1. Verifica e Validazione: Concetti Introduttivi

1.1 Definizioni

La **Verifica e Validazione** (V&V o "qualifica") sono due processi strettamente correlati che assicurano la qualità del prodotto software durante il suo ciclo di vita. Rispondono a due domande fondamentali:

- **Verifica**: "*Did I build the system right?*" (Ho costruito correttamente il sistema?)
- **Validazione**: "*Did I build the right system?*" (Ho costruito il sistema giusto?)

Verifica

La verifica attiene alla **coerenza, completezza e correttezza** del prodotto. È un processo che si applica ad ogni segmento temporale del progetto (ad ogni prodotto intermedio) per accertare che le attività svolte non abbiano introdotto errori.

Caratteristiche principali:

- Si applica a ogni fase del ciclo di vita
- Può essere eseguita internamente all'organizzazione
- È un processo analitico e sistematico
- Identifica difetti e anomalie durante lo sviluppo

Validazione

La validazione è una **conferma finale** che accerta la conformità del prodotto alle attese. Fornisce una prova oggettiva di come le specifiche del prodotto siano conformi al suo scopo e alle esigenze degli utenti.

Caratteristiche principali:

- Non si applica a un particolare segmento temporale
- Coinvolge sempre il committente
- È una self-fulfilling prophecy
- Conferma che il prodotto soddisfi i requisiti utente

---

2. Forme di Verifica

La verifica è un processo analitico che può essere eseguito in **due forme principali**:

2.1 Analisi Statica

- **Esecuzione**: senza eseguire il software
- **Applicabilità**: già a frammenti di prodotto
- **Momento**: prima dell'analisi dinamica
- **Target**: ogni prodotto intermedio (non solo software)

2.2 Analisi Dinamica

- **Esecuzione**: eseguendo il software o una sua parte
- **Requisito**: necessita di software eseguibile
- **Metodo**: tramite test (prove)
- **Momento**: dopo l'analisi statica

Sequenza temporale: L'analisi statica viene sempre fatta **prima** dell'analisi dinamica, poiché quest'ultima necessita di una parte di software già eseguibile.

---

3. Test: Strumenti e Tipologie

3.1 Caratteristiche dei Test

Un test, per essere utile, deve essere **facilmente ripetibile**. Per questo occorre:

1. **Tenere conto dell'ambiente** (non solo dell'input)
2. **Specificare**:
   - Pre-condizioni
   - Comportamenti attesi
3. **Descrivere**:
   - Procedure per eseguire il test
   - Procedure per analizzare i risultati

3.2 Strumenti per i Test

Logger

Scrive l'esito della prova in un file per tracciabilità e analisi.

Driver

Componente attiva fittizia che "pilota" l'esecuzione del test. Ogni test di unità deve essere chiamato da qualcuno: il driver simula il chiamante della procedura testata (contribuisce al fan-in).

Stub

Componente passiva fittizia che sostituisce del codice non ancora scritto. Lo stub simula le dipendenze della procedura testata (quelle che ne accrescono il fan-out).

Nota: Driver e Stub sono duali: il primo simula i chiamanti, il secondo simula le dipendenze.

3.3 Tipologie di Test

I test si distinguono in base al **livello architetturale** in cui vengono eseguiti:

Tipo di Test	Livello	Quando si stabilisce	Responsabilità
Test di Unità	Unità (più piccola quantità di SW verificabile)	Progettazione di dettaglio	Programmatore/Verificatore/Automa
Test di Integrazione	Componente	Progettazione logica	Team di verifica
Test di Sistema	Sistema completo	Analisi dei requisiti	Organizzazione interna
Collaudo (Test di Accettazione)	Sistema completo	Analisi del capitolato	Committente

Test di Unità

- Verifica la correttezza del codice a livello di unità
- Eseguiti con alto grado di parallelismo
- Possono essere automatizzati

Test di Integrazione

- Verificano il sistema in modo incrementale
- Integrano il funzionamento di più unità
- Stanno a livello di componente

Test di Sistema

- Accertano la copertura dei requisiti
- Attività interna all'organizzazione
- Precede il collaudo

Collaudo

- Supervisione del committente
- Seguito da rilascio del prodotto

- Fine della commessa (con eventuale manutenzione)

3.4 Test di Regressione

A seguito di un test fallito e di una correzione, i **test di regressione** sono l'insieme di test necessari ad accertare che la correzione non causi errori nelle parti del sistema che dipendono da essa.

