UP (UNIFIED PROCESS)

* Processo **iterativo** per lo sviluppo software OO evolutivo
* **Flessibile ai cambiamenti**
* Applicabile usando approccio **agile**:
  1. Iterazioni corte e timeboxed
  2. Raffinamento piani/requisiti/progettazione = i requisiti si definiscono un po’ alla volta e si programma fin da quasi subito
  3. Gruppi di lavoro auto-organizzati
  4. Sviluppo guidato da test
  5. Obiettivo comprensione piuttosto che documentazione
* Usa UML per pensare/comunicare 🡪 fase di MODELLAZIONE
* **Iterazioni iniziali** guidate da:
  1. Rischio
  2. Cliente
  3. Architettura
* Scenario di sviluppo = documento riassuntivo scelta pratiche e artefatti
* Fornisce approccio sistematico per trovare, documentare, organizzare i requisiti che cambiano:

1. SCRIVERE CASI D’USO CON CLIENTE
2. WORKSHOP CON SVILUPPATORI E CLIENTI
3. GRUPPI DI LAVORO
4. DIMOSTRAZIONI AL TERMINE DI OGNI ITERAZIONE

* Tipologie di requisiti secondo modello **FURPS+**

**F:** requisiti funzionali

**U:** facile usabilità del sistema, con documentazione e aiuto all’utente

**R:** affidabilità del sistema, capace di tollerare guasti e di essere ripristinato se necessario

**P:** Prestazioni = tempi di risposta, throughput, capacità e uso risorse

**S:** Sostenibilità = facile da mantenere (modificare, riparare, migliorare, adattare, configurare…)

**+:** Altro = vincoli di progetto, fisici, interoperabilità, operazionali…

* Definisce ruoli
* Produce artefatti = documenti/disegni/diagrammi/schemi
* USE-CASE DRIVEN
  + Contengono requisiti funzionali
  + Usati per pianificare iterazioni
  + Guidano analisi, progettazione, test
  + Influiscono nella scrittura del manuale utente e della *visione*
* Definisce il progetto in 4 fasi sequenziali:

1. IDEAZIONE
2. ELABORAZIONE
3. COSTRUZIONE
4. TRANSIZIONE
5. IDEAZIONE = Indagine di fattibilità (durata: breve)

* Decidere se progetto merita indagine più seria => si iniziano ad abbozzare gli artefatti!
* Workshop sui requisiti
* Analisi CASI D’USO (10% dettagliati, i restanti in formato breve)
* Analisi requisiti non funzionali più critici
* Analisi economica = stabilire grandezza progetto
* Preparazione ambiente di sviluppo = organizzare sviluppatori e strumenti
* **NON**: definisce piano di progetto affidabile, elabora tutti i requisiti
* **ARTEFATTI** (molti opzionali) abbozzati:
  + *MODELLO DEI CASI D’USO* = scenari tipici dell’utilizzo del sistema, dunque modello delle sue funzionalità.
    - Include diagramma UML dei casi d’uso, usato come modello di contesto del sistema e indice dei nomi dei UC
  + *SPECIFICHE SUPPLEMENTARI* = quello che non rientra nei casi d’uso (requisiti non funzionali o funzionali ma non esprimibili in UC), raccolgono altri attributi e requisiti compresi in URPS+
  + *GLOSSARIO* = dizionario dei dati (e metadati) e dei termini (+ sinonimi/pseudonimi) --> elimina ambiguità requisiti e migliora comunicazione
  + *VISIONE* = riassunto breve progetto (per dare visione comune) e requisiti di alto livello, riassume info del *modello dei UC* e *specifiche suppl*, riepiloga caratteristiche di sistema
  + *REGOLE DI BUSINESS* = regole di dominio, requisiti o politiche del progetto a cui si deve conformare il sistema definiscono come può funzionare in dominio (ES: per pagare con carta è richiesta la firma)
  + *LISTA E PIANO GESTIONE RISCHI*= descrizione rischi e idee per attenuarli
  + *PROTOTIPI E POF* = chiarire visione e validare idee tecniche
  + *PIANO DELL’ITERAZIONE* = cosa fare nella prima iterazione
  + *PIANO FASI E SVILUPPO SOFTWARE* = ipotesi su durata e sforzo fase di elaborazione
  + *SCENARIO DI SVILUPPO* = descrizione personalizzazione passi (= pratiche adottate) ed elaborati UP

