Training per esperti

A cura di Matteo Foini

# COME SVILUPPARE CON SUCCESSO

Questo articolo descrive tecniche di sviluppo basate su anni di esperienza nell'ingegneria del software. L'argomento viene introdotto con un livello di teoria adatto per aiutarvi a capire come tali tecniche vi permettono di creare VI scalabili, leggibili e manutenibili

Questo articolo descrive strategie e tecniche di programmazione che potete utilizzare per costruire i vostri VI. Imparerete come evitare l'aggiunta di caratteristiche impreviste che possano alterare l'intento originale dell'applicazione e rendere più difficile la manutenzione del codice. Imparerete anche tecniche di risoluzione dei problemi e come utilizzare in modo ottimale LabVIEW per risolvere problemi.
Esploreremo tutte le fasi del processo di sviluppo del software: progettazione, implementazione, test e messa in opera, come indicato in figura 1. Potete utilizzare queste procedure per costruire VI scalabili, leggibili e manutenibili.

## VI SCALABILI, LEGGIBILI E MANUTENIBILI

Se utilizzate LabVIEW per sviluppare applicazioni complesse, dovreste usare dei buoni principi di progettazione software. Il vostro scopo è sempre quello di creare VI che siano scalabili, leggibili e manutenibili.

• **Scalabile** - Deve essere facile aggiungere nuove funzionalità ad un'applicazione senza riprogettare completamente l'applicazione stessa.

• **Leggibile** - Deve essere facile ispezionare visivamente il progetto di un'applicazione e capirne lo scopo e le funzionalità.

• **Manutenibile** - Deve essere facile cambiare il codice da parte dello sviluppatore originale o di qualsiasi altro sviluppatore senza minare l'intento del codice originale.

Poiché LabVIEW è un linguaggio di programmazione, quando programmate con LabVIEW trovate molti degli stessi problemi di progettazione che incontrate quando programmate coi linguaggi tradizionali basati su testo. Tuttavia, LabVIEW offre molte funzionalità e tecniche potenti di programmazione che vi permettono di concentrarvi sulla stesura di un progetto anziché focalizzarvi, per esempio, su problemi di sintassi o di memoria.

## SCALABILITÀ

Per creare un VI scalabile, dovete iniziare a pensare alla struttura dell'applicazione già nelle prime fasi del processo di progettazione. Un VI scalabile ben progettato vi permette di modificare e aggiungere facilmente funzionalità al progetto originale. Per esempio, consideriamo un VI di

