design patterns e GRASP

design patterns

- una coppia soluzione/problema particolarmente importante a cui viene dato un nome
- vengono espressi in un formato molto rigido, ad es.
  - nome
  - descrizione sintetica della soluzione
  - descrizione del problema
  - esempio
  - discussione
  - controindicazioni
  - benefici
  - pattern correlati
  - nomi alternativi
- introdotti dal libro
  C. Alexander, S. Ishikawa, M. Silverstain. “A Pattern Language - Towns-Building-Construction”. 1977. Oxford University Press.

(software) design patterns

- GRASP
  - General Responsability Assignment Software Patterns
- Gang of Four (GoF)
  - dal libro
    E. Gamma, R. Helm, R. Johnson, J. Vlissides. “Design Patterns”. Addison-Wesley. 1995
- ...molti altri meno famosi e meno importanti

GRASP

GRASP: information expert

problema

data una operazione, chi la fa?

soluzione

generiche responsabilità di “fare” sono assegnate a che ha i dati necessari

esempio

Partita verifica la correttezza di una Mossa. Giocatore dovrebbe chiedere informazioni a Partita.

controindicazioni

non sempre ok

- chi salva lo stato degli oggetti nel database?
- se gli oggetti si auto-salvano l’interazione con il DB è ovunque e gli oggetti fanno cose non relative al dominio (bassa coesione)

GRASP: high cohesion

problema

come aumentare la comprensibilità?

soluzione

ciascun oggetto deve solo avere funzionalità correlate

esempio

nessun oggetto che rappresenta il domino sa come interagire con il DB o con una GUI

controindicazioni

molti oggetti piccoli possono essere ineffcienti specialmente in architetture distribuite.

oggetti troppo piccoli devono comunicare molto con altri (alto accoppiamento)

GRASP: low coupling

problema

come ridurre l'impatto di variazioni al progetto

soluzione

ridurre al minimo le dipendenze tra gli oggetti

esempio

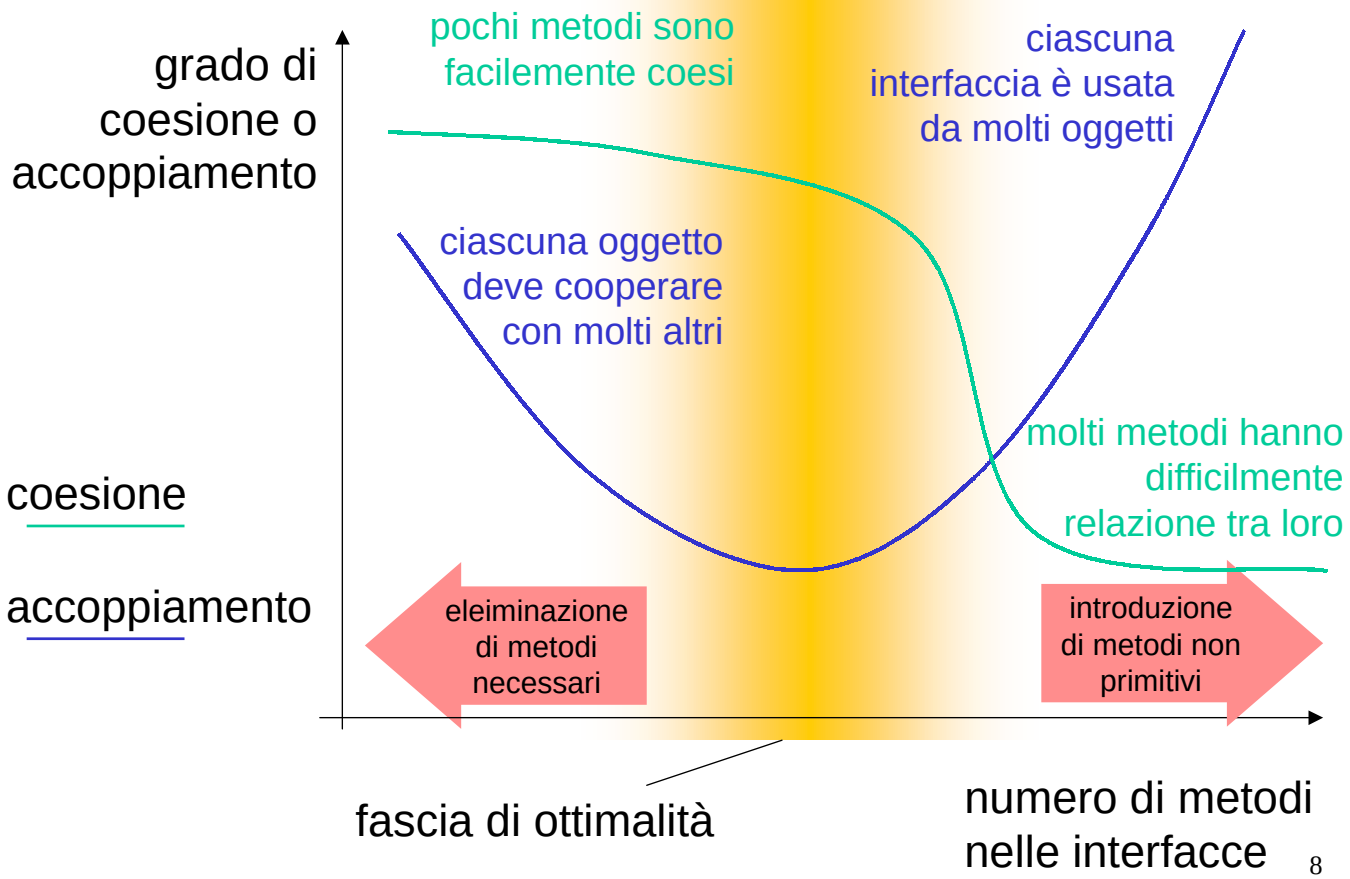
uso di interfacce semplici e stabili.

