Verification and Validation

Assuring that a software system meets users' needs

SW Inspection

V & V – What?

V & V

avvengono in tutto il ciclo di vita →

VERIFICA: vedo se il SW che ho costruito è conforme a quanto richiesto nelle specifiche → fa quello che mi han chiesto?

VALIDAZIONE: le specifiche risolvono i problemi che voleva l'utente? (→ i veri requisiti) il SW risolve i problemi dell'utente

- **Verification(1) and validation(2)** is intended to show that a system 1. conforms to its specification and 2. meets the requirements of the system customer
- Involves a.checking and b. review processes and system c. testing
- System **testing involves executing** the system with test cases that are derived from the specification of the real data to be processed by the system

V & V

- **Verification(1) and validation(2)** is intended to show that a system 1. conforms to its specification and 2. it really meets the needs and requirements of the system customer
- Involves a. checking and b. review processes and system c. testing
- System **testing involves executing** the system with test cases that are derived from the specification of the real data to be processed by the system

Verification vs Validation

- Verification:

  "Are we building the product right"

"Stiamo costruendo bene il sistema, ossia stiamo costruendo ciò che dovevamo costruire, secondo le specifiche?"

  - The software should conform to its specification

- Validation:

  "Are we building the right product"

"Stiamo costruendo il sistema giusto, ossia quello che veramente serve, che risolve i problemi?"

  - The software should do what the user really requires

Un'ulteriore prospettiva/precisazione sui processi di V& V

Andrea Baruzzo – Carlo Tasso

Verification (def.)

Verification is the process of determining whether a system completely satisfies its specifications. In other words, verification is checking that the system is built right (in the right –expected/specified- way).

It focuses on the matching between the system and the stated specifications.

Validation (def.)

Validation is the process of determining whether a system satisfactorily performs the real-world tasks for which it was developed. Another definition of Validation states that it is the process of determining if a system satisfies the explicit or **implicit** needs of the user/stakeholders. In any case, validation is checking that the right system has been built. Clearly, this concept is based on the implicit assumption that, despite the fact that detailed and accurate system specifications or a blue print are provided, what the user (or the organization/stakeholders) actually wants lies only in the heads of the people involved. This can hardly be clearly expressed in verbal or formal terms, so the ultimate check is only to put the system into practice.

Scope/Granularity of V&V

From another perspective, we can differentiate validation from verification with respect to the level of abstraction (or **granularity**) of the elements involved.

Validation is used at system or subsystem level,

Whereas

va più nel dettaglio

Verification is performed at a more variable granularity: from a single line of source code, to single units or classes, and from clusters of units (modules and subsystem) to the entire system.

V & V
Quando e quali attività

The V & V process: when

- Is a **whole** life-cycle process - V & V must be applied at each stage in the software process.
- Has two main objectives
  1. The discovery of defects in a system
  2. The assessment of whether or not the system is usable, useful, effective in the operational situation.

