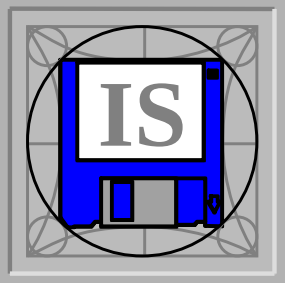
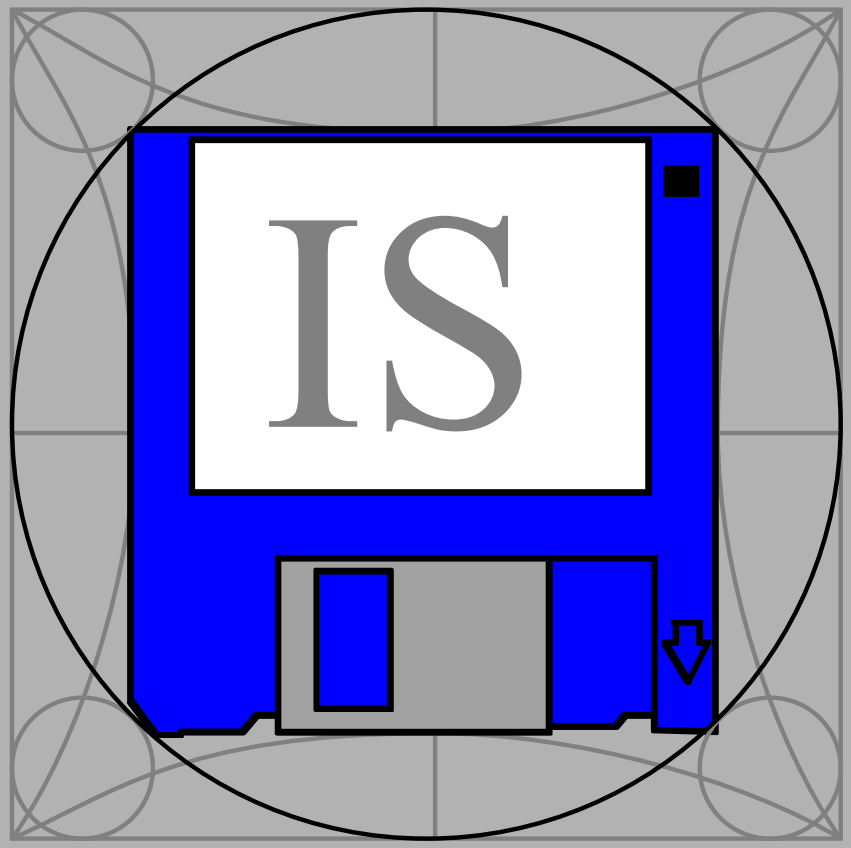


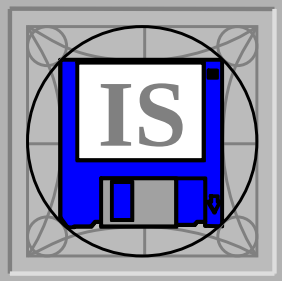
Verifica e validazione: introduzione



Ingegneria del Software

V. Ambriola, G.A. Cignoni,
C. Montangero, L. Semini

Aggiornamenti di: T. Vardanega (UniPD)