evitare di far riferimento a molti oggetti esterni.

controindicazioni

alto accoppiamento con software stabile è ok, es. accoppiamento con librerie standard del linguaggio in uso

GRASP: high cohesion vs. low coupling



GRASP: creator

problema

chi crea un oggetto?

soluzione

chi ha un rapporto stretto con l'oggetto, es: aggrega, usa, conosce, ha dati per inizializzazare (expert).

esempio

Partita aggrega istanze di StatoScacchiera e le crea

controindicazioni

la creazione può essere complessa, in tal caso si usano oggetti appositi (vedi GoF creational patterns)

GRASP: controller

problema

chi gestisce gli eventi in input al sistema?

soluzione

- un oggetto che rappresenta l'intera applicazione
- vari oggetti che rappresentano ciascuno uno use case

  "Windows", "applet", "view", "document" non sono mai controller!

esempio

la casse ControllerPagamento può gestire tutte le fasi di un pagamento

benefici

- disaccoppiamento tra interfaccia e rappresentazione del dominio
- coesione

GRASP: pure fabrication

problema

se le soluzioni ispirate al modello di dominio non sono buone cosa si fa?

soluzione

si possono creare classi “artificiali” al fine di supportare alta coesione e basso accoppiamento

esempio

PersistentStorageManager

benefici

alta flessibilità nel progetto

controindicazioni

va bilanciato con l’uso di soluzioni ispirate al modello di dominio (vedi Expert)

GRASP: indirection

problema

evitare accoppiamento diretto tra due o più oggetti per aumentare le possibilità di riuso

soluzione

inserire un oggetto intermedio

esempio

Leggere da file ci lega ad un particolare tipo di input. Leggere da un oggetto intermedio (che potremmo chiamare “stream”) ci permette di leggere sia da file che da socket e di inserire nuove possibilità in futuro.

benefici

basso accoppiamento

GRASP: protected variation

problema

come ridurre l'impatto di variazioni nel sistema

soluzione

identificare i punti di possibile variazioni e creare interfacce stabili attorno ad essi

benefici

- estensioni al sistema sono facili da aggungere
- nuove impementazioni di parti del sistema
- basso accoppiamento
- il costo dei cambiamenti è ridotto

GRASP: protected variation

esempi

- core
  - hiding (encapsulation), interfacce, polimorfismo, indirezione, standards
- data-driven design
  - rendere parametrizzabile il sistema in modo che il suo comportamento possa essere facilmente cambiato
- service lookup
  - UDDI, Jini, JNDI
- Interpeter-driven design
  - parte della logica è codificata in "regole" che vangono lette e interpretate

GRASP: protected variation

esempi

- reflection
  - il sistema è in grado di adattarsi ai cambiamenti del codice per mezzo di meccanismi di metalivello
- uniform access
  - metodi e accesso a campi supportati con la stessa sintassi (Ada, Eiffel, C#)
- Liskov substitution principle
  - protegge dalle variazioni delle classi derivate
- structure-hiding design (a.k.a. Don't Talk to Strangers, Law of Demeter)
  - inviare messaggi solo a: this e oggetti aggregati a this, parametri del metodo, oggetti creati nel metodo
  - cioè evitare: sale.getPayment().getAmount().getCurrency()

GRASP: protected variation

controindicazioni

- variation point: OK
  - punti del progetto in cui la variabilità è certa
- evolution point
  - punti del progetto in cui la variabilità è prevista
  - proteggersi dalle variazioni solo se la probabilità è alta
  - inutile progettare per variazioni che non avverranno mai
  - spesso riprogettare al momento opportuno è più semplice che progettare per prevenire tutte le possibili variazioni!