Azzolini Riccardo 2019-09-23

Modello di sviluppo a cascata

1 Code & fix

La prima metodologia usata per lo sviluppo del software è stata il "code & fix": si scriveva subito il codice, e si risolvevano i problemi man mano che li si incontrava.

Ciò ha portato, negli anni '60, alla *crisi del software*: mentre le prestazioni dell'hardware aumentavano in maniera esponenziale, la produttività degli sviluppatori era al massimo stagnante, se non in declino. Per risolvere tale crisi, è nata la disciplina del *software engeneering*, che ha introdotto dei modelli formali del processo di sviluppo.

2 Modello a cascata

Il **modello a cascata** (**waterfall**) paragona lo sviluppo del software alla produzione di un oggetto fisico (catena di montaggio):

- è composto da una serie di *fasi*, chiamate anche *attività*;
- segue una direzione lineare prevalente da una fase all'altra, senza cicli, che sono considerati dannosi, dato che evitarli permette una pianificazione e un controllo migliori;
- standardizza l'output di ciascuna fase, in modo che esso possa essere facilmente usato nelle fasi successive, anche nei casi in cui queste vengono svolte da team diversi.

Le fasi del modello a cascata sono:

1. **studio di fattibilità** (*feasibility study*);
2. **analisi e specifica dei requisiti** (*requirements analysis and specification*);
3. **design**;
4. **codifica e test di unità** (*coding and unit test*);
5. **test di integrazione e di sistema** (*integration and system test*);
6. **messa in esercizio** (*deployment*);
7. **manutenzione** (*maintenance*).

Le prime tre fasi, chiamate *fasi alte* (*early phases*), sono le più importanti, perché le decisioni effettuate nel corso di queste si riflettono sulle fasi successive. Le fasi restanti, invece, sono chiamate *fasi basse* (*late phases*).

3 Studio di fattibilità

- Si effettua un'analisi dei costi/benefici, sia sul breve periodo che a medio-lungo termine.
- Si valuta se conviene sviluppare internamente il progetto o acquistare un prodotto esistente.
- Si considerano le possibili alternative.
- Si determinano le risorse necessarie per il progetto.

Prodotto: **Documento di studio di fattibilità** (*Feasibility Study Document*), che contiene:

- una descrizione preliminare del problema (che verrà poi integrata con le informazioni acquisite nelle fasi successive, in particolare la specifica dei requisiti);
- la descrizione delle varie soluzioni possibili;
- stime preliminari dei costi e tempi delle diverse alternative (che, in questa fase, vengono solo elencate, senza effettuare già una scelta).

In pratica, per via delle scadenze (*time pressure*) e dei costi (*cost pressure*) dello studio di fattibilità, esso non può essere effettuato in maniera molto approfondita, soprattutto considerando che, in questa fase, non si è ancora sicuri che il cliente accetterà l'offerta. Se, però, lo studio è troppo superficiale, può essere che:

- non vengano esplorate tutte le alternative;
- non vengano determinati correttamente i rischi.

È quindi necessario trovare un equilibrio tra tempi/costi e livello di approfondimento.

4 Analisi e specifica dei requisiti

Comprende varie attività:

- Analisi del dominio in cui verrà utilizzata l'applicazione.

- Identificazione dei requisiti, cioè di cosa deve fare il software. A tale scopo, è necessario interagire con gli utenti finali (che, in genere, vorrebbero più funzionalità) e con il committente (che, solitamente, vorrebbe costi e tempi di realizzazione minori), raggiungendo poi un equilibrio tra le loro richieste mediante un processo di negoziazione e contrattazione.

Prodotto: **Documento di analisi e specifica dei requisiti** (*Requirements Analysis and Specification Document*, RASD), che:

- serve per la fase di design;
- *ha valore contrattuale* (è l'"allegato tecnico").

Esso deve contenere le risposte a 5 domande:

Who: chi (quali figure) userà il sistema;

Why: perché sviluppare il sistema, e perché gli utenti lo useranno (eventualmente, rispetto ad applicazioni fornite dalla concorrenza);

What: cosa farà il sistema (e non *How*, cioè come lo farà);

Where: dove sarà usato il listema, con quale architettura;

When: quando e per quanto tempo verrà usato il sistema.

I requisiti specificati nel documento si suddividono in:

- requisiti funzionali;
- requisiti non funzionali;
- requisiti sul processo di sviluppo e manutenzione (ad esempio, il committente può voler seguire i progressi dello sviluppo, imporre scadenze per determinate fasi, ecc.).

Tale documento deve inoltre essere:

preciso: deve dire esattamente cosa farà il software, senza lasciare ambiguità;

completo: deve comprendere tutte le funzionalità da implementare;[1]

congruente: non contraddittorio;

comprensibile: non deve contenere linguaggio e/o diagrammi tecnici, dettagli sulla realizzazione, ecc., in modo che possa essere compreso dall'utente finale;

modificabile: il software evolve, e le modifiche partono dall'aggiornamento dei requisiti.

[1] In questo modo, a sviluppo completo, il committente potrà verificare che siano state implementate tutte le funzionalità richieste, e al tempo stesso non potrà richiedere funzionalità aggiuntive senza modificare il contratto.