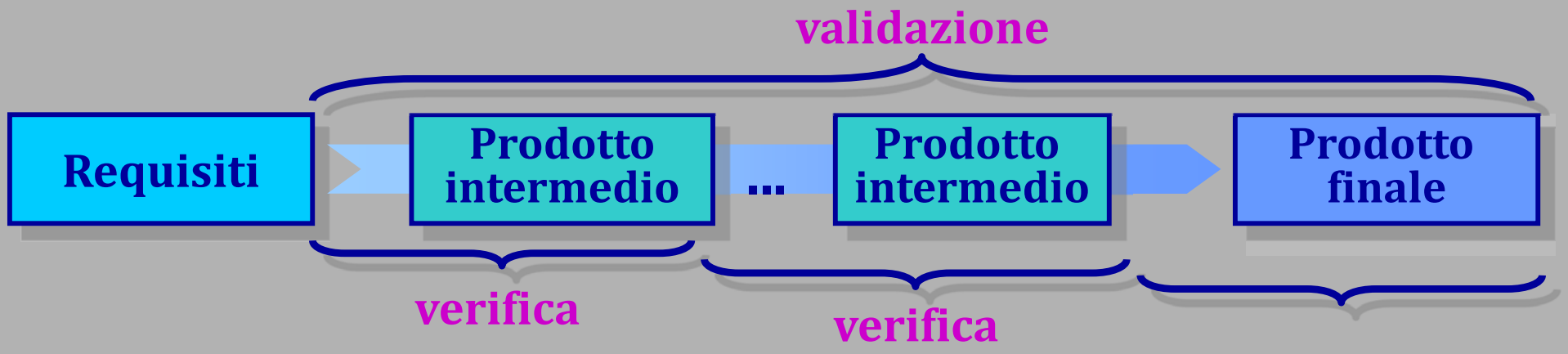


Secondo ISO/IEC 12207

- ***Software verification***
  - ***Provides objective evidence that the outputs of a particular phase of the software development life cycle meet all of the specified [process] requirements for that phase***
  - ***Software verification looks for consistency, completeness, and correctness [of the examined phase outputs], and provides support for a subsequent conclusion that software is validated***
- ***Software validation***
  - ***Confirmation by examination and provision of objective evidence that the software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled***

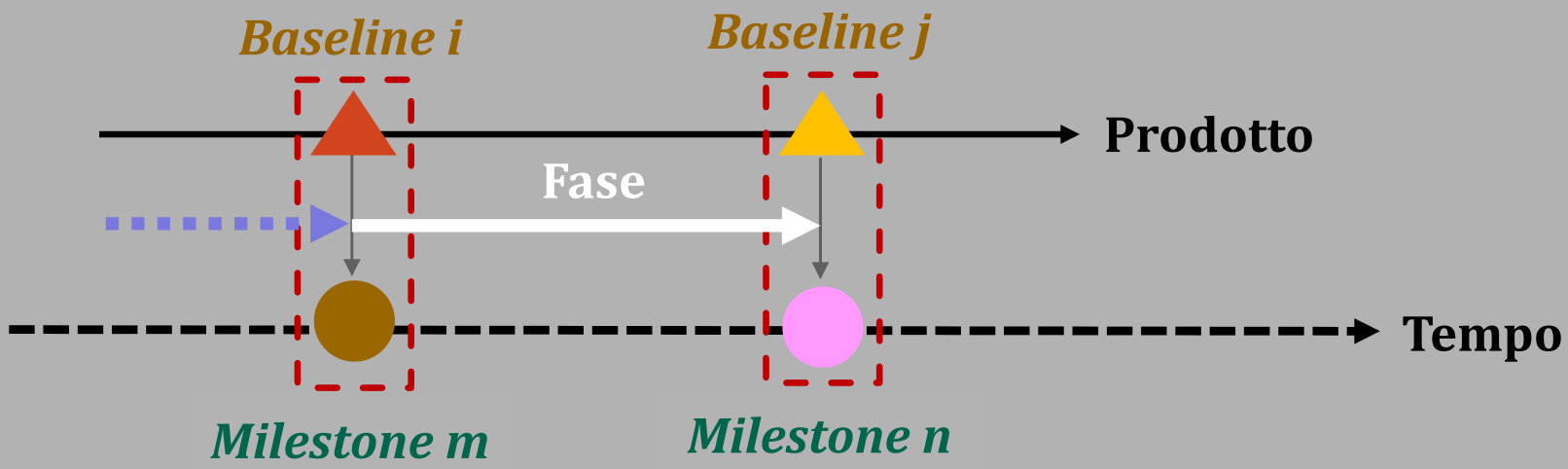
In pratica …

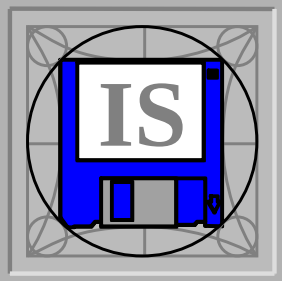
- La verifica accerta che l'esecuzione delle attività attuate nel periodo in esame («fase») non abbia introdotto errori
- La verifica prepara il successo della validazione
- La validazione accerta che il prodotto realizzato sia pienamente conforme alle attese



Quando svolgere verifica?

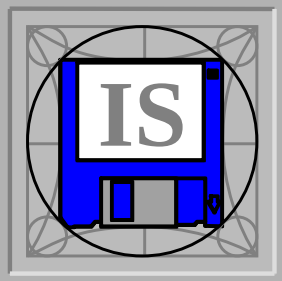
- A ogni incremento di *baseline*
  - In corrispondenza di una *milestone*
- Diciamo «fase» il tempo intercorrente tra due *milestone* successive



