con quelli che si ottengono con il vecchio: dati gli stessi input si verificano gli output, che devono essere uguali per le funzionalità rimaste inalterate.
- In altri termini, i test case sono prodotti fornendo gli input al vecchio sistema, che fornisce automaticamente i relativi output (da verificare poi sul nuovo sistema)

2.C Installation testing

dopo le COLLAUDO (in cui testo nei miei laboratori) devo testare nel luogo in cui va inserito

- Eseguito dopo che il sistema collaudato è stato installato nell'ambiente operativo, l'obiettivo è verificare che funzioni correttamente.
- E' la conclusione del **processo di deployment** (consegna, installazione e messa in funzione).
- Si focalizza sull'interazione del sistema con il contesto (altri sistemi, DB, rete, utenza, ecc.)

↳ porto il SW dall'utente, es. se è commerciale lo scarico io se è professionale me lo vengono ad installare

3. Validation testing

Validation testing

Used to demonstrate to the developer and to the customer that the software meets its **requirements** (!).

↳ requisiti (NON SPECIFICHE), quindi caratteristiche funzionali, non tecniche

A. For **custom** sw, this means that there should be **at least one test** for every requirement in the requirements document.
B. For **generic** sw products, it means that there should be tests for all of the system features, plus **combin**ations of these features, that will be incorporated in the product release.

You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.

3.A Requirements based testing

- A **general principle** of requirements engineering is that requirements should be testable.
- Requirements-based testing is a validation testing technique where you consider each requirement and derive a set of tests for that requirement.

3.B User testing

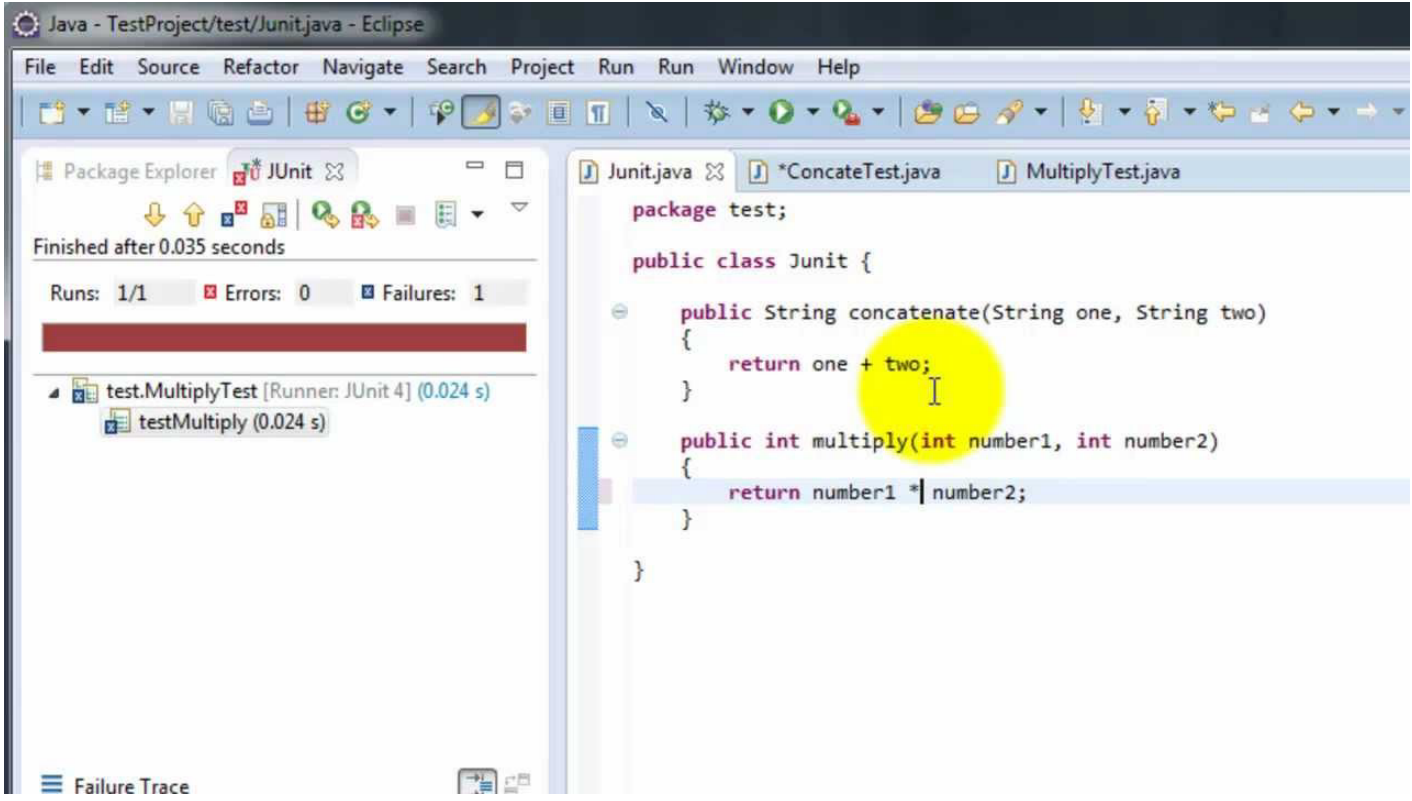
- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing. → situazione REALE
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - The reason for this is that influences from the user's working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.

