***Descrivere cosa si intende per affidabilità di un sistema software e con quali tecniche è possibile procedere per cercare di migliorala.***  
  
L'affidabilità di un sistema software è una delle sue qualità desiderabili e si riferisce alla capacità del software di **operare come atteso in un intervallo di tempo determinato**. Informalmente, un software è considerato affidabile nella misura in cui l'utente può fare affidamento sulle sue funzionalità. È importante distinguere l'affidabilità da concetti correlati come la **correttezza** e la **robustezza**.

* La **correttezza** è una qualità assoluta: un programma è funzionalmente corretto se si comporta secondo quanto stabilito dalle sue specifiche funzionali. Ogni deviazione dalla specifica rende un sistema non corretto, indipendentemente dalla gravità delle conseguenze.
* L'**affidabilità** è invece una nozione relativa: un software non corretto può comunque essere considerato affidabile se la conseguenza di un errore non è grave. L'insieme dei programmi affidabili include l'insieme dei programmi corretti, ma non viceversa. Tuttavia, in pratica, le specifiche potrebbero non essere descrizioni accurate dei requisiti effettivi dell'utente, rendendo possibile che un'applicazione corretta secondo la specifica non si comporti come desiderato dall'utente.
* La **robustezza** di un programma si manifesta quando si comporta in modo accettabile anche in circostanze non previste dalle specifiche, come l'inserimento di dati errati o malfunzionamenti hardware. Un programma può essere corretto ma non robusto se la specifica non contempla come gestire input scorretti. La robustezza e la correttezza sono strettamente correlate, con la linea di demarcazione tra le due data dalla specifica del sistema.

Per migliorare l’affidabilità del software, è necessario adottare un approccio ingegneristico basato su più tecniche integrate. Un primo passo è l’uso di **rigore e formalità**: un metodo strutturato consente di controllare costi e qualità. Anche una **buona documentazione** contribuisce all’affidabilità, pur non essendo formalmente rigorosa.

**Verifica e validazione** permettono di assicurare che il sistema rispetti i requisiti. Migliorano con strumenti come software monitor, progettazione modulare e linguaggi adeguati. L'obiettivo è un prodotto verificabile già a partire dai requisiti del cliente.

L’analisi dinamica, cioè il **testing**, è fondamentale per rilevare malfunzionamenti. Si parte con test di modulo per verificare singoli componenti, si passa ai test di integrazione (spesso incrementali) e si arriva ai test di sistema, che includono anche alpha e beta test. Le attività di **manutenzione** richiedono test di regressione per assicurare che correzioni o modifiche non introducano nuovi difetti. Il **debugging** completa il ciclo, localizzando e correggendo gli errori trovati nei test.

La **modularità** aiuta a gestire la complessità suddividendo il sistema in parti indipendenti e coese. Moduli con alta coesione e basso accoppiamento sono più facili da analizzare, testare e modificare, migliorando così la **riparabilità**, un fattore strettamente legato all’affidabilità. L’**astrazione** consente ai progettisti di concentrarsi sugli aspetti essenziali di un problema, ignorando i dettagli secondari. Questo facilita la comprensione del sistema e favorisce soluzioni più efficaci.

Poiché il software è soggetto a continue modifiche, è fondamentale progettarlo tenendo conto della sua evoluzione. La manutenibilità comprende la capacità di riparare (riparabilità) e di evolvere (evolvibilità). Anticipare il cambiamento confinandone gli effetti a porzioni ben definite del sistema è una strategia chiave per mantenere l’affidabilità nel tempo.

Infine, l’affidabilità è strettamente legata ai processi di sviluppo utilizzati. Modelli evolutivi come il **rilascio incrementale** o la **prototipazione**, in contrasto con il rigido modello a cascata, consentono di ottenere feedback rapido e di adattarsi ai requisiti instabili. Le metodologie agili, come l’Extreme Programming, valorizzano l’adattabilità ai cambiamenti. Il modello a spirale di Boehm introduce la gestione del rischio come elemento centrale, affrontando in anticipo le criticità che potrebbero compromettere l’affidabilità complessiva del sistema.

***Quando si parla di validazione del software, cosa si intende con i termini “revisione” e “walkthrough”?***

Quando si parla di validazione del software, i termini "revisione" (o "ispezione") e "walk-through" si riferiscono a tecniche di **analisi statica del codice sorgente** utilizzate per la verifica della qualità del software. Il **walk-through** è un'analisi informale del codice svolta da un gruppo di collaudatori e progettisti che **simulano manualmente l'esecuzione del codice** su alcuni casi di test selezionati. Questa attività cooperativa e organizzata si concentra sulla **scoperta degli errori**, piuttosto che sulla loro eliminazione o sulla proposta di alternative di progettazione. I partecipanti ricevono la documentazione in anticipo e il focus è strettamente sull'esame del codice, non sul codificatore. La **revisione** (o **ispezione del codice**) è simile al walk-through per gli aspetti organizzativi, l'ispezione del codice ha l'obiettivo di **scoprire errori comuni** esaminando il codice (o il progetto), piuttosto che simulandone l'esecuzione. Si basa sulla **ricerca mirata di specifici tipi di errori** predefiniti, come variabili non inizializzate, salti all'interno di cicli, o indici di array fuori dal campo di variabilità. È un mezzo di verifica molto importante e la sua utilità è stata confermata da numerose esperienze pratiche. Entrambe queste tecniche sono complementari al testing dinamico e sono strumenti fondamentali per individuare anomalie (difetti o bug) nel codice che potrebbero causare malfunzionamenti.