Principi dell'ingegneria del software

Leggere Cap. 3 Ghezzi et al.

università di ferrara
DA SEICENTO ANNI GUARDIAMO AVANTI.

Sommario

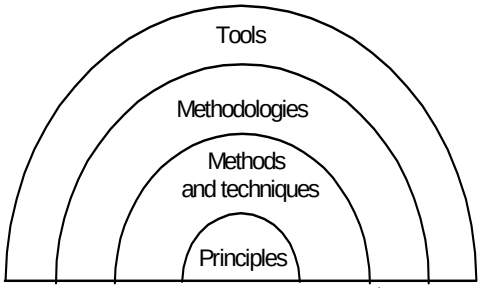
- Principi dell'ingegneria del software
- Relazioni fra
  - Principi
  - Metodi e tecniche
  - Metodologie
  - Strumenti
- Descrizione dei principi
- Caso di studio: compilatore

Principi dell'ingegneria del software

- Principi fondamentali che si applicano sia al processo che al prodotto
- Si applicano attraverso *metodi* e *tecniche*
- Metodi e tecniche si possono comporre in *metodologie*
- L'applicazione delle metodologie può avvalersi di (o essere imposta da) *strumenti*

Principi, metodi e tecniche, metodologie, strumenti

- Ogni strato si basa su quello immediatamente più interno
- La variabilità nel tempo cresce verso l'esterno



I principi

- Rigore e formalità
- Separazione degli interessi
- Modularità
- Astrazione
- Anticipazione del cambiamento
- Generalità
- Incrementalità

Rigore e formalità

- L'ingegneria del software è un'attività creativa
- Deve però essere praticata in modo sistematico
- Il rigore aumenta la nostra fiducia nei prodotti
- La formalità (definizione matematica dei concetti usati) è il rigore al massimo grado

Rigore e formalità

- L'ingegnere deve scegliere il grado di rigore opportuno (il rigore costa)
- In generale, a una maggior necessità di fiducia (componenti critici) deve corrispondere un maggior rigore
- La formalità facilita le procedure automatiche di generazione e verifica

Rigore e formalità

- Esempi nel prodotto:
  - prova formale (matematica) di correttezza di un programma
  - generazione sistematica dei dati di test
- Esempio nel processo:
  - una documentazione rigorosa (anche se informale) del processo facilita la gestione dei progetti e il controllo dei tempi di produzione

Rigore

- Il rigore influenza positivamente anche:
  - manutenibilità
  - riusabilità
  - portabilità
  - comprensibilità
  - interoperabilità

Separazione degli interessi

- Affrontare la complessità di un problema considerandone separatamente gli aspetti.
- Quasi sempre gli aspetti di un problema come la produzione di un software sono correlati e a volte in contrasto.
- Tuttavia, nella pratica, la separazione degli interessi è necessaria (anche per suddividere il lavoro fra persone)

Separazione degli interessi

- Separazione delle attività in diversi periodi temporali
- Separazione della trattazione delle diverse qualità (es.: correttezza, prestazioni, usabilità)
- Separazione del sistema in diverse parti (modularità)
- Separazione degli aspetti relativi al dominio del problema da quelli relativi all'implementazione

Separazione degli interessi

- A volte si perde la possibilità di fare ottimizzazioni globali
- Perciò occorre decidere in anticipo cosa trattare separatamente e cosa no

Modularità

- Istanza del principio della separazione degli interessi, applicato alle parti del sistema.
- Un sistema può essere suddiviso in parti dette *moduli.*
- Si opera (specifica, progettazione, implementazione, verifica):
  1. separatamente sui singoli moduli
  2. sulla loro interazione

Modularità

- Top-down (*divide et impera*)
  1. Prima si suddivide il sistema nei moduli che lo compongono
  2. Poi si opera sui singoli moduli
- Bottom-up:
  1. Prima si opera su singoli moduli
  2. Poi li si compone per formare il sistema