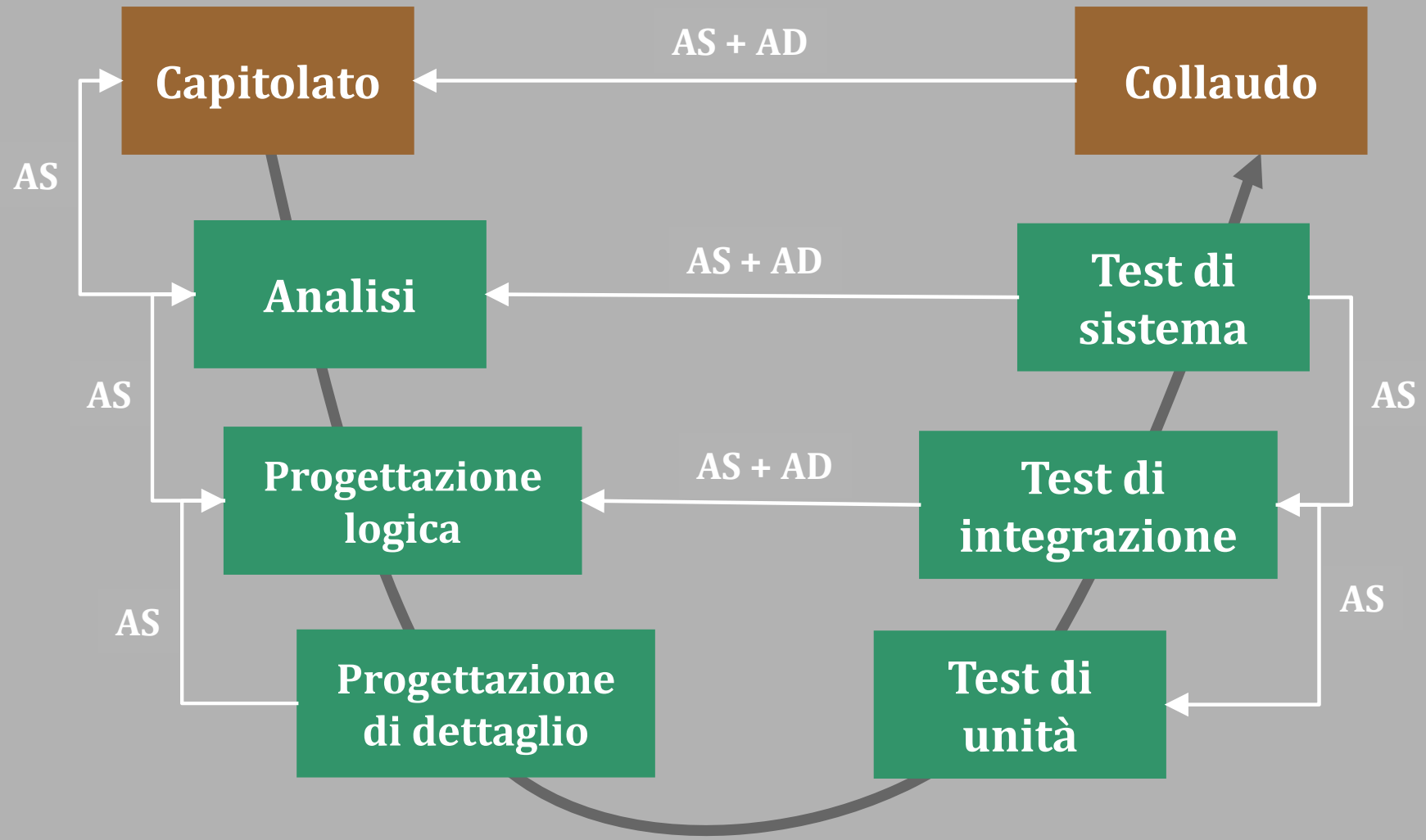


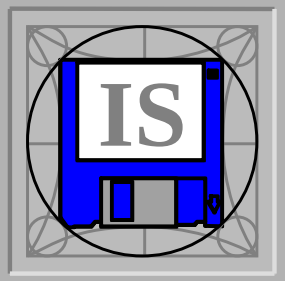
La verifica ha due forme

- **Analisi statica**
  - **Non richiede esecuzione dell'oggetto di verifica**
    - Attuabile già nelle attività propedeutiche alla programmazione
  - **Studia la documentazione e il codice (sorgente e oggetto)**
  - **Accerta conformità a regole, assenza di difetti, presenza di proprietà desiderate**
- **Analisi dinamica**
  - **Richiede esecuzione dell'oggetto di verifica**
    - Attuabile solo dopo l'avvio della programmazione
  - **Viene effettuata tramite prove (i.e. *test*)**
  - **Viene usata anche nella validazione**



