Azzolini Riccardo 2019-09-24

Modello di sviluppo a cascata

1 Dati aneddotici sugli errori

- Con le tecniche di ispezione sistematica (che consistono nella lettura del codice, senza eseguirlo) si scoprono il 50–75 % degli errori. Tali tecniche hanno comunque dei vantaggi:
  - permettono di individuare alcuni tipi di errori che non verrebbero rilevati con il testing;
  - rivelano immediatamente la causa degli errori, mentre il testing ne segnala solo l'esistenza, e quindi deve comunque essere seguito da un'ispezione (per il debugging);
  - hanno un costo inferiore al testing;
  - si possono applicare anche a semilavorati non eseguibili.
- I moduli (es. metodi) con flusso di controllo più complesso contengono più probabilmente errori, ed è più difficile individuarli tutti.
- Spesso, i test coprono solo il 50 % (circa) del codice, cioè fanno eseguire solo la metà delle istruzioni. L'ideale sarebbe una copertura del 100 %, ovvero che tutte le istruzioni vengano eseguite almeno una volta, ma ciò non garantisce comunque la correttezza (perché le stesse istruzioni potrebbero provocare degli errori in altre circostanze). In generale, però, è raro raggiungere una copertura superiore al 70 % (nonostante l'obiettivo fissato nei progetti sia solitamente l'85 %), perché, anche con l'aggiunta di più dati di test, si tendono a "ripercorrere" soprattutto istruzioni già testate.
- Il codice consegnato contiene il 10 % degli errori trovati nel testing (che sono comunque tanti).
- Gli errori introdotti presto sono scoperti tardi, e il costo di rimozione aumenta con il tempo: solitamente, la rimozione di errori da sistemi grandi e vecchi costa da 4 a 10 volte tanto quanto la rimozione da sistemi piccoli e nuovi. Un modo per cercare di prevenire problemi del genere è effettuare delle ispezioni sistematiche già in fase di design.
- La rimozione di errori causa l'introduzione di nuovi errori. Per questo, i sistemi di grandi dimensioni tendono a stabilizzarsi a una certa densità di difetti.

2 Evoluzione

Le cause dell'evoluzione sono:

- difetti;
- cambiamenti di contesto (manutenzione adattativa);
- cambiamenti dei requisiti (gli utenti possono fare nuove richieste, anche a causa dell'introduzione del sistema);
- specifiche sbagliate (perché i requisiti non sono stati determinati correttamente, oppure perché il dominio non è ben conosciuto);
- requisiti non conosciuti dall'inizio, e quindi fissati o cambiati in corso d'opera.

Per affrontare l'evoluzione, bisogna:

- anticipare i cambiamenti prevedibili;
- progettare il software in modo che sia facilmente modificabile (questo è uno dei principali obiettivi del software engineering).

3 Gestione dei cambiamenti

Siccome il software è molto facile da modificare, in situazioni di emergenza i cambiamenti vengono spesso applicati direttamente al codice, senza aggiornare i documenti, che allora non riflettono più lo stato del progetto.

Invece, è buona pratica cambiare prima i documenti (eventualmente fino ai requisiti), e solo in seguito modificare il codice.

Infatti:

- la manutenzione del software non è quasi mai prevista;
- piccole modifiche possono avere grandi effetti, anche imprevisti (soprattutto se il software non è stato progettato in modo adeguato).

Quindi, quando non viene gestita correttamente, la manutenzione causa dei disastri.

4 Varianti del ciclo a cascata

Il ciclo a cascata è un modello generico. Per questo, un'organizzazione che lo utilizza ne definisce solitamente una propria variante, adattata alla sua specifica realtà.

5 Svantaggi del modello a cascata

Il modello a cascata si basa sull'assunzione che il dominio e i requisiti siano ben conosciuti e stabili, ma ciò è vero solo in pochi casi. In altre situazioni, alcune delle sue prescrizioni non sono realistiche: ad esempio, non si possono eliminare i cicli.

Inoltre, dal punto di vista dell'utente/committente, questo modello è un "black box": una volta stabilite le specifiche, lo sviluppo del progetto avviene in modo completamente interno all'azienda sviluppatrice, e l'utente non può fornire feedback fino alla consegna del prodotto finito. Una possibile soluzione è rendere più visibile il processo di sviluppo, stabilendo ad esempio delle milestone alla fine di ogni fase: in questo modo, l'utente può valutare il progresso e fornire eventuale feedback. Lo sviluppo diventa così più efficace, perché è più difficile divergere dalle esigenze dell'utente.

6 Verifica e convalida

Con la **verifica** si controlla se si sta sviluppando il prodotto nel *modo giusto*. Essa avviene interamente, tra una fase e l'altra (ad esempio, si controlla che il design rispetti le specifiche, poi che il codice rispetti il design, ecc.).

La **convalida**, invece, esamina se si sta sviluppando il *prodotto giusto*, richiedendo a intervalli regolari il feedback dell'utente.