Static and dynamic V&V

Attività **STATICA**: viene svolta senza eseguire il codice

Attività **DINAMICA**: richiede l'esecuzione del codice

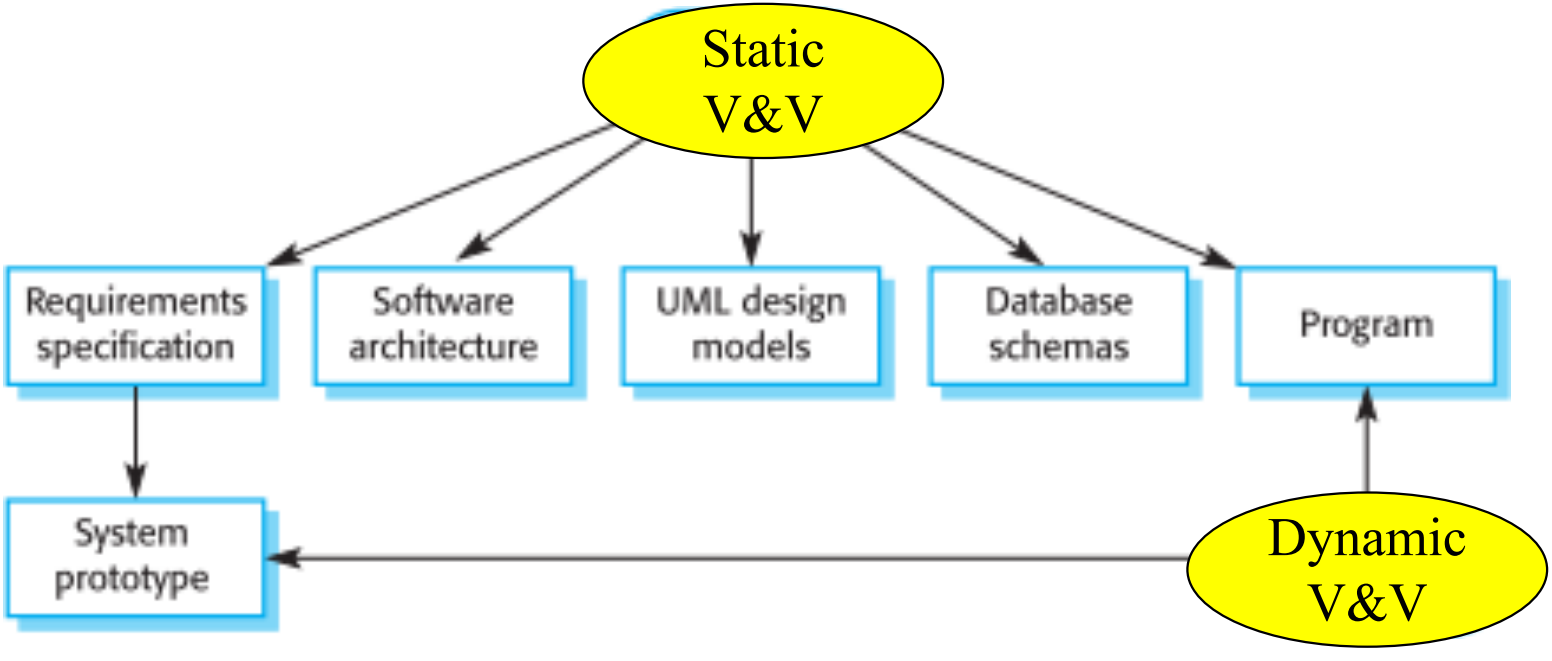
Static and dynamic V&V: le due tecniche principali

1. *Software **inspections*** Concerned with analysis of the static system representation to discover problems (**static** V & V) => guardo il sistema, ovvero la documentazioni
   - May be supplemented by tool-based document and code analysis

   *APPLICABILE IN **TUTTE** LE FASI DI SVILUPPO*
2. *Software **testing*** Concerned with exercising and observing product behaviour (**dynamic** V & V)
   - The system is executed with (test) data and its operational behaviour is observed

   *APPLICABILE SOLAMENTE QUANDO UN **ESEGUIBILE** è A DISPOSIZIONE*

Inspections and testing



Possibili Attività della VERIFICA

*1. Software **inspection*** (of documentation, specifications, design, and source code).

Mode: Static

*2. **Formal** Methods* (on specifications and corresponding code)

Mode: Static

*3. (**Defect**) **Testing*** (of specified functionalities), *Regression Testing, … ne vedremo parecchie…*

Mode: Dynamic

Possibili Attività della VALIDAZIONE

1. ***Validation testing*** (direct use of the system, β-testing, statistical testing, … simulation…)

Mode: Dynamic

2. ***Prototyping***

Mode: Dynamic

3. ***Inspection*** (Requirements, design*, code*, test data validation,)

Mode: Static

(*) evaluating design and programming good practice

Altri aspetti generali importanti

V & V goals

- Verification and validation should establish confidence that the software is fit for purpose
- This does **NOT** mean completely free of defects
- Rather, it must be good enough for its intended use and the type of use will determine the degree of confidence that is needed

How much V & V confidence is required?