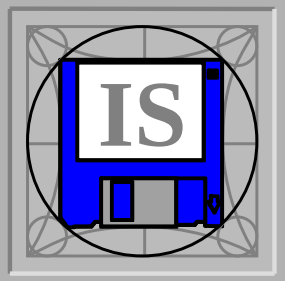
Verifica e validazione nello sviluppo





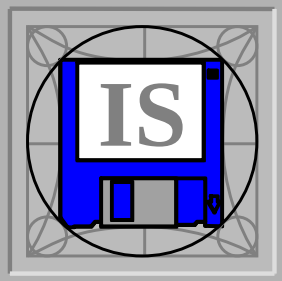
Analisi statica

- Non richiedendo esecuzione dell'oggetto di verifica è applicabile a ogni prodotto di processo
  - Per tutti i processi attivati nel progetto
- Può usare metodi di lettura (*desk check*)
  - Impiegati solo per prodotti semplici
- Oppure metodi più formali
  - Basati su prova assistita di proprietà, utile soprattutto quando la dimostrazione empirica ha costo proibitivo



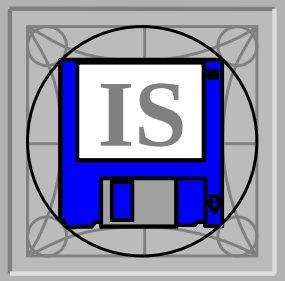
Metodi di lettura

- *Walkthrough* e *Inspection*
  - Svolte tramite studio dell'oggetto di verifica
  - Lettura umana o automatizzata
- Efficacia dipendente dall'esperienza dei verificatori
  - Nell'organizzare le attività da svolgere
  - Nel documentare le risultanze
- Modalità relativamente complementari



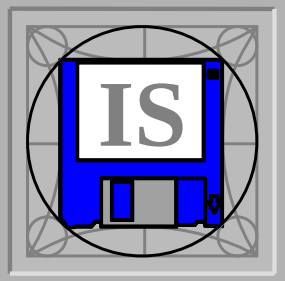
Walkthrough: definizione

- **Obiettivo**
  - **Rilevare la presenza di difetti attraverso lettura critica ad ampio spettro del prodotto in esame**
- **Agenti**
  - **Gruppi misti verificatori/sviluppatori, ma ruoli distinti**
- **Strategia**
  - **Esame privo di assunzioni o presupposti**
    - Per il codice: percorrerlo simulandone possibili esecuzioni
    - Per documenti: studiarne ogni parte come farebbe un compilatore



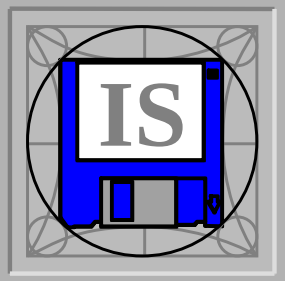
Walkthrough: attività

- **Passo 1: pianificazione**
  - Autori e verificatori
- **Passo 2: lettura**
  - Solo verificatori
- **Passo 3: discussione**
  - Autori e verificatori
- **Passo 4: correzione dei difetti**
  - Solo autori
- **Ogni passo documenta attività svolte e risultanze**



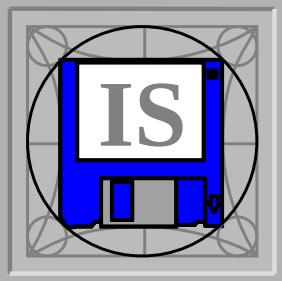
Inspection: definizione

- **Obiettivi**
  - **Rilevare la presenza di difetti, eseguendo lettura mirata dell'oggetto di verifica**
- **Agenti**
  - **Verificatori**
- **Strategia**
  - **Ricerca focalizzata su presupposti (*error guessing*)**



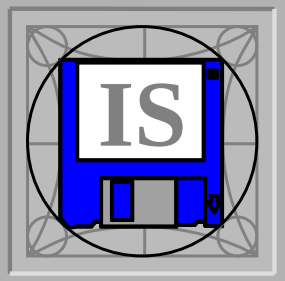
Inspection: attività

- Passo 1: pianificazione
- Passo 2: definizione lista di controllo
  - Cosa vada verificato selettivamente
- Passo 3: lettura
- Passo 4: correzione dei difetti
  - A carico degli autori
- Ogni passo documenta attività svolte e risultanze