NB: scopo elaborati è pensare, il loro sviluppo completo avviene nell’*elaborazione*

1. ELABORAZIONE = Implementazione iterativa dell’architettura core

🡪 Sfrutta feedback e workshop dalla costruzione incrementale di parti del sistema

🡪 Viene programmato e verificato il nucleo rischioso dell’architettura software

🡪 Vengono scoperti la maggior parte dei requisiti e stabilizzati

🡪 Vengono attenuati i rischi maggiori

🡪 Il team esegue indagine seria su requisiti e architettura

🡪 Viene generato codice e progettazione di qualità per il sistema finale = si iniziano a programmare versioni già utilizzabili nel sistema finale

🡪 ARTEFATTI: quelli dell’ideazione +

* + - * **Modello di dominio** = visualizzazione dei concetti del dominio
      * **Modello di progetto** = insieme diagrammi di progettazione logica (diagrammi classi software, diagrammi interazioni fra oggetti, diagrammi package…)
      * Documento dell’architettura software
      * Modello dei dati = Schema ER
      * Storyboard UC e prototipi UI

🡪 Durata di pochi mesi

**ITERAZIONE 1**

🡪 Implementa un sottoinsieme dei requisiti o degli UC completi

🡪 Si inizia la programmazione di qualità-produzione ed i test per il sottoinsieme di requisiti e si inizia dunque a sviluppare

**PIANIFICAZIONE ITERAZIONI SUCCESSIVE**

🡪 Fatta prima dell’inizio di ogni iterazione

* In base al rischio (tecnico, incertezza dello sforzo, usabilità)
* Copertura = coprire tutte le parti principali del sistema
* Criticità = funzioni che il cliente considera di elevato valore di business
* Spesso su più iterazioni viene distribuita l’analisi di un UC e anche quella dei suoi scenari alternativi: in questo modo si estende il sistema in modo graduale per gestire tutte le funzionalità richieste

**SYSTEM SEQUENCE DIAGRAM (SSD)**

**🡪** Elaborato della disciplina dei requisiti (non esplicitamente menzionati in UP)

🡪 Illustra eventi di input-output relativi al sistema che si sta sviluppando

* + - **EVENTO** = interazione fra attore e sistema: attore genera eventi di sistema, ossia richiede al sistema l’esecuzione di operazioni di sistema

🡪 Espressi attraverso diagrammi di sequenza UML

🡪 Se ne modella uno per ogni UC e per ogni suo scenario alternativo

🡪 Fa da input a *Modello di dominio* (per poi modellare gli oggetti) e a *contratti*

Struttura:

1. Attore primario – Sistema
2. Uno scenario alla volta
3. Passi che rappresentano l’iterazione fra attore e sistema
4. Iterazione = mostrata come messaggio con parametri

OPERATORI:

***ALT*** = indica alternativa ad un passo (if-else)

***OPT*** = indica opzionalità del passo (if)

***LOOP*** = indica che un passo può essere ripetuto più volte (for)

***PAR*** = indica che i passi possono essere seguiti in parallelo

***REGION*** = indica sezione critica in cui può essere in esecuzione un solo thread

**MODELLO DI DOMINIO**

🡪 Siamo ancora dentro la prima iterazione (e sviluppato nella disciplina di modellazione del business)

🡪 Mostra concetti significativi relativi ai casi d’uso

🡪 Può influenzare i contratti, glossario e modello di progetto

🡪 Lo ricaviamo dal testo degli UC

Caratteristiche:

1. Rappresentazione **visuale** delle classi concettuali (= oggetti reali del dominio)
2. Formato da **diagrammi di classi UML** con
   * **Classi concettuali** = rappresentazione concetto del mondo reale e del dominio di interesse
     1. *Simbolo* = rappresenta la classe
     2. *Intensione* = definizione in linguaggio naturale della classe
     3. *Estensione* = insieme di oggetti appartenenti alla classe

Per poterle individuare si possono usare:

🡪 Pattern di analisi specifici == modelli esistenti

🡪 Elenchi di categorie

🡪 Analisi linguistica

Classi Descrizione = contiene descrizione, usata nel pattern item-descriptor

Generalizzazioni = identificazione di caratteristiche comuni fra concetti che permette

di identificare concetto generale e concetto specializzato -> vale principio

sostituibilità

Astratta = se ogni elemento della classe è anche elemento di una delle sue

Sottoclassi (es: Pagamento 🡪 contanti, carta, assegno)