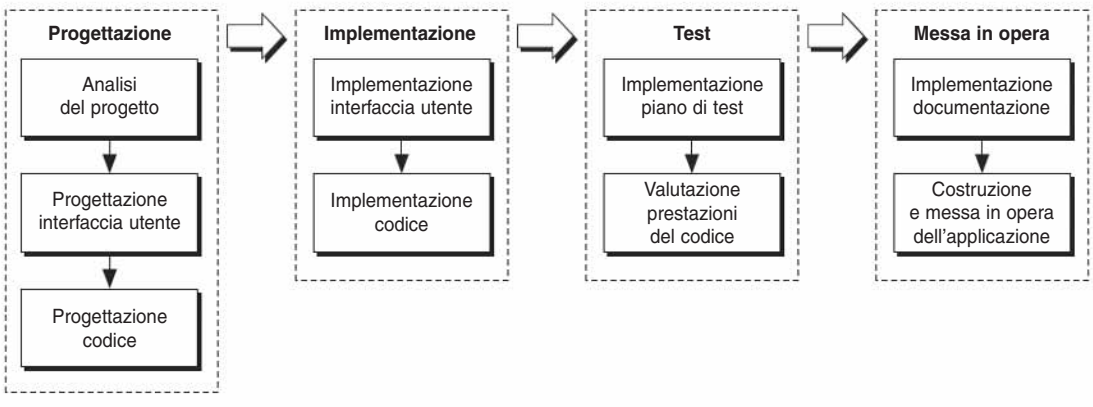


Figura 1 – Mappa di sviluppo del software

acquisizione che acquisisce dati da tre termocoppie. Supponiamo che i requisiti dell'applicazione cambino e che sia necessario acquisire dati da centinaia di termocoppie. Se il VI originale è stato progettato per essere scalabile, estendere il VI per acquisire dati da centinaia di termocoppie sarà più facile che progettare un nuovo VI.
Utilizzate buone pratiche di progettazione per creare VI scalabili. Molte applicazioni esistenti devono essere riscritte quando sono necessari dei cambiamenti, perché il codice non è stato progettato per essere scalabile. Per un VI non scalabile, anche semplici cambiamenti, come acquisire dati da più sensori o controllare più attuatori, possono richiedere una riscrittura. Quando progettate un'applicazione qualsiasi, considerate lo scopo dell'applicazione e come gestire i cambiamenti quando la scala dell'applicazione va oltre le specifiche originali.

## LEGGIBILITÀ

Nella vostra esperienza di lavoro con LabVIEW, vi sarete imbattuti in esempi di schemi a blocchi non ben strutturati, difficili da leggere e difficili da capire. Il codice confuso e non manutenibile viene chiamato a volte "spaghetti code"; tale codice illeggibile può rendere impossibile decifrare le funzionalità di un diagramma a blocchi.

## MANUTENIBILITÀ

Un VI scritto seguendo delle solide basi di progettazione software e avente una solida architettura vi permette di aggiungere nuove funzionalità senza dovere riscrivere completamente l'applicazione.
Quando sviluppate un'applicazione, tenete presente che un altro programmatore potrebbe avere la necessità di usare e modificare il VI in futuro.
L'articolo vi spiega come applicare le funzionalità di LabVIEW e utilizzare buoni principi di progettazione del software, direttamente correlati a LabVIEW, per creare applicazioni ben formate, scalabili, leggibili e manutenibili.

## REGOLE PER SVILUPPARE CON SUCCESSO

LabVIEW facilita l'integrazione di componenti di sistemi di acquisizione dati, test e controllo. Dato che creare applicazioni in LabVIEW è così facile, molte persone iniziano a sviluppare immediatamente VI con una pianificazione relativamente modesta. Per le applicazioni semplici, come test di laboratorio veloci o applicazioni di monitoraggio, questo approccio può essere appropriato. Tuttavia, per i progetti di sviluppo più grossi, una buona pianificazione del progetto è vitale.

## LABVIEW: UN LINGUAGGIO DI PROGRAMMAZIONE

LabVIEW è un linguaggio di programmazione general purpose progettato specificamente per la creazione di applicazioni di misura e automazione. Le applicazioni LabVIEW possono spaziare da un semplice VI a grosse applicazioni, contenenti molti VI organizzati in gerarchie complesse. Mano a mano che espandete l'uso di LabVIEW e create applicazioni più articolate, inevitabilmente, il codice che scrivete diventa più complesso.
Molte industrie in tutto il mondo utilizzano LabVIEW come strumento per eseguire una vasta gamma di operazioni di misura e automazione, molto spesso in ambienti dove la sicurezza è critica: come programmatori, quindi, dovete creare applicazioni che siano sicure, facili da manutenere e facili da capire.

## IL CICLO DI VITA DEL SOFTWARE

Per affrontare le complessità proprie dei grandi progetti di sviluppo del software, molti sviluppatori seguono un nucleo base di principi di sviluppo che definiscono il campo dell'ingegneria del software. Uno dei componenti fondamentali è rappresentato dal modello del ciclo di vita, che descrive i passi da seguire quando si sviluppa del software: dal concetto iniziale alle fasi di rilascio, manutenzione e successivo upgrade del software.
Esistono molti modelli diversi del ciclo di vita. Ciascuno di essi ha i propri vantaggi e svantaggi in termini di tempo di rilascio, qualità e gestione dei rischi.
Questo paragrafo descrive alcuni dei modelli più comuni ma ne esistono molti altri nati da combinazioni dei principali; pertanto, potete personalizzare i modelli esistenti per adattarli ai requisiti di un progetto. Benché la discussione in questa sede rimanga teorica, considerate sempre nella pratica tutte le fasi che questi modelli abbracciano, come decidere quali requisiti e specifiche il progetto debba soddisfare e come affrontare i loro cambiamenti.
Il modello del ciclo di vita è un fondamento dell'intero processo di sviluppo. Buone decisioni possono migliorare la qualità del software che sviluppate e diminuire il tempo necessario per lo sviluppo.