Inspection vs. *walkthrough*

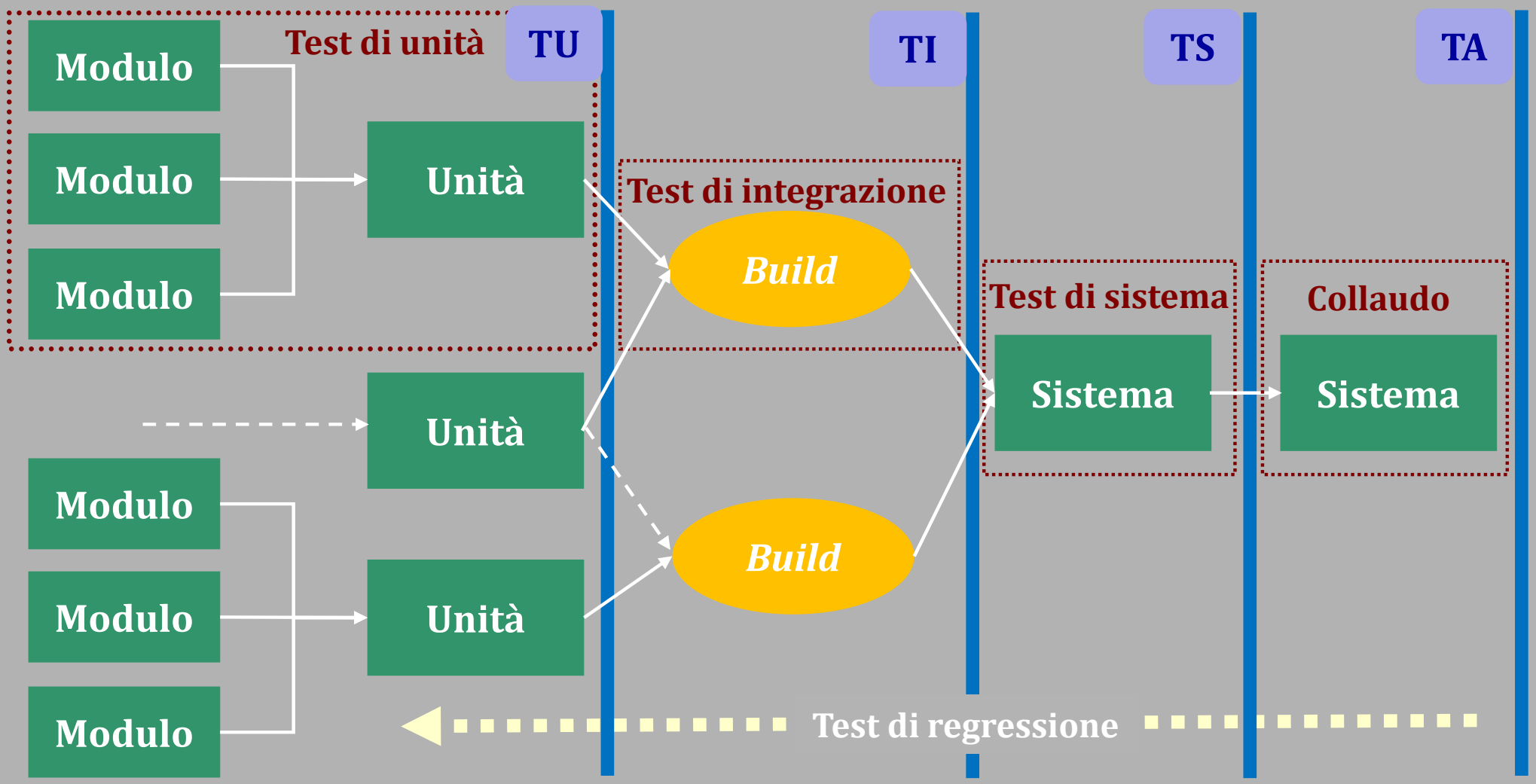
- **Affinità**
  - **Entrambe basate su *desk check***
  - **Verificatori e autori su fronti opposti**
  - **Documentazione rigorosa di attività ed esiti**
- **Differenze**
  - ***Inspection* si basa su presupposti**
    - Dove invece *Walkthrough* richiede maggiore attenzione
    - Ma i presupposti devono essere ben fondati
  - ***Inspection* è più rapido**
    - Mentre *Walkthrough* è più collaborativo

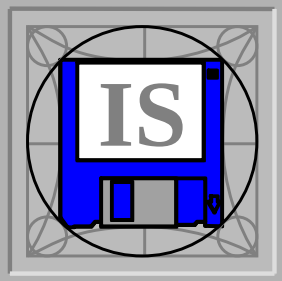


Analisi dinamica: ambiente di prova

- I *test* devono essere ripetibili: per questo specificano
  - Ambiente d'esecuzione: HW/SW, stato iniziale
  - Attese: ingressi richiesti, uscite ed effetti attesi
  - Procedure: esecuzione, analisi dei risultati
- I *test* vanno automatizzati: per questo usano strumenti
  - *Driver* componente attiva fittizia per pilotare il *test*
  - *Stub* componente passiva fittizia per simulare la parte del sistema utile al *test* ma non oggetto di esso (vedere alla voce: «Per approfondire #21»)
  - *Logger* componente non intrusivo di registrazione dei dati di esecuzione per analisi dei risultati

