1 Lezione del 24-10-25

1.1 Strategie di test

Come la progettazione, anche il testing può essere effettuato in modalità **top-down** o **bottom-up**.

- **Top-down**: si parte dalla *"testa"* del progetto e si va scendere. L'idea è di andare a simulare dalla radice fino agli altri moduli del sistema, prima che questi moduli vengano implementati. Il processo si ripete per via ricorsiva finché l'intero sistema non risulta come simulato.

  In questo caso, per testare i moduli più alti bisogna disporre dei moduli di prova, detti *simulatori* o **stub**, che vanno appunto a simulare il comportamento dei moduli più bassi.

- **Bottom-up**: si parte dai dettagli e si va a scendere. L'idea è che si vanno a collaudare prima i moduli di più basso livello nella gerarchia prodotta dalla progettazione: quando si valutano come corretti, si passa al livello superiore, e via dicendo fino a che non si è simulato il sistema intero.

  Chiaramente, testare gli stadi più bassi richiede lo sviluppo di software detti **driver** (da non confondere con i driver dei S/O). Un driver è un software che ha il compito di *pilotare* una funzionalità di livello più basso, simulando appunto come questa funzionalità potrebbe venire chiamata da eventuali stadi superiori.

  Il vantaggio sta nel fatto che si testano prima gli stadi più bassi, trovando gli errori *subito* (e non più tardi nello sviluppo, che sarebbe molto più costoso).

La strategia che si implementa più spesso nella realtà è in verità una strategia **mista**. Quando le funzionalità di basso livello risultano più complicate, è infatti utile sfruttare un approccio bottom-up, mentre se il sistema è governato da una complessa logica di alto livello, è più comodo il top-down.

1.1.1 Verifiche statiche

Il punto critico dei test è rappresentato dal costo. Una forma di testing alternativa è quella di **verifica** (o *ispezione*) **statica** del codice. Questo solitamente ci permette di individuare fra il 60% e il 90% degli errori.

1.1.2 Testing

L'ispezione statica chiaramente non basta, in quanto non intercetta gli errori dovuti all'esecuzione dinamica del programma. Occorre quindi procedere con un'apposita fase di **testing** dove si valuta il programma su più campioni di dati, in modo da verificarne il comportamento come corretto.

Dobbiamo ricordare che è il testing é:

- Mai **esaustivo**, e.g. un milione di test non equivarranno mai una verifica formale;
- **Costoso**, cioè richiede sfruttamente di risorse macchina, e sopratutto risorse umane (tempo speso a scrivere test, eseguire test, ecc...).

Per effettuare un test bisogna:

- Costruire un **test-case**, cioè disporre una situazione dove si conosce l'**input** e l'**output** corretto; I test-case vanno ricavati alle specifiche e dai documenti ricavati dalla progettazione;
- Dopo che il test-case è pronto, e si è configurato il sistema perché esegua il test, bisogna appunto eseguirlo, ottenere l'output e confrontarlo con quello desiderato;
- I test devono essere **ripetibili**: bisogna poter ottenere più test (magari variando parametri) sullo stesso test-case, in modo da essere il più esaustivi possibile.

Una prima idea per generare gli input è quello di generarli **casualmente**. Questa soluzione è economica, ma ha lo svantaggio di non essere molto uniforme per piccoli campioni. Inoltre, gli input non saranno particolarmente significativi all'applicazione vera e propria (quindi creando situazioni irreali e di difficile valutazione).

In questo risulta quindi più utile unire le specifiche del programma, la struttura stessa del programma, e sopratutto l'**esperienza** sul dominio del problema che il programma risolve. Questa spesso non viene dal programmatore, ma dagli attori che sono all'interno del dominio di problema stesso (i clienti).

1.1.3 Procedure di testing

Ci sono almeno 2 modi principali su come procedere al testing:

- Testing a **scatola nera** (o **black box** testing): in questo caso si ricavano gli input solo guardando alle specifiche del programa. Il codice scritto a questo punto non viene considerato. Questo tipo di testing è tipico delle prime fasi di testing.

  Dal punto di vista teorico, l'idea del testing a scatola nera è quello di dividere dividere i possibili input in **classi di equivalenza**, in modo che ogni possibile input abbia lo stesso **potere di testing**.

- Testing **strutturale**: unisce le specifiche di programma al codice finora scritto per la generazione di test. Ha il vantaggio di testare più precisamente alcune procedure e dettagli che test a scatola nera non individuerebbero nel codice.

  Dal punto di vista teorico, quello che vorremmo fare è sviluppare test che *eslporino* tutto lo spazio di stato del programma, cioè attivino tutte le procedure (pensa flussi alternativi), in modo da testare funzionalità a cui magari si accede più raramente.

1.2 Testing automatizzato

Chiaramente, è molto utile realizzare **test automatici**, cioè sfruttare strumenti software per effettuare automaticante i test al posto nostro.

I casi tipici di automazione dei test sono:

- Test **funzionali** per un prodotto, pensati per verificare le specifiche desiderate del prodotto;
- Test per **nuovi rilasci** di un prodotto, atti a verificare che la funzionalità precedente non sia stata compromessa.

I **simulatori** sono ambienti all'interno di cui si può effettuare il testing non solo delle procedure del software, ma della totalità delle condizioni operative che dovrebbero trovarsi all'esterno del software.

Questo chiaramente risulta molto costoso, ma può essere utile (se non fondamentale) per programmi particolarmente complessi, o per domini di problema particolarmente critici (mezzi di trasporto, industria biomedica, ecc...).