Infine, possono essere inclusi (ed è fortemente consigliato che lo siano) anche:

- un manuale utente preliminare;
- un piano di test di sistema, cioè un insieme di casi di test (input e output) da usare per determinare il funzionamento del software completo, ma che, in questa fase, aiutano anche a definire con precisione i requisiti.

5 Design

In questa fase si definisce l'architettura del software:

- **componenti**, detti anche **moduli** (ad esempio, possono essere le classi di un linguaggio ad oggetti);
- **relazioni** (*statiche*, come ad esempio l'ereditarietà) tra componenti;
- **interazioni** (*dinamiche*, come ad esempio le chiamate di metodi) tra componenti;

Il design deve essere svolto in modo da supportare lo sviluppo concorrente di componenti diversi e la separazione delle responsabilità (così, nelle fasi successive, ognuno "saprà cosa deve fare").

Prodotto: **Documento di design** (*Design Document*). A differenza dei documenti precedenti, questo non deve essere leggibile dall'utente, quindi contiene linguaggio tecnico, appositi diagrammi, ecc., che permettono di descrivere con precisione l'architettura del sistema.

6 Codifica e test di unità

Ciascun modulo viene implementato con il linguaggio di programmazione scelto,[2] e poi testato dallo sviluppatore stesso (per questa fase, non serve un team di testing dedicato).

I test compresi in questa fase sono detti *test di unità* perché ciascun modulo (unità) viene testato indipendentemente dagli altri, "come se il resto del mondo non esistesse".

Spesso, la codifica e il testing di un modulo sono effettuate contemporaneamente, sfruttando i risultati dei test per assistere lo sviluppo.

I programmi sviluppati in questa fase comprendono anche la loro documentazione.

[2] La scelta definitiva del linguaggio di programmazione viene effettuata solo in questa fase (anche se, solitamente, si ha un'idea già dall'inizio), per evitare di "legarsi le mani" finché non è assolutamente necessario. In generale, in un progetto, conviene aspettare a prendere decisioni finché non si hanno informazioni sufficienti per fare una scelta informata.

7 Test di integrazione e di sistema

I moduli vengono integrati in (sotto)sistemi, e questi ultimi vengono testati. Solitamente, si prevede di avere dei malfunzionamenti, ma non si sa dove essi si verificheranno. Per questo, conviene procedere in modo incrementale, testando prima dei sottosistemi semplici, e procedendo via via a quelli più completi/complessi (ad esempio, aggiungendo una classe alla volta: in questo modo, ogni difetto individuato può essere causato solo da interazioni che coinvolgono la classe appena aggiunta).

In teoria, il test di integrazione assume che le singole unità siano corrette, e che quindi vadano verificate solo le interazioni. In pratica, invece, si possono ancora avere errori nelle singole unità.

Dopo il test di integrazione, si effettua un test completo del sistema, svolto internamente all'azienda sviluppatrice per verificare il funzionamento del prodotto, e infine un *test di accettazione*. Quest'ultimo coinvolge l'utente, permettendogli di verificare che il funzionamento del software rispetti il contratto.

7.1 Distribuzione dell'effort

Come esempio, la distribuzione dell'effort (sforzo) calcolata su 125 progetti dell'azienda HP è:

- 18 % specifica;
- 19 % design;
- 34 % codifica e test di unità, di cui
  - 10 % codifica,
  - 24 % test di unità;
- 29 % test di integrazione e di sistema.

Le distribuzioni di altri progetti presentano alcune variazioni, ma in genere sono piuttosto simili a questa.

Osservazioni:

- La codifica vera e propria costituisce solo il 10 % dell'effort.
- Le varie fasi di testing richiedono, complessivamente, oltre il 50 % dell'effort del progetto. Per ridurre tale percentuale, sono state sviluppate delle metodologie che permettono di rendere più efficaci, "potenziare", le fasi precedenti.

8 Messa in esercizio

L'obiettivo di questa fase è distribuire e installare l'applicazione sui sistemi degli utenti. A volte ciò è banale, ma non sempre (ad esempio, se ci sono parametri particolari da configurare, basi di dati da predisporre, ecc.)

9 Manutenzione

La manutenzione comprende tutti i cambiamenti effettuati dopo la consegna dell'applicazione.

Spesso questa fase costa, da sola, più di tutto il resto del progetto (cioè costituisce oltre il 50 % del costo totale). Infatti, nonostante sia descritta dal modello a cascata come una fase sola, la manutenzione richiede in realtà la ripetizione dell'intero procedimento.

La ripartizione tra i tre tipi di manutenzione è approssimativamente:

- correttiva: 20 %
- adattativa: 20 %
- perfettiva: 50 %

(il totale non è 100 % perché questi dati sono un'approssimazione).

10 Altre attività

Oltre alle 7 fasi sequenziali del modello a cascata, ci sono alcune attività che vengono effettuate per tutta la durata del ciclo di sviluppo:

documentazione: essa deve essere aggiornata in tutte le fasi di sviluppo, e anche in caso di cambiamenti apportati durante la manutenzione;

verifica: controllare che ciascuna fase rispetti le precedenti (ad esempio, che la fase di design rispetti le specifiche, che il codice rispetti il documento di design, ecc.);

management: attività di gestione che permette l'avanzamento del progetto.