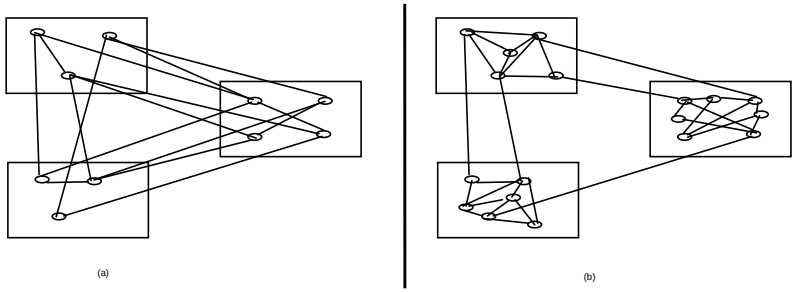
Modularità

- Benefici: possibilità di
  - scomporre un sistema in parti più semplici
  - comporre un sistema a partire da parti più semplici
  - comprendere un sistema come insieme di parti interagenti
  - modificare un sistema operando localmente sui moduli
  - favorisce la comprensibilità e la manutenibilità

Modularità

- Alta coesione dei moduli
  - gli elementi di un modulo sono raggruppati per un motivo logico
  - es.: funzioni raggruppate come metodi di una classe se condividono lo stato
- Debole accoppiamento fra moduli
  - due moduli hanno poche dipendenze reciproche
  - è più facile considerarli separatamente
  - sostituirne uno è meno costoso

Modularità



Accoppiamento forte

Accoppiamento debole

Astrazione

- Consiste nell'ignorare i dettagli non significativi e considerare gli aspetti importanti di un problema
- Necessaria per affrontare la complessità dei problemi
- Che cosa sia significativo dipende dallo scopo dell'astrazione

Astrazione

- L'astrazione produce modelli
  - La specifica (formale o informale) di un software è un modello della sua implementazione
  - Ragionando sulla specifica si possono ignorare i dettagli implementativi
  - Es.: la semantica dei linguaggi di programmazione astrae dall'hardware sottostante

Astrazione:

- I commenti nel codice sono astrazioni di ciò che fa una funzione, procedura o frammento di codice
- L'astrazione viene applicata nei processi per stimare i costi di un nuovo progetto: si utilizzano alcune caratteristiche salienti del nuovo progetto per confrontarlo con progetti già svolti

Anticipazione del cambiamento

- Nel tempo, il software può richiedere cambiamenti per adattarsi ai cambiamenti dell'ambiente e dei requisiti
- La possibilità di cambiare un software (evolvibilità) richiede che esso sia progettato in vista del cambiamento
  - Cercare di localizzare i probabili cambiamenti in singoli moduli

Anticipazione del cambiamento

- La riusabilità è fortemente influenzata dall'anticipazione del cambiamento
  - Può essere vista come evolvibilità a livello dei singoli componenti
- Necessario poter gestire le versioni del software (*configuration management*)
- Applicabile anche ai processi:
  - prevedere possibili turnover
  - prevedere la manutenzione

Generalità

- Il problema considerato potrebbe essere un'istanza di un problema più generale
- Il problema generale potrebbe essere già stato risolto, o la sua soluzione potrebbe essere riusata in seguito
- E' però necessario pesare i costi e la perdita di prestazioni portati da una maggiore generalità

Generalità

- Prodotti di uso generale (*off-the-shelf*): pacchetti generali e componenti per problemi di uso comuni. Es.
  - word processor
  - elaborazione testi
  - fogli elettronici
- Passo successivo: server applicativi (*application servers*), forniscono le funzionalità generalizzate in maniera remota

Incrementalità

- In molti casi può essere utile portare avanti un processo come sequenza di passi successivi verso il prodotto finale.
- Permette di ricevere feedback sui prodotti intermedi
- L'incrementalità è strettamente correlata all'evolvibilità

Incrementalità

- Esempi:
  - consegnare una versione del sistema con funzionalità ridotte, per ricevere un primo feedback
  - implementare prima le funzionalità, poi prendere in considerazione le prestazioni
  - produrre una serie di prototipi mentre aumenta la comprensione dei requisiti

Incrementalità

- La *prototipazione rapida* richiede un ciclo di vita differente
- Ciclo di vita flessibile e iterativo
- Richiede accurata documentazione dei passi intermedi e delle loro relazioni per evitare di perdere il controllo