- Depends on system's purpose, user expectations and marketing environment
  - Software function
    - » The level of confidence depends on how **critical** the software is to an organisation
  - User **expectations**
    - » Users may have low expectations of certain kinds of software (more in the past, less now)
  - Marketing environment
    - » Getting a product to market early may be more important than finding defects in the program

V & V planning

- Il V&V può arrivare ad impegnare il 50%+ dell'effort e dei costi dello sviluppo ⍰ VA PIANIFICATO
- E' bene iniziare il processo prima possibile
- Bilanciare inspection con testing

The structure of a software test plan

- The testing process
- Requirements traceability
- Tested items
- Testing schedule
- Test recording procedures
- Hardware and software requirements
- Constraints

Metodi Statici

SW Inspection

Software inspections

- Un gruppo di lavoro esamina tutti i prodotti, finali o intermedi, che possono essere esaminati, i documenti e il codice sorgente ((requirements, design, test data, etc.)
- Obiettivo: trovare anomalie, riconoscere aspetti non corretti o adeguati, difformità da standard, inconsistenza
- Do not require execution of a system so may be used before implementation (→ for analysing documents, design documents, documentation, …)
- → **Very effective** technique for discovering bugs

Advantages of inspection

- Many different defects may be discovered in a single inspection. In testing, one defect, may mask another so several executions are required
- 'It reuses'/'**it is based on**' domain and programming knowledge/**experience**. So reviewers are likely to have already seen the types of error that commonly arise.
- It may check **other** aspects (ad esempio design quality, vedi più avanti il concetto di quality review)
- Does not need full version of the system

Inspections and testing

- Inspections and testing are **complementary** and not opposing techniques
- Both should be used during the V & V process
- Inspections can check conformance with a specification but **cannot** check conformance with the customer's real requirements
- Inspections **cannot** check non-functional characteristics such as performance, usability, etc.

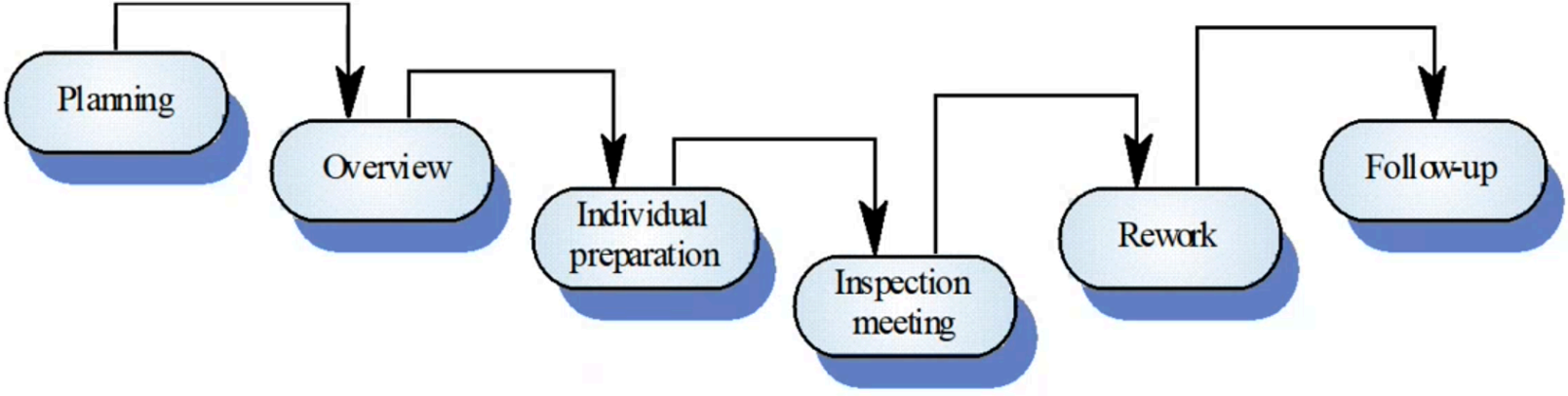
Program inspections

- Nelle sessioni di lavoro (denominate *review*) si ‘cercano’ i difetti
- La correzione NON è inclusa nell’attività: per quello c’è il debugging, analogamente al testing
- Defects may be: logical errors, anomalies in the code that might indicate an erroneous condition (e.g. an un-initialised variable) or non-compliance with standards

Inspection: pre-conditions per poter eseguire

1. A precise specification must be available
2. Team members must be familiar with the organisation standards
3. Syntactically correct code must be available
4. An error checklist should be prepared
5. Management must accept that inspection will increase costs early in the software process
6. Management must not use inspections for staff appraisal