## IL MODELLO "CODE AND FIX"

Il modello "code and fix" è probabilmente la metodologia di sviluppo utilizzata più frequentemente. Si comincia immediatamente a sviluppare, con una pianificazione minima o nulla, sistemando i problemi mano a mano che si presentano, finché il progetto è completo.
Il modello "code and fix" rappresenta la scelta preferita quando la pianificazione prevede tempi stretti di sviluppo, perché vi mette in condizione di scrivere subito codice e vedere dei risultati immediatamente.
Purtroppo, se si scoprono problemi strutturali significativi in una fase avanzata del processo di sviluppo, normalmente è necessario riscrivere grosse parti dell'applicazione. Modelli di sviluppo alternativi possono aiutarvi a scoprire tali problemi già nelle prime fasi, quando eseguire

cambiamenti è più facile e meno costoso. Il modello "code and fix" è appropriato solo per piccoli progetti non destinati a servire come base per sviluppi futuri.

## IL MODELLO A CASCATA

Il modello a cascata è il classico modello dell'ingegneria del software, uno dei più vecchi e ampiamente utilizzato nei progetti delle grandi aziende. Enfatizzando l'importanza della documentazione e della pianificazione fin dalle prime fasi, il modello permette di rilevare i difetti presenti nel progetto prima che possano svilupparsi ulteriormente e funziona bene per progetti nei quali il controllo di qualità sia una questione di primaria importanza.

Il ciclo di vita a cascata puro consiste di più fasi non sovrapposte, come mostrato in figura 2. Il modello inizia stabilendo i requisiti di sistema e i requisiti software e prosegue con il progetto generale dell'architettura, il progetto dettagliato, la generazione del codice, il collaudo e la manutenzione. Il modello a cascata serve come riferimento per molti altri modelli del ciclo di vita.

L'elenco seguente spiega i passi necessari per utilizzare il modello a cascata:

- Requisiti di sistema – Stabilisce i componenti necessari per costruire il sistema, inclusi i requisiti hardware, gli strumenti software e gli altri componenti necessari. Esempi includono la scelta dell'hardware, come le schede plug-in (numero di canali, velocità di acquisizione e così via) e componenti software esterni come database o librerie.
- Requisiti software – Stabilisce le aspettative sulle funzionalità del software e identifica quali requisiti di sistema dipendono dal software. L'analisi dei requisiti include la determinazione delle interazioni necessarie con altre applicazioni e database, i requisiti di prestazioni, i requisiti di interfaccia utente e così via.
- Progettazione generale dell'architettura - Determina l'infrastruttura software del sistema necessaria per soddisfare i requisiti specificati. Il progetto definisce i principali componenti e l'interazione di tali componenti, ma non definisce la struttura di ciascun componente. Determinerete inoltre le interfacce e gli strumenti esterni da usare nel progetto.
- Progettazione dettagliata - Esamina i componenti software definiti nella fase di progettazione generale e produce una specifica su come implementare ogni componente.
- Generazione del codice - Implementa le specifiche dettagliate del progetto.
- Collaudo – Determina se il software risponde ai requisiti specificati e trova tutti gli errori presenti nel codice.
- Manutenzione – Risolve i problemi e le richieste di miglioramento dopo il rilascio del software.

In alcune organizzazioni, un comitato di controllo dei cambiamenti mantiene la qualità del prodotto rivedendo ciascun cambiamento eseguito nella fase di manutenzione.

