

Ingegneria del Software 2

Materiale, link utili, avvisi

<http://www.dmi.unict.it/~tramonta/se2>

Forum su uniweb 2

leggere gli avvisi
fare domande
partecipare alle discussioni

Lezioni

- Due per settimana
 - lunedì 15-17
aula 4
 - mercoledì 15-17
aula 4
- Sono obbligatorie
- Coprono tutto il corso ...

Syllabus (provvisorio)

- Tecniche (allo stato dell'arte) per
 - Progettazione (Design pattern, Aspect- e Object- Oriented)
 - Refactoring
 - Software Visualization
 - Testing
 - Metriche
- Prerequisito: Ingegneria del Software

Materiale

Libri consigliati

- Fowler. Refactoring: Improving the Design of Existing Code. Pearson Addison-Wesley
- Laddad. *AspectJ in Action*. Manning

Web

verrà suggerito del materiale disponibile su web

Articoli

verranno suggeriti o forniti articoli scientifici

Progetti

Progetti

- Due modalità
 - Durante il corso
 - Sarà assegnato più avanti (tra qualche settimana)
 - Si concluderà a fine corso, non oltre
 - Tutte le deadline dovranno essere rispettate
 - A fine corso
 - Sarà assegnato dopo la fine delle lezioni, su richiesta ed in date da concordare
 - Dovrà essere concluso dopo 4 settimane dall'assegnazione, non oltre
 - Tutte le deadline dovranno essere rispettate

Progetti

- Verrà seguito il processo di sviluppo XP
- Fase 1: requisiti
 - Deliverable: story card e codice di test (commentato)
- Fase 2: progettazione ed implementazione (release 1)
 - Deliverable: CRC, codice applicazione (commentato), codice di test
- Fase 3: progettazione ed implementazione (release 2)
 - Deliverable: story card, CRC, codice applicazione (commentato), codice di test
- Fase 4: come la fase 3, ma per una release successiva

Esami

- Valutazione
 - 50% progetto
 - 50% orale
- I deliverable che mi fornirete saranno valutati in base a
 - Chiarezza e Leggibilità codice
 - Modularità (separation of concerns, alta coesione, basso accoppiamento)
 - Dovreste sempre verificare di
 - Attenervi ai principi di progettazione
 - Aderire alle buone norme di codifica
 - Usare design pattern, refactoring, test

Principi della progettazione OO

- **Information hiding**: ciò che non contribuisce alle caratteristiche essenziali di una classe deve essere nascosto (attributi ed alcuni metodi *private*)
 - Solo alcuni metodi forniscono ciò che si vuol far conoscere all'esterno
 - Avere caratteristiche nascoste limita la propagazione dei cambiamenti
- **Alta Coesione**: alta coesione indica alto grado di connessione degli elementi all'interno di un modulo
 - Una classe è coesa se implementa un solo piccolo compito, quindi tutti i suoi elementi (metodi) contribuiscono a realizzare il **singolo compito**
 - Analogamente si può definire la coesione per un metodo
 - Principio di singola responsabilità
 - Classi poco coese (usate ad es. per visualizzare, trasformare e registrare dati) sono difficili da comprendere e da riusare

Coesione per un modulo

- Scala di coesione, dal più basso al più alto grado
 - **Coincidental cohesion**, quando non c'è una relazione significativa tra elementi di un modulo (non fare mai!)
 - **Logical cohesion**, esiste qualche relazione logica tra gli elementi di un modulo, es. editare tutti i dati, ma dati diversi e quindi in modo differente (non fare!)
 - **Temporal cohesion**, come logical ma gli elementi hanno relazioni temporali, es. moduli per init, termination, clean, etc. (non consigliabile)
 - **Communicational cohesion**, gli elementi del modulo hanno in comune il riferimento ad un set di dati (così così)
 - **Sequential cohesion**, i dati in output da un elemento sono in input al successivo elemento (bene)
 - **Functional cohesion**, tutti gli elementi servono a svolgere una singola funzione (molto bene)
 - **Data cohesion**, tutto è parte di una ben definita astrazione (ottimo)

Come determinare la coesione

- Descrivere lo scopo di un modulo in una frase
 - Se la frase è composta, ovvero contiene congiunzioni, virgole o più di un verbo ---> è probabile che il modulo abbia più di una funzione: logical o communicational cohesion
 - Se la frase contiene parole che indicano il tempo, come "prima", "quindi", "dopo" ---> si ha temporal cohesion
 - Se il verbo non è seguito da uno specifico oggetto ---> probabilmente si ha logical cohesion
 - Parole come "startup", "init" indicano temporal cohesion

Principi della progettazione OO

- Basso Coupling tra moduli
 - Il coupling è la misura del numero e del tipo di associazioni tra un modulo ed un altro
 - Un alto coupling complica il sistema poiché un modulo è più difficile da comprendere e cambiare
 - La complessità del sistema è ridotta se i moduli hanno la più debole forma di coupling

Tipi di coupling

- Dal peggiore al migliore
 - Content coupling, un modulo usa o modifica dati di un altro modulo (non fare mai!)
 - Common coupling, un modulo usa dati di un'area comune (non desiderabile!)
 - External coupling, due moduli condividono lo stesso formato di dati, es. protocollo di comunicazione (non desiderabile!)
 - Control coupling, un modulo passa ad un altro un (control) flag per dire cosa fare (non desiderabile!)
 - Stamp coupling, i moduli comunicano usando un certo tipo (ok), ma il tipo contiene più dati di quelli necessari
 - Data coupling, un modulo comunica passando solo i dati che il ricevente necessita (molto bene)

Principi della progettazione OO

- Conseguenze locali: un cambiamento in qualche punto del sistema software non dovrebbe causare problemi in altri punti
- Minimizzare ripetizioni: lo stesso codice ripetuto in più posti forza a decidere se un cambiamento deve essere ripetuto per le altre copie
 - Quindi non si soddisfa il principio di conseguenze locali
 - Un modo per minimizzare le ripetizioni è di avere frammenti piccoli (piccoli metodi, piccoli oggetti, piccoli package)
- Logica e dati insieme: i cambiamenti su uno si propagano sull'altro. Se sono vicini allora le conseguenze di cambiamenti saranno locali
- Simmetria: identificare ed esprimere chiaramente simmetrie rende il codice facile da capire
 - Simmetria: se ho il metodo `add()` ho il metodo `remove()`, gli attributi di un oggetto hanno tutti lo stesso tempo di vita
 - Simmetria: la stessa idea è espressa allo stesso modo ovunque appare nel codice