* + **Associazioni** = relazione fra classi significativa, persistente;

🡪 Le estremità si chiamano *ruoli* e volendo possono avere un nome;

🡪 Una classe può avere associazione riflessiva (= con se stessa);

🡪 *Aggregazioni* = relazione intero-parte

🡪 Composizione = aggregazione forte:

* + - * Ogni istanza della parte appartiene ad una sola istanza alla volta dell’intero
      * Ogni parte appartiene sempre ad un intero
      * La parte esiste solo se esiste l’intero

Immagine che contiene Rettangolo

Descrizione generata automaticamente

* + **Attributi** = valore logico (proprietà) degli oggetti della classe
    1. Dati primitivi
    2. Tipi enumerativi

Un attributo può essere derivato dalle informazioni derivanti l’associazione con altri oggetti => si indica con /nome\_attributo

1. Ci permette di:
   * Comprendere meglio il dominio
   * Definire linguaggio comune
   * Fonte di ispirazione per la progettazione

**CONTRATTI**

(🡪 Non esplicitamente menzionati in UP)

🡪 Servono da input per la progettazione di oggetti

🡪Utili per l’analisi dei requisiti e nella OOA

🡪Principali input:

* + - SSD
    - Modello di dominio
    - UC

🡪 A cosa servono: descrivono i cambiamenti nello stato del sistema dopo che ha eseguito un’operazione

STRUTTURA:

1. **OPERAZIONE** = nome e firma dell’operazione (con parametri e tipo)
   1. *Trasformazione* = sistema cambia il suo stato
   2. *Interrogazione* = sistema calcola e restituisce valori (non ha post-cond)
2. **RIFERIMENTI** = UC in cui si verifica l’operazione
3. **PRE-CONDIZIONI** = ipotesi necessarie sullo stato del sistema o degli oggetti del dominio affinché possa avvenire l’operazione; è una descrizione sintetica dello stato di avanzamento dell’UC ed è utile indicare gli oggetti rilevanti a quel punto dell’UC e che si vogliono poi menzionare nelle post-cond
4. **POST-CONDIZIONI** = descrive in dettaglio i *cambiamenti* degli oggetti concettuali nel modello di dominio dopo l’operazione

Cambiamenti:

* 1. Creazione oggetti
  2. Nuove associazioni / eliminazione associazioni
  3. Attributi modificati

DISCIPLINA DI PROGETTAZIONE (OOD)

COME PROGETTARE OGGETTI:

- 3 metodi:

1. Codifica
2. Disegno poi codifica 🡪 Pattern GRASP-GOF
3. Solo disegno

- Vale sempre il principio **LOW REPRESENTATIONAL GAP** = salto rappresentazionale basso fra come si pensa il dominio e la corrispondenza con gli oggetti software

- 2 tipi modelli per gli oggetti (relazionati fra di loro = si creano in parallelo)

1. **DINAMICO** = SD di UML -> rappresenta comportamento del sistema, interazione fra oggetti software per realizzare UC e metodi delle classi software

* Si pensa in modo dettagliato e preciso agli oggetti: quali devono esistere e come collaborano fra di loro
* Applica:
  + 1. Progettazione guidata dalle **responsabilità (RDD = Responsability-Driven Development) 🡪 principi di assegnazione di responsabilità =** si sviluppano gli oggetti pensando alle loro responsabilità (si identifica, si attribuisce all’oggetto software, si identificano eventuali collaborazioni)
    2. Principi **GRASP** 🡪 **design pattern**

SD di UML:

* Illustrano **interazione** (= come due oggetti si scambiano messaggi per eseguire un compito) fra oggetti attraverso scambio di messaggi
* Un messaggio è inviato ad un oggetto a cui è assegnata la responsabilità di quel compito
* **DSD** (= Design SD) -> Diagramma di sequenza di progetto = SD utilizzato dal punto di vista software o di progetto

Immagine che contiene diagramma

Descrizione generata automaticamente



Lifeline: partecipanti all’interazione

Immagine che contiene diagramma

Descrizione generata automaticamente

**INVIO DI MESSAGGI:**

Immagine che contiene diagramma

Descrizione generata automaticamente



Oggetto ricevente è **istanza di una meta-classe**

**RICEZIONE RISPOSTE:**

Immagine che contiene diagramma

Descrizione generata automaticamente

2. linea msg di risposta (opzionale se non viene restituito niente)