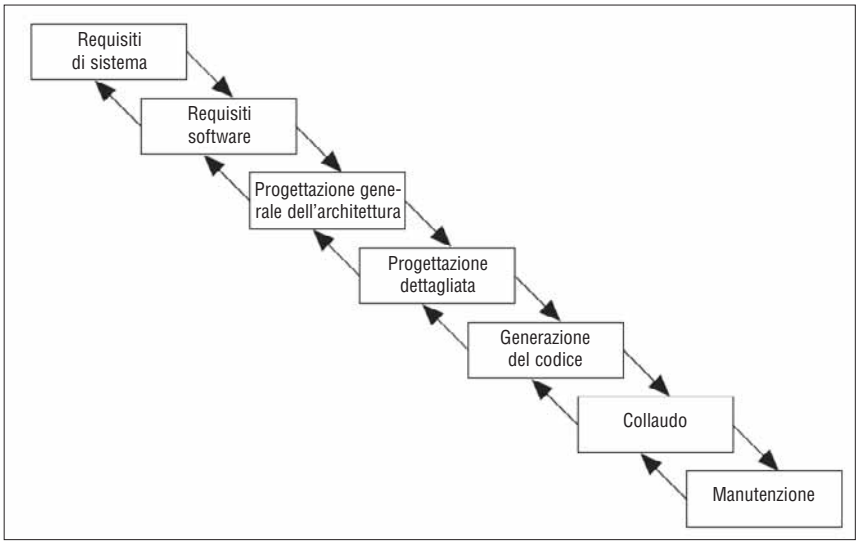


Figure 2 – Le fasi del modello a cascata

Considerate l'applicazione dell'intero modello a cascata del ciclo di sviluppo quando sia necessario apportare correzioni o implementare richieste di miglioramento.

In ogni fase, create documenti che spiegano gli obiettivi e descrivono i requisiti per quella fase. Alla fine di ogni fase, eseguite una revisione per determinare se il progetto può procedere nella fase successiva.

Dovete anche incorporare la prototipazione in ogni fase a partire dalla progettazione generale dell'architettura.

Molti ritengono che non sia possibile applicare questo modello a tutte le situazioni. Per esempio, con il modello a cascata puro, dovete definire i requisiti prima di iniziare il progetto e dovete definire il progetto completo prima di iniziare la codifica. Non essendoci sovrapposizione tra le fasi, nello sviluppo reale potreste scoprire problemi durante le fasi di progettazione o codifica che evidenziano errori o lacune presenti nei requisiti.

Il metodo a cascata non proibisce il ritorno ad una fase precedente: per esempio, dalla fase di progettazione alla fase dei requisiti. Tuttavia, ciò implica costose fasi di revisione e lo sviluppo di un'ampia documentazione.
Pertanto, correggere successivamente le sviste compiute nella fase dei requisiti risulta dispendioso.
Poiché l'effettivo sviluppo avviene tardi nel corso del processo, normalmente non si vedono risultati per molto tempo. Questo ritardo può lasciare perplessi sia i dirigenti che i clienti finali. Molte persone ritengono inoltre che la quantità di documentazione richiesta sia eccessiva e poco flessibile.
Benché il modello a cascata abbia i suoi punti deboli, risulta comunque istruttivo, perché enfatizza fasi importanti dello sviluppo del progetto. Anche se non applicate questo modello, considerate ciascuna di tali fasi e la sua relazione con il vostro progetto.

## IL MODELLO A CASCATA MODIFICATO

Molti ingegneri raccomandano versioni modificate del ciclo di vita a cascata.
Tali modifiche tendono a focalizzarsi sulla possibililtà di sovrapporre alcune fasi, riducendo il volume di documentazione necessario e il costo del ritorno a fasi precedenti per la loro revisione. Un'altra modifica comune è quella di incorporare la prototipazione nelle fasi dei requisiti.
La sovrapposizione delle fasi, come la fase dei requisiti e la fase di progettazione, rende possibile integrare nei requisiti un feedback dalla fase di progettazione. Tuttavia, la sovrapposizione delle fasi può rendere difficile sapere quando si è finito di lavorare con una di esse. Di conseguenza, è più difficile tracciare il progresso nello sviluppo del progetto. Senza fasi distinte, i problemi possono costringervi a deferire decisioni importanti, quando risulta più costoso apportare correzioni.

