Analisi statica e verifica del software

Simone Colli
Manuel Di Agostino
`authoremail`

Appunti del corso tenuto dal **Prof. Vincenzo Arceri**

Università degli Studi di Parma
Anno Accademico 2025/2026

Indice

1 Introduzione

1.1 Cos'è l'affidabilità del software?

Definizione 1.1: Affidabilità del Software (IEEE 610.121990)

"The ability of a system or component to perform its required functions under stated conditions for a specified period of time"
Ovvero:
"La capacità di un sistema o componente di eseguire le funzioni richieste in condizioni specificate per un periodo di tempo specificato"

Definizione 1.2: Gestione dell'Affidabilità del Software (IEEE 982.11988)

"The process of optimizing the reliability of software through a program that emphasizes software error prevention, fault detection and removal, and the use of measurements to maximize reliability in light of project constraints such as resources, schedule and performance"
Ovvero:
"Il processo di ottimizzazione dell'affidabilità del software attraverso un programma che enfatizza la prevenzione degli errori software, il rilevamento e la rimozione dei guasti, e l'uso di misurazioni per massimizzare l'affidabilità alla luce dei vincoli del progetto come risorse, tempi e prestazioni"

Sfruttando le definizioni 1.1 e 1.2, è possibile riassumere che l'affidabilità del software consiste in 3 attività:

- Prevenzione degli errori.
- Rilevamento e rimozione dei guasti.
- Valutazione per massimizzare l'affidabilità, in particolare valutazioni a supporto delle prime due attività.

1.2 Perché l'affidabilità del software è importante?

L'importanza dell'affidabilità del software deriva dalle enormi conseguenze economiche e operative che i bug possono causare.

Alcune stime indicano che:

- I bug nei software costano circa 60 miliardi di dollari all'anno negli Stati Uniti.
- L'economia mondiale perde circa 250 miliardi di dollari all'anno a causa di qualsiasi tipo di attacco palese (overt attack).
- I difetti nel software rendono la programmazione molto dolorosa.

Alcuni esempi di fallimenti dovuti a bug software includono:

- Il disastro del volo 501 di Ariane 5 nel 1996, che è esploso dopo 37 secondi dal decollo a causa di un errore di overflow durante la conversione da un float 64 bit ad un intero 16 bit. Questo evento ha causato una perdita di circa 370.000.000 dollari.
- Il bug del Pentium FDIV nel 1994, che ha causato errori di calcolo nelle divisioni in virgola mobile. Questo bug ha portato a una perdita di circa 475.000.000 dollari.
- L'aggiornamento difettoso di CrowdStrike nel 2024, che ha causato più di 5000 voli cancellati. Questo evento ha impattato su banche, governi e infrastrutture critiche. Le perdite economiche sono state stimate a circa 5.400.000.000 dollari.

La lista di esempi potrebbe continuare, ma l'importante è comprendere che l'affidabilità del software è cruciale per evitare perdite economiche significative e garantire il corretto funzionamento dei sistemi.

1.3 Validazione e Verifica

La validazione e la verifica sono due processi distinti ma complementari nell'ambito dello sviluppo del software, entrambi mirati a garantire che il software soddisfi i requisiti e funzioni correttamente.

La **validazione** si concentra sulla domanda: "Stiamo costruendo il prodotto giusto?". Essa verifica che il software soddisfi le esigenze e le aspettative degli utenti finali. La **verifica**, d'altra parte, si concentra sulla domanda: "Stiamo costruendo il prodotto nel modo giusto?". Essa assicura che il software sia sviluppato correttamente secondo le specifiche tecniche e i requisiti di progetto.

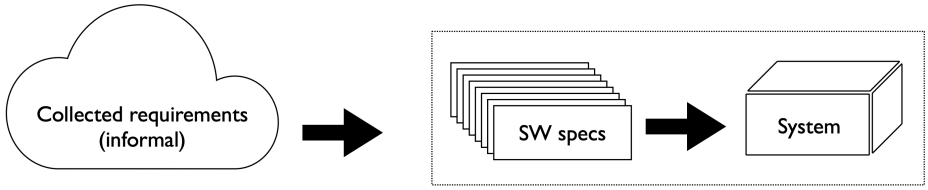

Figura 1: L'immagine illustra la differenza tra validazione e verifica.

Esempio 1.1: Risposta dell'ascensore

Un esempio pratico di validazione e verifica può essere trovato nel comportamento di un sistema di risposta dell'ascensore.
Una specifica validabile ma non verificabile potrebbe essere: "Se un utente preme il pulsante di richiesta a un piano i, un ascensore disponibile deve arrivare al piano i entro un tempo ragionevole".
Una specifica verificabile è: "Se un utente preme il pulsante di richiesta a un piano i, un ascensore disponibile deve arrivare al piano i entro 30 minuti".

1.4 Verifica del Software

Lo scopo della verifica del software è verificare alcune affermazioni di correttezza riguardante un programma:

- La **correttezza funzionale**, ovvero che il programma soddisfi i requisiti funzionali specificati.
- L'**assenza di errori non funzionali** (non cosa fa il programma, ma come lo fa), come ad esempio: soddisfacimento di vincoli spaziali e/o temporali, assenza di errori a runtime, soddisfacimento di vincoli di sicurezza, ecc.

Bad news: theres no silver bullet

Non esiste un metodo di verifica che sia contemporaneamente:

- **Automatico**: non richiede interazione umana.
- **Potente**: è in grado di dimostrare proprietà non banali (*non-trivial properties*).
- **Corretto (Sound)**: non dimostra mai una proprietà se questa non è vera.
- **Completo (Complete)**: dimostra sempre la proprietà se essa è vera e non fallisce mai nel dichiarare la correttezza di un programma corretto.

Il **teorema di Rice** afferma che tutte le proprietà non banali del comportamento di un programma scritto in un linguaggio di programmazione Turing-completo sono indecidibili. L'**indecidibilità** di una proprietà implica che non esiste alcun metodo di verifica automatico che sia allo stesso tempo **corretto** e **completo**.

2 Background matematico

Per poter discutere in modo rigoroso di tecniche di verifica del software, è necessario introdurre alcuni concetti matematici di base relativi alla teoria della computazione e alla logica formale.

2.1 Set notation

2.2 Partial order

2.3 Powerset

2.4 Diagramma di Hasse

2.5 Upper e lower bounds

2.6 Lattice