1. Sintassi returnVar = msg(par)



Immagine che contiene diagramma, schematico

Descrizione generata automaticamente

**Self/this:** Per mostrare un msg che un oggetto manda a sé stesso.

Immagine che contiene diagramma

Descrizione generata automaticamente

Freccia tratteggiata = **chiamata asincrona**



**COLLEGAMENTI**:

Immagine che contiene diagramma

Descrizione generata automaticamente



Immagine che contiene diagramma

Descrizione generata automaticamente



**FRAME:**

Immagine che contiene diagramma

Descrizione generata automaticamenteImmagine che contiene tavolo

Descrizione generata automaticamenteImmagine che contiene diagramma

Descrizione generata automaticamente



Immagine che contiene diagramma

Descrizione generata automaticamente

**Occorrenza di iterazione =** riferimento ad una iterazione dentro un’altra iterazione che mette in relazione i relativi diagrammi

1. **STATICO** = Diagrammi delle classi UML -> definisce nomi package, nomi classi/attributi/firme e relative associazioni

**Design Class Diagram (DCD):**

* + - Diagramma delle classi di progetto
    - Fa parte del modello di progetto
    - Dal punto di vista concettuale usato per visualizzare il modello di dominio
    - Dal punto di vista software permette di modellare gli oggetti software
    - Usati per indicare le operazioni (firma + params)

Immagine che contiene diagramma

Descrizione generata automaticamente

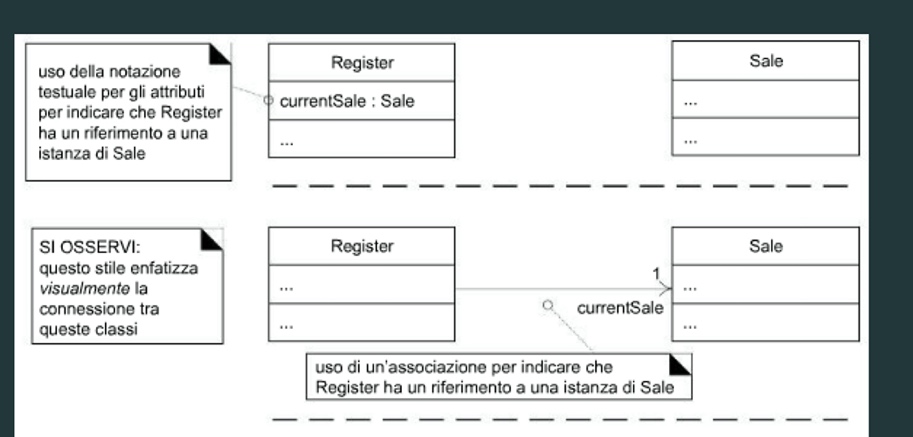
Immagine che contiene diagramma

Descrizione generata automaticamente

Linea di dipendenza = indica relazione di dipendenza fra le due classi se *SubclassFoo* cambiasse allora potrebbe influire anche su *Fruit*



**ATTRIBUTI:**



- Se non è indicata la visibilità solitamente sono ***privati***

- La freccia indica che la classe *Register* ha un attributo *currentSale* di tipo *Sale*



Immagine che contiene testo

Descrizione generata automaticamente



**OPERAZIONI**:

* + - Sono dichiarazioni di metodi con sintassi:

*visbility name (params) : return-type {property string}*

* + - Di default sono pubblici

**DECORATORI:**

Immagine che contiene tavolo

Descrizione generata automaticamente



**STEREOTIPI**:

* + - Usati per raffinare un concetto di modellazione esistente
    - Definiti dentro una collezione di stereotipi

Immagine che contiene diagramma

Descrizione generata automaticamente



**INTERFACCE**:

* + - Implementazione di un’interfaccia = realizzazione di interfaccia
    - LOLLIPOP = classe X implementa interfaccia Y
    - SOCKET = classe X richiede interfaccia Y