## PROTOTIPAZIONE

Uno dei problemi principali con il modello a cascata è che i requisiti spesso non vengono completamente compresi nelle prime fasi di sviluppo. Quando raggiungete le fasi di progettazione o codifica, conviene cominciare a vedere come ogni parte funziona insieme alle altre e scoprire eventualmente se occorre apportare qualche modifica ai requisiti.
La prototipazione è uno strumento efficace per dimostrare come un progetto soddisfi un insieme di requisiti. Potete costruire un prototipo, modificare i requisiti e rivedere il prototipo diverse volte finché non avete un quadro chiaro degli obiettivi globali. Oltre a chiarire i requisiti, un prototipo definisce anche molte aree del progetto simultaneamente. Il modello a cascata puro permette la prototipazione nella fase avanzata di progettazione generale dell'architettura e nelle fasi successive, ma non nelle prime fasi dello sviluppo.
Tuttavia, anche la prototipazione ha i suoi svantaggi: la sensazione di avere un sistema funzionante può indurre i clienti ad attendersi un sistema completo prima di quanto sia possibile. Nella maggior parte dei casi, i prototipi si basano su compromessi che permettono di procedere più rapidamente, ma che impediscono al prototipo di rappresentare una base efficace per futuri sviluppi. Dovete decidere presto se volete utilizzare il prototipo come base per futuri sviluppi e tutte le parti devono concordare con questa decisione prima che inizi lo sviluppo.
Fate attenzione che la prototipazione non diventi una maschera per un ciclo di sviluppo "code and fix". Prima di iniziare la prototipazione, raccogliete chiari requisiti e create un piano di progettazione. Limitate la quantità di tempo che dedicherete alla prototipazione prima di iniziare. I limiti di tempo aiutano ad evitare un eccesso di lavoro nella fase di prototipazione. Mano a mano che incorporate cambiamenti, aggiornate i requisiti e il progetto corrente. Quando avete finito la prototipazione, considerate la possibilità di tornare a uno degli altri modelli di sviluppo.

## METODI DI PROTOTIPAZIONE IN LABVIEW

Vi sono diversi modi per prototipare un sistema in LabVIEW. Nei sistemi che presentano requisiti di I/O che sono difficili da soddisfare, potete sviluppare un prototipo per testare i cicli di controllo e acquisizione e le loro velocità. Nei prototipi di I/O, dati casuali possono simulare i dati acquisiti nel sistema reale.
I sistemi aventi molti requisiti di interfaccia utente sono perfetti per la prototipazione. E' difficile determinare sulla carta il metodo da usare per visualizzare i dati o chie-

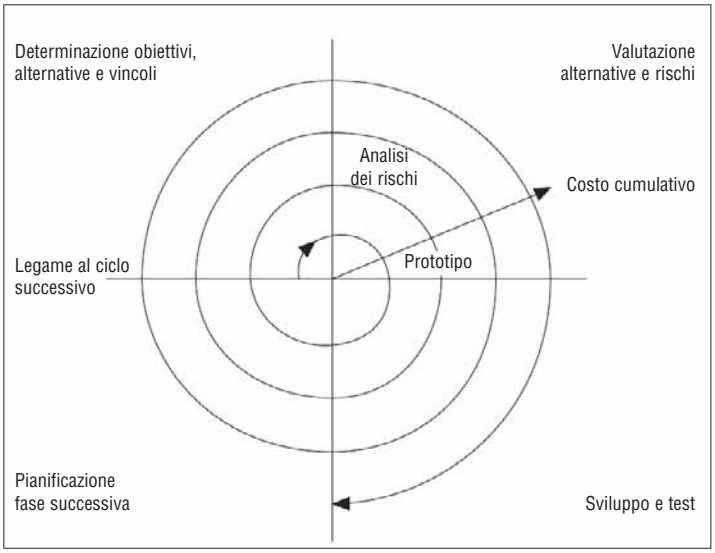


Figure 3 – Modello del ciclo di vita a spirale

dere impostazioni all'utente. Al contrario, considerate la progettazione di pannelli frontali dei VI con i controlli e gli indicatori che vi occorrono. Lasciate il diagramma a blocchi vuoto e immaginate il funzionamento dei controlli e se le varie azioni implementate richiedano eventualmente altri pannelli frontali.
Se siete impegnati su un progetto per un cliente, usare prototipi dei pannelli frontali è un modo estremamente efficace per discutere con il cliente come potete soddisfare i suoi requisiti.