Analisi dinamica: tipi di *test*





Glossario

- **Unità**
  - La più piccola quantità di SW che sia utilmente sottoponibile a verificare individuale
  - Tipicamente prodotta da un singolo programmatore
  - Va intesa in senso architetturale: non linee di codice ma entità di organizzazione logica
    - Singola procedura, singola classe, piccolo aggregato (*package*)
- Il **modulo** (come determinato dal linguaggio di programmazione) è una frazione dell'unità
- Il **componente** integra più unità correlate e coese

Esempio

```
procedure Main is
   …
begin
   …
   Compute(…)
   …
end;
```

Programma

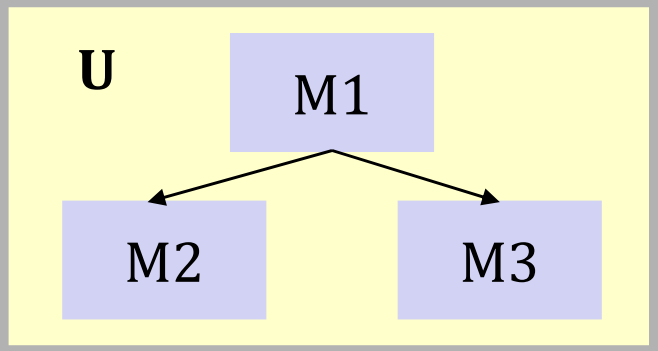
Unità

```
procedure Compute (…, Result : out Integer) is
   Intermediate : Integer := 0;
begin
   …
   Intermediate := Initialize (…);
   …
   Elaborate (Intermediate);
   …
   Result := Commit (Intermediate);
end;
```

Modulo

Stub e *driver*

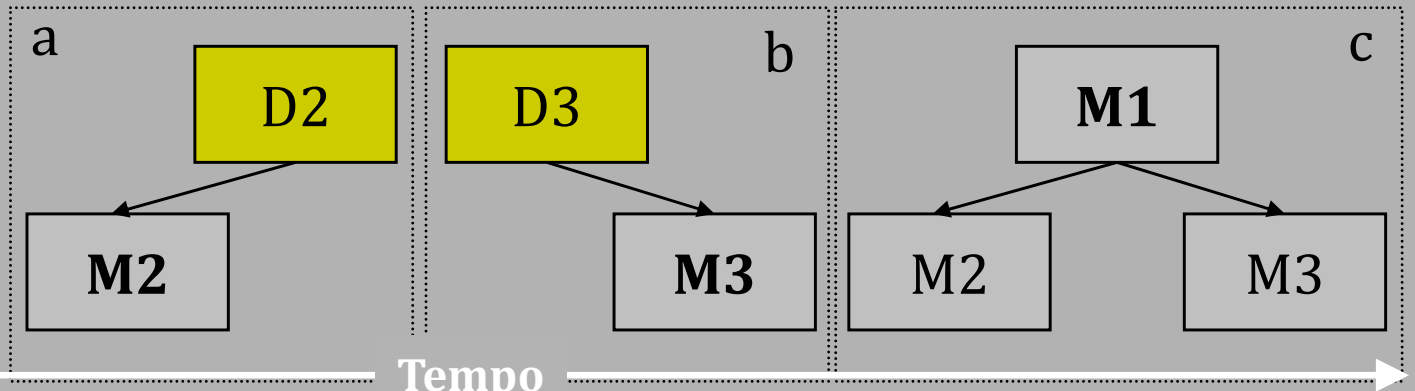
Unità U composta dai moduli M1, M2, M3



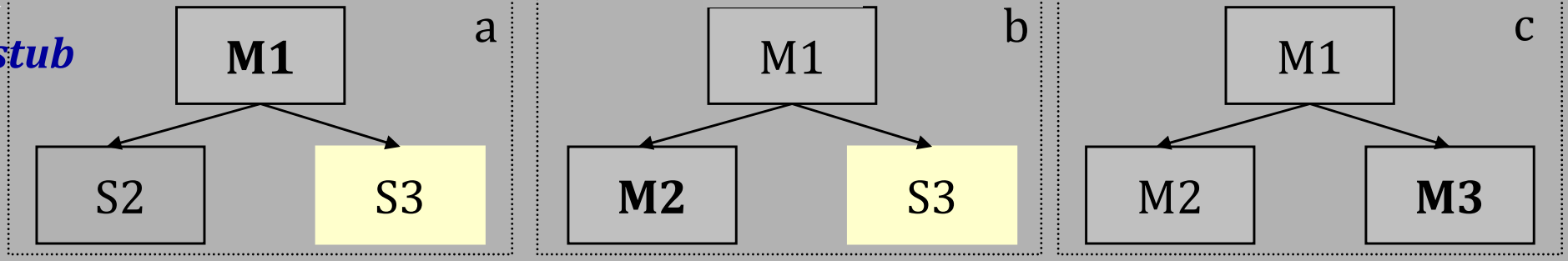
La direzione degli archi indica la relazione d'uso (chi usa chi)