Immagine che contiene diagramma

Descrizione generata automaticamente



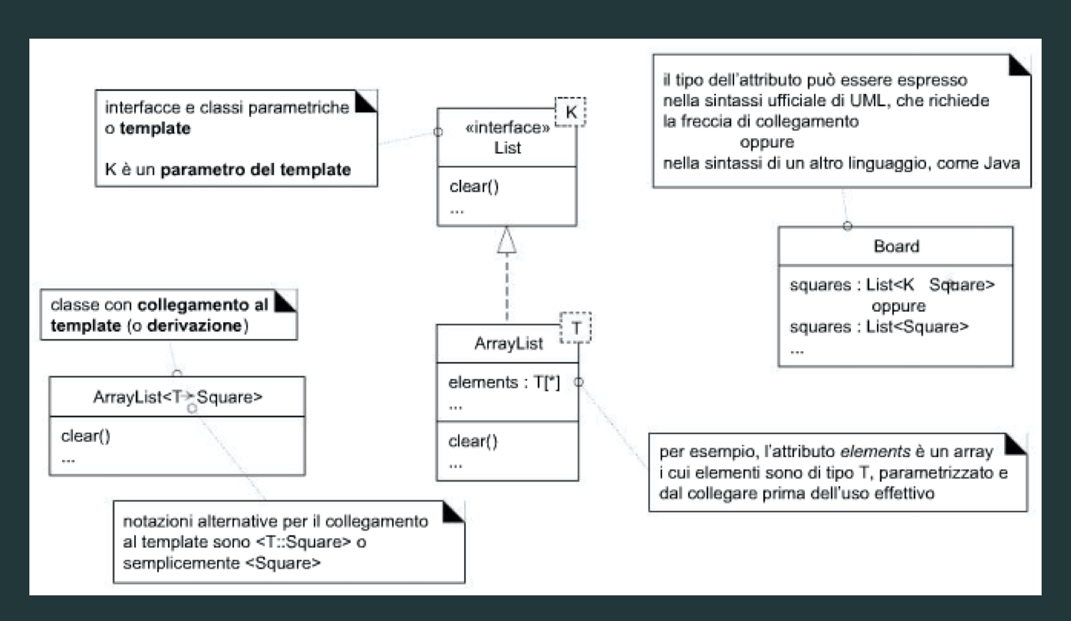
**SINGLETON:**

* + - Esiste una sola istanza di una classe

Immagine che contiene testo

Descrizione generata automaticamente

**TEMPLATE (= GENERICI):**



INPUT DELLA PROGETTAZIONE AD OGGETTI:

* + - 1. Testo UC
      2. SSD
      3. Contratti

OUTPUT:

1. Diagrammi UML di interazione (DCD e DSD)
2. Abbozzi prototipi interfaccia utente
3. Modelli basi di dati

**GENERAL RESPONSABILITY ASSIGNMENT SOFTWARE PATTERNS (GRASP)**

**RESPONSABILITA’ =** contratto; correlate al comportamento dell’oggetto in base al suo ruolo; realizzate tramite oggetti ed i loro metodi

Due tipi:

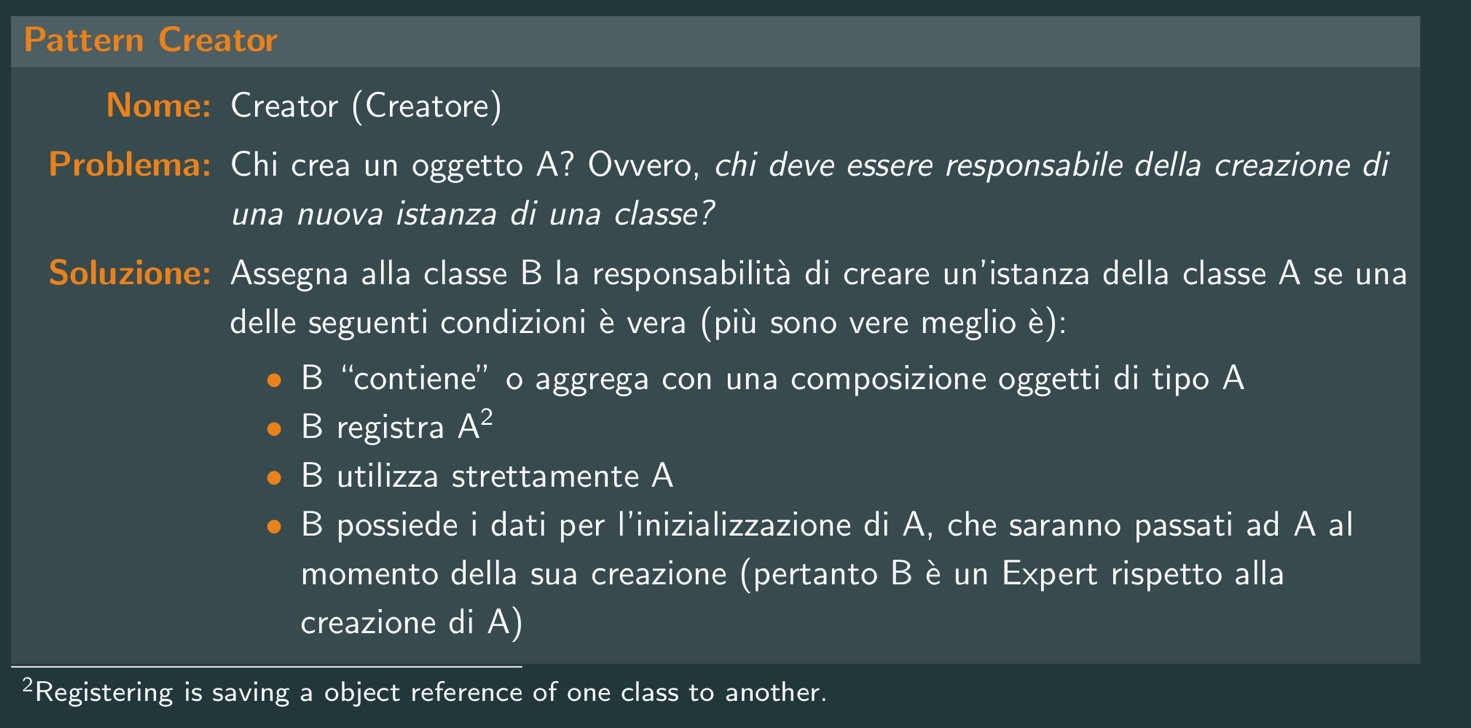
1. **DI FARE** 
   1. creare oggetto
   2. eseguire calcolo
   3. chiedere ad altri oggetti di eseguire azioni
   4. coordinare attività altri oggetti
2. **DI CONOSCERE**
   1. i propri dati
   2. oggetti correlati
   3. cose che può derivare

**OBIETTIVO GRASP:**

* + - aiutare a comprendere l’assegnazione delle responsabilità
    - permettere di progettare un sistema facile da comprendere, mantenere estendere e riutilizzare
    - **Progettazione Modulare** = software deve usare moduli **coesi** e **debolmente accoppiati**

**PATTERN = principi codificati in formato strutturato che descrivono un problema di progettazione ricorrente e la sua soluzione**