Types of user testing

- 3.B.1 Alpha testing → ottica utente ma non inserito nell'ambiente finale
  - Users of the software work with the development team to test the software at the developer's site.
- 3.B.2 Beta testing → ottica utente ma inserito nell'ambiente finale (es. le app in BETA che sono di testing e sono scaricabili)
  - A release of the software is made available to many users to allow them to experiment and to raise problems that they discover with the system developers.
- Acceptance testing (già visto in 2.B
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

Test automation

- Testing is an expensive process phase. Testing workbenches provide a range of tools to reduce the time required and total testing costs.
- Automated testing systems support the automatic execution of tests.
- Junit, …



Automated testing

- Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.
- Unit testing frameworks provide generic test classes that you extend to create specific test cases. They can then run all of the tests that you have implemented and report, often through some GUI, on the success of otherwise of the tests.

Automated test components

- A setup part, where you initialize the system with the test case, namely the inputs and expected outputs.
- A call part, where you call the object or method to be tested.
- An assertion part where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful if false, then it has failed.

Set 13 - Cosa ricordare: concetti, motivazioni, conseguenze, relazioni fra concetti, ecc.

- Organizzazione generale del processo di testing, unit/ testing, integration testing; validation testing vs. defect testing; impossibilità pratica di un testing 'esaustivo'. Def. di base: test case, test data, processo per lo svolgimento del singolo test, regression testing.
- Testing (T.) delle componenti. 1.A Black Box T., Equivalence partitioning, criteri generali per progettare i casi di test, criteri per definire le partizioni di equivalenza, criteri per il test di sequenze (array), esempi. 1.B Structural T.; 1B.1 White Box T., esempi; 1.B.2 Path Testing, grafo del flusso di un programma, complessità ciclomatica, suo utilizzo, esempio; 1.B.3 T. delle interfacce, tipologie di interfacce, tipici errori e linee guida; 1.B.4 Stress testing.
- Integration T. Approccio incrementale, approccio, stub e driver, 2.A.1 regression T.; 2.A.2 top-down T., e 2.A.3 bottom up T.; 2.B Acceptance T., processo; 2.B.1 Release T.; 2.B.2 Performance T.; 2.B.3 Back-to-Back T., 2.C Intallation T.
- Validation T. 3.A Requirement based T.; 3.B User T.: 3.B.1 Alpha T., 3.B.2 Beta T.,
- Cenno ai tool di Test Automation