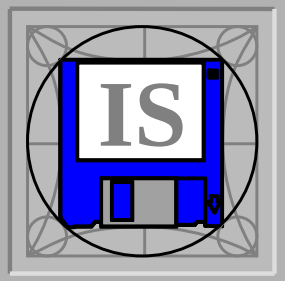
Test di unità su U

Con *driver*



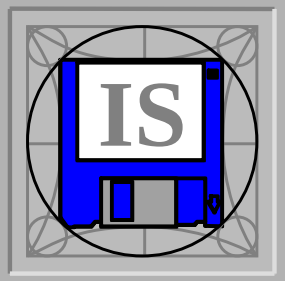
Con *stub*





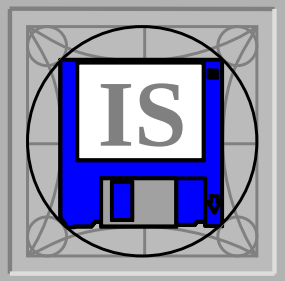
Test di unità

- È agevolato da attività mirate di analisi statica
  - Per determinare limiti di iterazioni, flusso di esecuzione, valori di variabili, ecc.
- Si svolge con il massimo grado di parallelismo e automazione
  - Poiché i TU sono tanti (più sono meglio è …)
- Per le unità più semplici, può essere responsabilità del loro stesso autore
  - Altrimenti, assegnato a verificatore indipendente
- Deve accertare la correttezza del codice *as implemented*
  - Il *test* non può adattare il sorgente del codice cui si applica



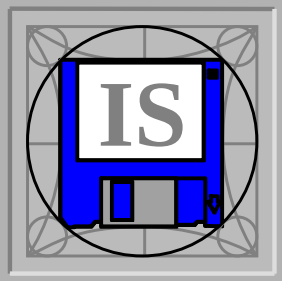
La risoluzione dei problemi

- La verifica serve per scovare problemi e risolverli tempestivamente
- La soluzione dei problemi attiene al processo di supporto «*problem resolution*» di ISO/IEC 122017, che si occupa di
  - Sviluppare una strategia di gestione dei problemi
  - Registrare ogni problema rilevato e classificarlo in uno storico
  - Analizzare ogni problema e determinare soluzioni accettabili
  - Realizzare la soluzione scelta
  - Verificare l'esito della correzione (vedi: Test di Regressione)
  - Assicurare che tutti i problemi noti siano sotto gestione