Compilatore: rigore e formalità

- Un compilatore è un prodotto critico: gli errori portano alla produzione di codice non corretto. Necessità di alta fiducia sulla sua correttezza
- Le specifiche dei compilatori sono generalmente fornite in modo formale (grammatiche dei linguaggi, modelli di esecuzione)

Compilatore: rigore e formalità

- Notazione per la sintassi: forma di Backus-Naur
- Teoria degli automi e dei linguaggi formali

Compilatore: separazione degli interessi

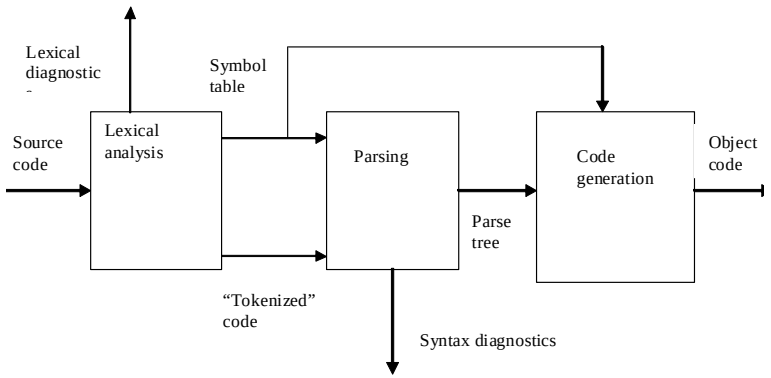
- Correttezza, efficienza (della compilazione e del codice oggetto) e usabilità (messaggi diagnostici) trattate separatamente
- Es.: non occorre pensare ai messaggi di errore mentre si implementano gli algoritmi di allocazione dei registri
- Aspetti correlati e contrastanti: diagnostica a run-time (es.: controllo degli indici di array) e prestazioni.

Compilatore: modularità

- La compilazione si può suddividere in alcune fasi:
  - Analisi lessicale
  - Analisi sintattica (*parsing*)
  - Generazione del codice
- La suddivisione in fasi si può riflettere nella suddivisione del sistema in moduli

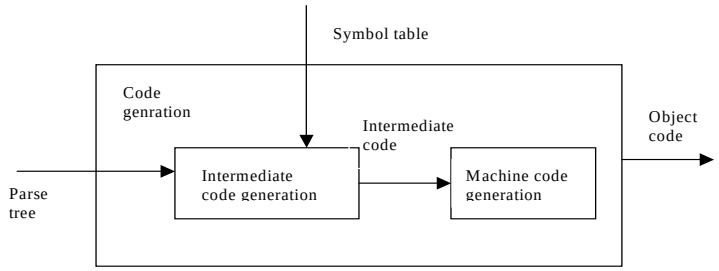
Compilatore: modularità

diagramma a scatole e linee (*box-and-line diagram*)



Compilatore: modularità

- Ulteriore suddivisione in moduli



Compilatore: astrazione

- Applicata in molti casi. Ad esempio:
  - utilizzo di una sintassi astratta anziché concreta (istruzione condizionale anziché if-then-else, che assume forma diversa in Pascal, C, ….)
  - generazione di codice intermedio da eseguire su macchina virtuale (es. bytecode Java)

Compilatore: anticipazione del cambiamento

- Sono possibili
  - cambiamenti della sintassi o semantica del linguaggio di programmazione
  - introduzione di istruzioni macchina più potenti
  - introduzione di nuovi dispositivi di input/output che richiedono istruzioni specifiche
    - le istruzioni di I/O in C e Ada sono parte di una libreria, invece che del linguaggio come in Pascal, per una migliore flessibilità

Compilatore: generalità

- Uso di codice intermedio per
  - generare codice per più macchine fisiche
  - generare codice per più linguaggi
- Compilatore parametrico rispetto al linguaggio sorgente (*compiler compiler*). Es.
  - lex e yacc in Unix
  - Definite Claue Grammars in Prolog

Compilatore: incrementalità

- Sviluppo di un compilatore per un sottoinsieme del linguaggio, poi aggiunta progressiva di tutte le caratteristiche
- Concentrarsi successivamente su
  - correttezza
  - diagnostica
  - ottimizzazione
- Fornire via via un insieme sempre più ampio di librerie