Caratteristiche:

- Possono essere complessi
- Devono essere studiati ad hoc
- Garantiscono che le modifiche non introducano nuovi difetti

---

4. Quality Assurance

4.1 Definizione

L'**analisi statica** e l'**analisi dinamica** sono parte dell'attività di **quality assurance** (garanzia di qualità).

Principi fondamentali:

- È un controllo che viene fatto **prima** (non dopo)
- Assicura la qualità tempestivamente
- Viene svolta sui **processi** (non sul prodotto stesso)
- Si basa su un modello di qualità di prodotto (es. ISO/IEC 9126)

4.2 Obiettivi

La quality assurance mira a:

- Prevenire i difetti anziché rilevarli tardivamente
- Ridurre i costi di correzione
- Migliorare la qualità complessiva del processo
- Garantire conformità agli standard

---

5. Analisi Statica: Fondamenti

5.1 Premessa

Un software di qualità deve possedere:

1. **Capacità funzionali** (qualità esterna)
   - Specificano cosa il sistema debba fare
   - Definite nei requisiti
2. **Caratteristiche non funzionali** (qualità interna)
   - Necessarie al buon funzionamento del sistema
   - Prestazioni, robustezza, sicurezza

5.2 Proprietà da Verificare

L'analisi statica accerta il possesso di svariate proprietà:

- **Di costruzione**: architettura, codifica, integrazione
- **D'uso**: esperienza utente, precisione, affidabilità
- **Di funzionamento**: prestazioni, robustezza, sicurezza

5.3 Compromesso tra Espressività e Verificabilità

Principio fondamentale: Tanto più un linguaggio di programmazione è espressivo, tanto meno è verificabile.

Implicazioni pratiche:

- Trovare il giusto compromesso tra funzionalità (potere espressivo) e integrità (costo di verifica)
- Adottare uno **standard di codifica** che tenga conto delle esigenze di verifica
- Vietare alcuni costrutti per facilitare la verifica

5.4 Verifica Accompagnata alla Produzione

Il costo di rilevazione e correzione di un errore è tanto maggiore quanto più avanzato è lo stadio di sviluppo.

Ecco perché è importante:

- Accompagnare la produzione con la verifica
- Non posticipare la verifica il più tardi possibile
- Rendere possibile una verifica non retrospettiva

5.5 Considerazioni Pragmatiche

L'efficacia dei metodi di verifica è funzione della qualità di strutturazione del codice.

Esempi:

- Una procedura con un **singolo punto di uscita** facilita l'analisi del suo effetto sullo stato del sistema
- La **verificabilità** è funzione inversa dell'ampiezza del contesto oggetto di verifica

- Confinare **scope e visibilità** aumenta la verificabilità
- Una buona **architettura** facilita la verifica tramite:
  - Incapsulamento dello stato
  - Controllo di accesso
  - Separazione tra interfaccia e implementazione

---

6. Metodi di Lettura

L'analisi statica si applica ad ogni prodotto intermedio per tutti i processi attivi nel progetto. Si distinguono due tecniche principali:

6.1 Classificazione

1. **Metodi di lettura (desk check)**
   - Svolti da umani
   - Controllano coerenza, completezza e correttezza
   - Impiegati per prodotti semplici
2. **Metodi formali**
   - Svolti da macchine
   - Basati su prova assistita di proprietà
   - Utili quando la dimostrazione empirica ha costo proibitivo

6.2 Inspection

Definizione: Lettura mirata alla ricerca di errori noti.

Caratteristiche

- **Obiettivo**: Rilevare la presenza di difetti tramite lettura mirata
- **Agenti**: Verificatori distinti e separati dai programmatori
- **Strategia**: Focalizzare la ricerca su presupposti (errori noti)
- **Metodo**: Basato sull'individuazione di errori presupposti

Attività (4 fasi)

1. **Pianificazione**
2. **Definizione di una lista di controllo** (checklist)
   - La lista evolve nel tempo
   - Viene continuamente migliorata
   - Cosa va verificato selettivamente
3. **Lettura** del codice/prodotto
4. **Correzione dei difetti**

    - A carico degli autori

Ogni fase richiede documentazione (rapporto delle attività).

Vantaggi e Limiti

- ✅ **Più rapida** del walkthrough
- ✅ Efficace per errori noti
- ❌ Può generare **falsi negativi** (errori non presupposti sfuggono)
- ❌ Non esaustiva (dipende dalla completezza della lista)

6.3 Walkthrough

Definizione: Lettura critica "a largo spettro" del prodotto in esame.