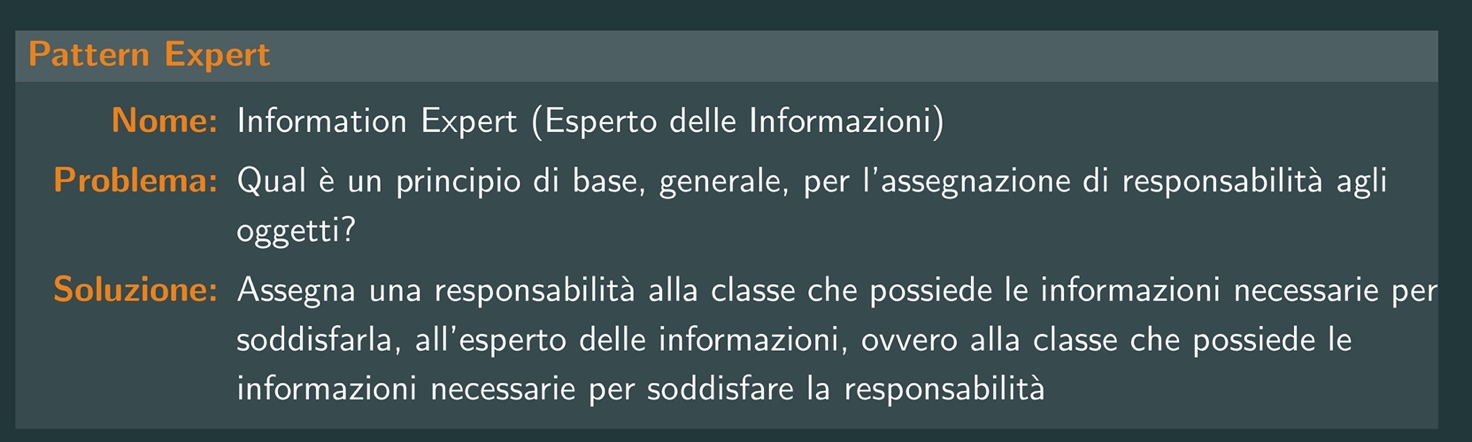
**CREATOR**

****



* + - **Un oggetto composto è buon candidato per creare le sue parti**

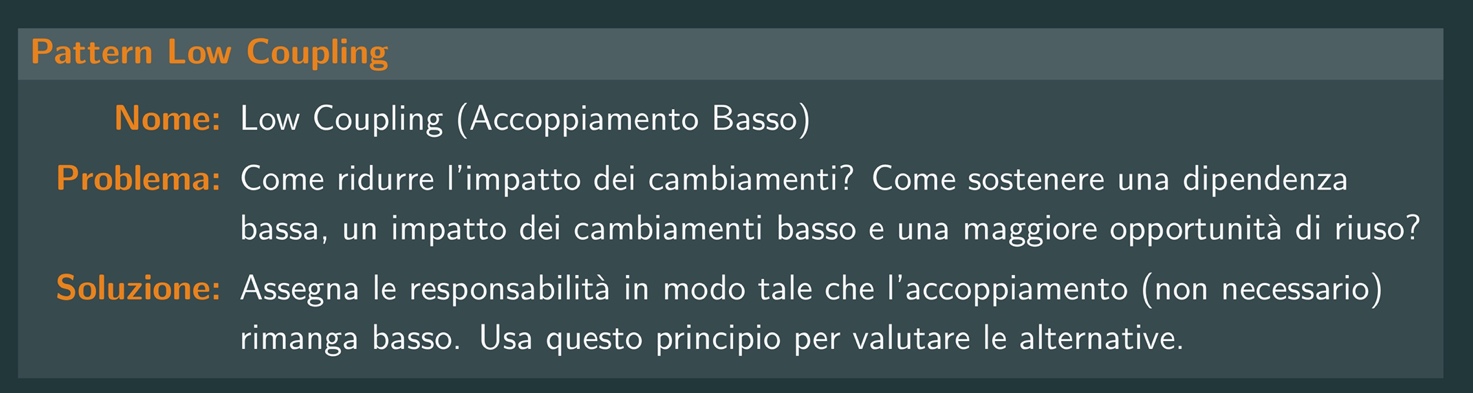
**INFORMATION EXPERT**

****



* + - **Ci sono classi esperte di informazioni parziali => collaborano insieme per realizzare l’obiettivo**

**LOW-COUPLING**



* + - **Misura di quanto un elemento è fortemente connesso/dipenda ad altri: forte accoppiamento obbliga a cambiamenti locali, aumenta complessità delle classi e rende difficile il riuso**
    - FORME COMUNI DI ACCOPPIAMENTO:
      * **X (classe) ha attributo di tipo Y**
      * **X referenzia istanza di tipo Y**
      * **Oggetto di tipo X richiama operazioni di oggetto di tipo Y**
      * **Oggetto di tipo X crea oggetto di tipo Y**
      * **X ha un metodo che contiene un elemento di tipo Y**
      * **X è sottoclasse di Y**
      * **Y è interfaccia e X la implementa**

**HIGH COHESION**

**Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente**

* + - Misura di quanto sono collegate fra loro le responsabilità di un elemento
    - **Alta coesione** = responsabilità altamente correlate + esecuzione giusta quantità di lavoro
    - **FORME DI COESIONE:**
      * **Coesione dati = classe implementa tipo di dato**
      * **Coesione funzionale = elementi della classe svolgono una singola funzione**
      * **Coesione temporale = elementi raggruppati perché usati nello stesso tempo**
      * **Coesione per pura coincidenza = MALE!**