## IL MODELLO A SPIRALE

Il modello a spirale è una popolare alternativa al modello a cascata. Esso enfatizza la gestione dei rischi, permettendovi di scoprire i maggiori problemi nelle prime fasi del ciclo di sviluppo. Nel modello a cascata, dovete completare il progetto prima di iniziare la codifica.

| | Rischio | Probabilità | Perdita | Esposizione al rischio | Gestione del rischio |
|---|---|---|---|---|---|
| 1 | I rate di acquisizione sono troppo elevati | 5 | 9 | 45 | Sviluppare un prototipo per dimostrare la fattibilità |
| 2 | Il formato dei file potrebbe non essere efficiente | 5 | 3 | 15 | Sviluppare dei benchmark per valutare la velocità di manipolazione dei dati |
| 3 | Interfaccia utente non ben definita | 2 | 5 | 10 | Coinvolgere il cliente; sviluppare un prototipo |

Tabella 1 – Sintesi delle compatibilità per le shared variable pubblicate in rete

Con il modello a spirale, il progetto viene suddiviso in un insieme di rischi che è necessario affrontare. Iniziate con una serie di iterazioni nelle quali analizzate il rischio più importante, valutate le opzioni per risolvere il rischio, gestite il rischio, valutate i risultati e pianificate l'iterazione successiva. La figura 3 illustra il modello del ciclo di vita a spirale.
I rischi sono tutti gli aspetti che non sono chiaramente definiti o che hanno la possibilità di influire negativamente sul progetto. Per ogni rischio, considerate le due cose seguenti:

- La probabilità che il rischio si verifichi
- La severità dell'effetto del rischio sul progetto (perdita)

Potete usare una scala da 1 a 10 per ciascuno di questi elementi, dove 1 rappresenta la minima probabilità o perdita e 10 rappresenta il massimo. L'esposizione al rischio è il prodotto di questi due valori.
Usate qualcosa di simile alla tab. 1 per tenere traccia degli elementi di massimo rischio del progetto.
In generale, affrontate prima i rischi che hanno la più alta esposizione al rischio. In questo esempio, la prima spirale affronta la possibilità che le velocità di acquisizione dati siano troppo elevate. Se dopo la prima spirale dimostrate che le velocità sono elevate, potete passare a una diversa configurazione hardware per soddisfare i requisiti di acquisizione. Ogni iterazione può identificare nuovi rischi. In questo esempio, usare hardware più potente può introdurre il nuovo rischio di costi più elevati.
Supponiamo che stiate progettando un sistema di acquisizione dati con una scheda di acquisizione plug-in. In questo caso, il rischio è se il sistema può acquisire, analizzare e visualizzare i dati abbastanza rapidamente.
Alcuni dei vincoli, in questo caso, sono il costo del sistema e i requisiti di una specifica velocità di campionamento e precisione.
Una volta determinate le opzioni e i vincoli, potete valutare i rischi. In questo esempio, create un prototipo o un benchmark per testare le velocità di acquisizione. Dopo aver visto i risultati, potete valutare se continuare con l'approccio o scegliere un'opzione differente. Lo farete riesaminando i rischi in base alla nuova conoscenza che avete acquisito costruendo il prototipo.
Nella fase finale, valuterete i risultati con il cliente. In base all'input del cliente, potete riesaminare la situazione, decidere il prossimo rischio più elevato e ricominciare il ciclo. Il processo continua finché il software è finito o quando decidete che i rischi sono troppo elevati e terminate lo sviluppo. E' possibile che nessuna di queste opzioni sia percorribile, perché le opzioni sono troppo costose, laboriose o non soddisfano i requisiti.
Il vantaggio del modello a spirale, rispetto al modello a cascata, è che potete valutare quale rischio affrontare ad ogni ciclo. Potendo valutare prima i rischi grazie ai prototipi, potete affrontare i maggiori ostacoli e selezionare delle alternative già nelle prime fasi, cosa che si rivela meno costosa. Con un modello a cascata standard, invece, le ipotesi relative ai componenti rischiosi possono essere distribuite lungo l'intero progetto e quando scoprite i problemi, il lavoro aggiuntivo necessario può rivelarsi molto costoso.

**Note sull'autore**
Laureato in ingegneria nucleare al Politecnico di Milano, Matteo Foini lavora in qualità di Technical Marketing Engineer presso National Instruments Italy.