Caratteristiche

- **Obiettivo**: Rilevare la presenza di difetti tramite lettura critica
- **Agenti**: Gruppi misti verificatori/sviluppatori (ruoli distinti)
- **Strategia**: Esame privo di assunzioni o presupposti
    - Per il codice: percorrerlo simulandone possibili esecuzioni
    - Per i documenti: studiarne ogni parte come farebbe un compilatore
- **Metodo**: "Attraversamento a pettine" - non so cosa cerco, ma cerco ovunque

Attività (4 fasi)

1. **Pianificazione**
2. **Lettura** critica
3. **Discussione** delle risultanze
4. **Correzione dei difetti**

Ogni fase richiede documentazione (rapporto delle attività).

Vantaggi e Limiti

- ✅ **Esaustività** maggiore (ricerca a campo aperto)
- ✅ Può scoprire errori inaspettati
- ❌ Richiede **più attenzione** dell'inspection
- ❌ Può generare **falsi positivi**
- ❌ **Più lento** dell'inspection
- ✅ **Più collaborativo**

6.4 Confronto: Inspection vs Walkthrough

Aspetto	Inspection	Walkthrough
Approccio	Ricerca mirata	Ricerca aperta
Base	Presupposti (errori noti)	Esperienza e intuizione
Velocità	Più rapido	Più lento
Attenzione	Focalizzata	Richiede più attenzione
Collaborazione	Meno collaborativo	Più collaborativo
Rischi	Falsi negativi	Falsi positivi
Completezza	Dipende dalla lista	Più esaustivo

Le due tecniche si completano a vicenda e sono entrambe necessarie per una verifica efficace.

6.5 Efficacia

L'efficacia dei metodi di lettura dipende dall'**esperienza dei verificatori**:

- Nell'organizzare le attività da svolgere
- Nel documentare le risultanze

---

7. Tracciamento

7.1 Definizione

Il **tracciamento** è una parte fondamentale dell'analisi statica. È una verifica atta a dimostrare due caratteristiche della soluzione.

7.2 Proprietà Dimostrate

Completezza della Soluzione

- Tutti i requisiti sono soddisfatti
- **Matematicamente**: la soluzione è **condizione sufficiente** per il problema
- Nessun requisito viene trascurato

Economicità della Soluzione

- Nessuna funzionalità superflua
- Nessun componente ingiustificato
- **Matematicamente**: la soluzione è **condizione necessaria** per il problema
- Nessun elemento in eccesso

7.3 Applicazione nel Ciclo di Vita

Il tracciamento ha luogo in due momenti:

1. **Su ogni passaggio dello sviluppo** (ramo discendente del modello a "V")
   - Dal capitolato all'analisi
   - Dall'analisi alla progettazione
   - Dalla progettazione al codice
2. **Su ogni passaggio della verifica** (ramo ascendente del modello a "V")
   - Dal codice ai test di unità
   - Dai test di unità ai test di integrazione
   - Dai test di integrazione ai test di sistema
   - Dai test di sistema al collaudo

7.4 Rappresentazione

Il tracciamento viene documentato attraverso:

- **Matrici delle dipendenze**
- **Documentazione di tracciamento**
- Collegamenti bidirezionali tra requisiti e implementazione

7.5 Obiettivi

Il tracciamento permette di:

- Identificare eventuali **requisiti derivati** (aggiuntivi)
- Investigare il loro soddisfacimento
- Verificare che ogni requisito sia implementato
- Verificare che ogni componente sia giustificato
- Facilitare la manutenzione e l'evoluzione

7.6 Automazione

Il tracciamento:

- Deve essere **automatizzato** quando possibile
- Viene creato durante lo sviluppo
- Serve in ogni passaggio della verifica
- Riduce errori e omissioni

---

8. Tipologie di Analisi Statica

L'analisi statica si effettua con **diversi metodi**, ciascuno con obiettivi specifici. L'analisi statica costruisce **modelli astratti** del software in esame, considerando ogni programma come un **grafo orientato** e studiandone i cammini possibili.

8.1 Analisi di Flusso di Controllo

Obiettivi:

- Accertare che il codice esegua nella sequenza attesa
- Verificare che il codice sia ben strutturato
- Localizzare **codice non raggiungibile** sintatticamente
- Identificare parti del codice che possano porre problemi di terminazione:
  - Chiamate ricorsive
  - Iterazioni infinite

Approccio: Modella il programma come grafo di controllo e analizza i percorsi di esecuzione.