Test di regressione

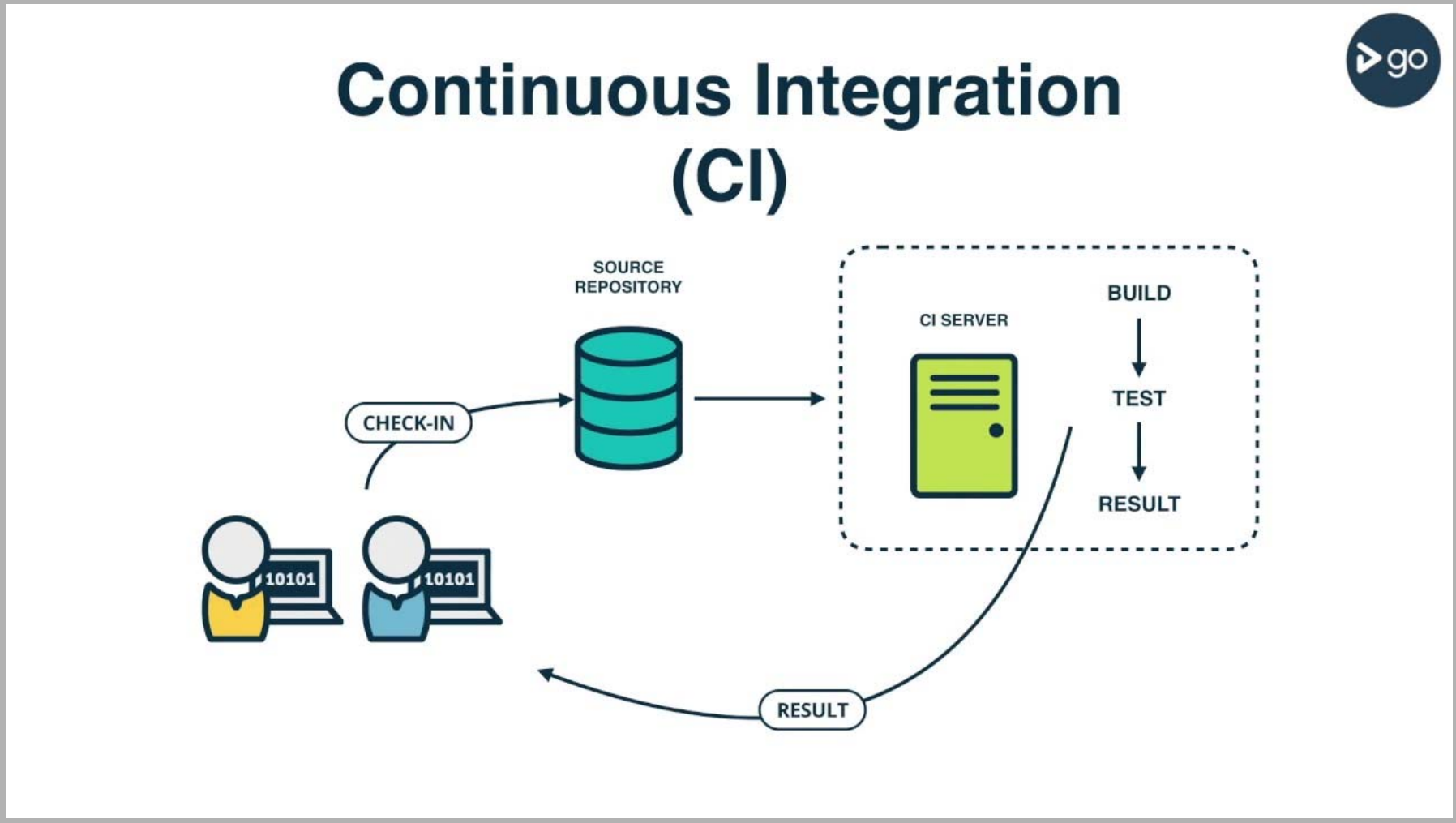
- Modifiche effettuate per aggiunta, correzione o rimozione, non devono pregiudicare le funzionalità già verificate, causando regressione
  - Il rischio aumenta all'aumentare dell'accoppiamento e al diminuire dell'incapsulazione
- Il *test* di regressione comprende tutti i *test* necessari ad accertare che la modifica di una parte P di S non causi errori in P, in S, o in ogni altra parte del sistema che sia in relazione con S
  - Ripetendo *test* già specificati e già eseguiti

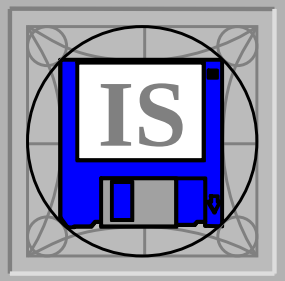


Test di integrazione – 1

- Per costruzione e verifica <u>incrementale</u> del sistema
  - Componenti sviluppati in parallelo e verificati incrementalmente
  - La *build* incrementale è automatizzabile
  - In condizioni ottimali l'integrazione è priva di problemi
- Quali problemi rileva
  - Errori residui nella realizzazione dei componenti
  - Modifica delle interfacce o cambiamenti nei requisiti
  - Riuso di componenti dal comportamento oscuro o inadatto
  - Integrazione con altre applicazioni non ben conosciute

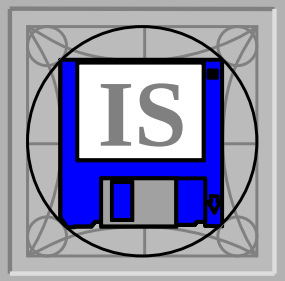
Test di integrazione – 2





Test di sistema e collaudo

- **Validazione**
  - ***Test* di sistema come attività interna del fornitore**
    - Per accertare la copertura dei requisiti SW
    - Propedeutico al collaudo
  - **Collaudo come attività supervisionata dal committente**
    - Per dimostrazione di conformità del prodotto sulla base di casi di prova specificati nel o implicati dal contratto
- **Implicazioni contrattuali**
  - **Il collaudo è attività formale di fronte al committente**
    - Il rilascio del prodotto (e l'eventuale fine del progetto) consegue al buon fine del collaudo



Riferimenti

- Standard for Software Component Testing, British Computer Society SIGIST, 1997
- M.E. Fagan, Advances in Software Inspection, *IEEE Transaction on Software Engineering*, luglio 1986
- G.A. Cignoni, P. De Risi, “Il test e la qualità del software”, Il Sole 24 Ore, 1998