The inspection process: fasi



Inspection procedure

1. System overview presented to inspection team
2. Code and associated documents are distributed to inspection team in advance, for individual pre-screening
3. Inspection takes place and discovered errors are noted
4. Modifications are made to repair discovered errors
5. Re-inspection may or may not be required

Inspection teams

- Made up of at least 4 members (4 ruoli)

1. Author of the code being inspected
2. Inspector who finds errors, omissions and inconsistencies
3. Reader who reads the code to the team
4. Moderator who chairs the meeting and notes discovered errors

- Other roles are *Scribe* (records results of the meeting) and *Chief moderator* (checks the process)

Inspection checklists

- Checklist of common errors should be used to drive the inspection
- Error checklist is programming language dependent
- The 'weaker' the type checking, the larger the checklist
- Examples: Initialisation, Constant naming, loop termination, array bounds, pointers, etc.

Fault class	Inspection check
Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the lower bound of arrays be 0, 1, or something else? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a delimiter explicitly assigned?
Control faults	For each conditional statement, is the condition correc Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted f
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output?
Interface faults	Do all function and procedure calls have the correct number of parameters? Do formal and actual parameter types match? Are the parameters in the right order? If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	If a linked structure is modified, have all links been correctly reassigned? If dynamic storage is used, has space been allocated correctly? Is space explicitly de-allocated after it is no longer required?
Exception management faults	Have all possible error conditions been taken into account?

Inspection checks

Inspection rate and indicative costs

- 500 statements/hour during overview
- 125 source statement/hour during individual preparation
- 90-125 statements/hour can be inspected
- Inspection is therefore an **expensive** process [**BUT** less than testing (Selby et al.)]
- Inspecting 500 lines costs about 40 man/hours effort = £2800 (a 50-100€ all'ora €)

non richiedono l'esecuzione

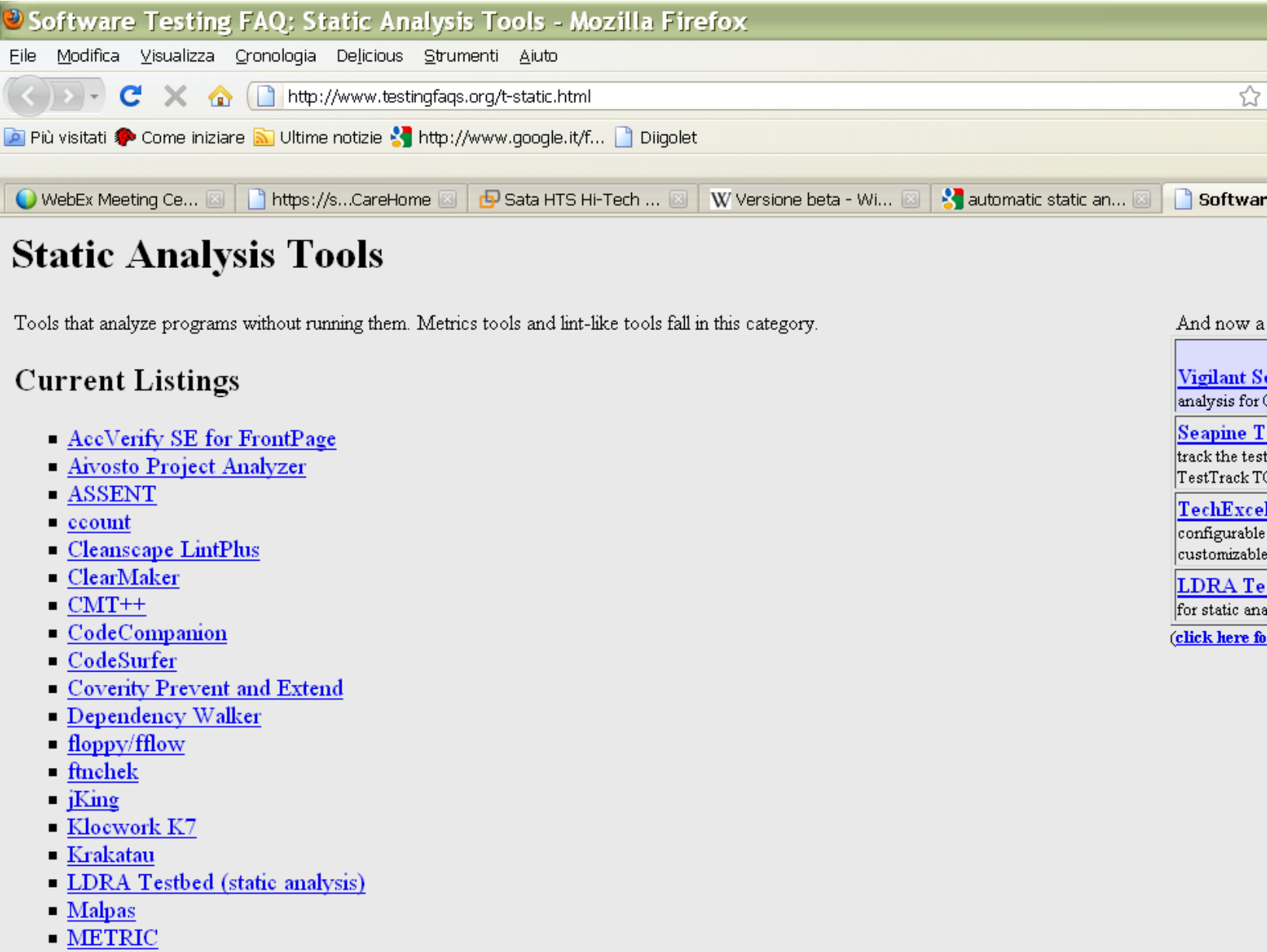
Altro metodo statico: Automated static analysis