8.2 Analisi di Flusso dei Dati

Obiettivi:

- Accertare che nessun cammino d'esecuzione acceda a **variabili non valorizzate**
- Concentrare l'analisi sulle sequenze di accesso alle variabili
- Modalità di accesso: lettura, scrittura

Anomalie rilevate:

- Più scritture successive senza letture intermedie
- Letture che precedano scritture
- Presenza di variabili globali
- Violazioni al principio di incapsulamento

Nota: L'analisi è complicata dalla presenza di dati globali accessibili da ogni parte del programma.

8.3 Analisi di Flusso dell'Informazione

Obiettivi:

- Determinare come l'esecuzione di un'unità di codice crei dipendenze tra ingressi e uscite
- Identificare **effetti laterali** inattesi o indesiderati

Vincoli:

- Le sole dipendenze consentite sono quelle previste dalla specifica
- Può essere applicata a:

- Un singolo modulo
- Più moduli collegati
- L'intero sistema

8.4 Esecuzione Simbolica

Definizione: Verifica proprietà del programma mediante **manipolazione algebrica** del codice sorgente.

Caratteristiche:

- Non richiede specifica formale
- Usa tecniche di analisi di flusso (controllo, dati, informazione)
- Si esegue effettuando "sostituzioni inverse"

Risultato: Trasforma il flusso sequenziale del programma in un insieme di **assegnamenti paralleli** nei quali i valori di uscita sono espressi come funzione diretta dei valori di ingresso.

8.5 Verifica Formale del Codice

Obiettivi:

- Provare la **correttezza** del codice sorgente rispetto alla specifica formale dei requisiti
- Esplorare **tutte le esecuzioni possibili**

Vantaggio: Permette di fare ciò che non è fattibile mediante prove dinamiche (copertura completa).

Requisiti: Necessita di una specifica formale dei requisiti.

8.6 Verifica di Limite

Obiettivi:

- Verificare il comportamento del sistema ai valori limite
- Controllare gestione di overflow, underflow
- Validare condizioni al contorno

8.7 Analisi d'Uso dello Stack

Obiettivi:

- Analizzare l'utilizzo della memoria stack
- Prevenire stack overflow
- Ottimizzare l'uso della memoria

8.8 Analisi Temporale

Obiettivi:

- Verificare i requisiti temporali (real-time)
- Analizzare worst-case execution time (WCET)
- Garantire rispetto delle deadline

8.9 Analisi d'Interferenza

Obiettivi:

- Rilevare interferenze in sistemi concorrenti
- Identificare race condition
- Verificare sincronizzazione tra thread/processi

8.10 Analisi del Codice Oggetto

Obiettivi:

- Verificare il codice compilato/bytecode
- Controllare ottimizzazioni del compilatore
- Validare il codice generato

Considerazioni Finali

Importanza della Verifica e Validazione

La V&V è essenziale per:

- Garantire la qualità del software
- Ridurre i costi di correzione
- Aumentare la fiducia nel prodotto
- Soddisfare le aspettative del committente

Principi Chiave

1. **Verifica precoce**: Più tardi si rileva un errore, più costa correggerlo
2. **Approccio sistematico**: Utilizzare metodi complementari (inspection + walkthrough)
3. **Automazione**: Automatizzare dove possibile (tracciamento, test)
4. **Documentazione**: Ogni attività di verifica deve essere documentata
5. **Quality assurance**: Prevenire è meglio che correggere

Buone Pratiche

- Separare interfaccia e implementazione

- Massimizzare l'incapsulamento
- Adottare standard di codifica verificabili
- Utilizzare liste di controllo evolutive
- Combinare analisi statica e dinamica
- Tracciare requisiti e implementazione
- Documentare tutte le attività di verifica

Glossario Essenziale

- **Verifica**: Processo per accertare l'assenza di errori in ogni fase
- **Validazione**: Conferma finale della conformità alle attese
- **Analisi Statica**: Verifica senza esecuzione del software
- **Analisi Dinamica**: Verifica tramite esecuzione (test)
- **Inspection**: Lettura mirata alla ricerca di errori noti
- **Walkthrough**: Lettura critica a largo spettro
- **Tracciamento**: Verifica di completezza ed economicità
- **Quality Assurance**: Garanzia di qualità preventiva
- **Driver**: Componente che pilota l'esecuzione di un test
- **Stub**: Componente che simula dipendenze non ancora implementate
- **Desk Check**: Metodo di lettura umana del codice
- **Lista di Controllo**: Elenco evolutivo di elementi da verificare

---