Automated static analysis

- Static analysers are software tools for source text processing → individuiamo solo i bug, non guardiamo il funzionamento → usando dei pattern
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the V & V team
- Very effective as an **aid to inspections**. A supplement to but not a replacement for inspections

Static analysis automated 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



Static Analysis Tools

Tools that analyze programs without running them. Metrics tools and lint-like tools fall in this category.

Current Listings

- AccVerify SE for FrontPage
- Aivosto Project Analyzer
- ASSENT
- ccount
- Cleanscape LintPlus
- ClearMaker
- CMT++
- CodeCompanion
- CodeSurfer
- Coverity Prevent and Extend
- Dependency Walker
- floppy/fflow
- ftnchek
- jKing
- Klocwork K7
- Krakatau
- LDRA Testbed (static analysis)
- Malpas
- METRIC

And now a

Vigilant S
analysis for

Seapine T
track the test
TestTrack T

TechExcel
configurable
customizable

LDRA Te
for static ana

(click here fo

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia
Toolbox
Print/export

List of tools for static code analysis

From Wikipedia, the free encyclopedia

This is a list of tools for static code analysis.

Contents [hide]

1 Historical products
2 Open-source or Non-commercial products
2.1 Multi-language
2.2 .NET (C#, VB.NET and all .NET compatible languages)
2.3 ActionScript
2.4 C
2.5 C++
2.6 Java
2.7 JavaScript
2.8 Objective-C
3 Commercial products
3.1 Multi-language
3.2 .NET
3.3 Ada
3.4 C / C++
3.5 Java
4 Formal methods tools
5 See also
6 References
7 External links

Historical products

- Lint — The original static code analyzer of C code.

Open-source or Non-commercial products

Set 16 - Cosa ricordare: concetti, motivazioni, conseguenze, relazioni fra concetti, ecc.

- Definizione (i) e illustrazione di Verifica (Vr) e di Validazione (Vl) globalmente V&V), quando viene svolto nel ciclo di sviluppo, V&V statica (inspection) e dinamica (testing), applicabilità nel ciclo di sviluppo, scope di V&V, livello di granularità, attività statiche e dinamiche svolte per la Vr e per la Vl.
- Elementi di base del TESTING, 'quanto testing serve?', fattori da cui dipende il livello di V&V, testing vs. debugging, processo di debugging, pianificazione del V&V, Processo di Testing, varie fasi, V-Model.
- Ispezione del SW, caratteristiche, potenzialità e vantaggi, relazioni con il testing, tipologia di fault scopribili, precondizioni per effettuare l'inspection, processo di inspection, partecipanti, checklist di tipici controlli da fare, velocità/impegno del processo di inspection
- Analisi statiche automatiche, tipologie di difetti identificabili.