**CONTROLLER**

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente



* + - Controller riceve e gestisce messaggi legati alle operazioni di sistema (non sono classi di dominio, ma fa parte dello strato di dominio)
    - ! controller MVC -> fa parte della UI e gestisce l’interazione con l’utente e la sua implementazione dipende dalla tecnologia UI utilizzata e dalla piattaforma
    - Non esegue il lavoro ma lo delega agli oggetti expert
    - Collega UI a strato della logica applicativa:
      * Rappresentando un sistema complessivo
      * Oggetto che rappresenta un dispositivo all’interno del quale è eseguito il software
      * Oggetto che rappresenta UC/sessione
    - Oggetti della UI catturano eventi generati dagli attori e delegano le richieste di operazioni
    - NEL CASO 2:
      * Si utilizza la stessa classe Controller per tutti gli eventi di sistema di quel’ UC
      * Una sessione è istanza di una conversazione con l’attore
    - **FOCADE CONTROLLER** = rappresenta sistema complessivo, sta sopra gli altri layer dell’applicazione ed è punto di accesso per le chiamate ai layer sottostanti -> atatto quando non ci sono troppi eventi da gestire
    - **CONTROLLER DI UC** = un controller diverso per ogni UC, usato per supportare il sistema

ESEMPI DI PROGETTAZIONE CON GRASP

1. *Scegliere il controller per il messaggio dell’operazione di sistema* = oggetto sistema complessivo/sottosistema, dispositivo speciale, punto d’accesso, ricevitore, gestore di tutti gli eventi di sistema di uno scenario d’uso
2. *Scegliere il Creator adeguato*
3. COSTRUZIONE

🡪 Si dovrebbero stabilizzare i requisiti principali

* In 2-3 (anche 1, ma poco) si svolgono attività nell’ambito di ***discipline*** e le loro inter-relazioni



Immagine che contiene grafico

Descrizione generata automaticamente

🡪 CORE WORKFLOW

🡪 Insieme di attività in una determinata area lavorativa

🡪 Risultato: **artefatti raffinati** = CORE WORKOUT

DISCIPLINA DEI REQUISITI

* Scoprire cosa deve essere costruito e orientare sviluppo del sistema
* ***Requisisti di sistema*** = capacità e condizioni a cui il sistema deve essere conforme (nel linguaggio del committente)
* Ogni requisito ha:
  1. Breve descrizione
  2. Stato (proposto, approvato…)
  3. Costo stimato per implementazione
  4. Priorità
  5. Rischio associato all’implementazione

IN UP:

1. **Produrre lista requisiti potenziali** = usata anche per stimare taglia progetto e decidere sequenza iterazioni
2. **Capire contesto del sistema** (due approcci possibili)

* **MODELLAZIONE DEL DOMINIO** = descrive concetti del sistema importanti come ***oggetti*** e li mette in relazione tramite associazioni
* MODELLAZIONE DEL BUSINESS = descrive processi di business ossia insieme di attività correlate e strutturate che producono uno specifico servizio o prodotto
  + Super-insieme del modello di dominio
  + Prodotto dell’ingegneria del business
  + Scopo: migliorare i processi di business

Per i due approcci si usa il diagramma UML dei UC.

1. **Individuare requisiti funzionali e non**
   * FUNZIONALI: sono descrizioni testuali, catturati tramite UC
   * NON FUNZIONALI: inclusi nei casi d’uso se in relazione con il requisito funzionale descritto nel UC, altrimenti, quelli più generali si trovano nelle *specifiche supplementari*

🡪 UC & SS: input per definire modello di dominio

🡪 Non viene considerato il modello di business!

CASI D’USO (UC)

* Collezione di **descrizioni testuali** di scenari d’uso del sistema interessanti, ossia descrivo l’interazione fra sistema e attore per raggiungere un obiettivo specifico
* 10% degli UC scritto nella fase di *ideazione* usando appositi template
* ***Attori*** = qualcosa o qualcuno dotato di comportamento, ruolo svolto da persone, organizzazioni, software o macchine
  + *Attore primario* = raggiunge obiettivi utente ed usa i servizi del sistema
  + *Di supporto* = offre servizio al sistema, utile per chiare interfacce esterne e protocolli
  + *Fuori scena* = ha interesse nel comportamento del UC, utile per garantire che tutti gli interessi vengano soddisfatti
* ***Scenario*** = sequenza specifica di azioni ed interazioni fra sistema e alcuni attori, descrive un particolare uso del sistema
* Utili a rappresentare i requisiti in input alla OOA/D, ma non sono essi stessi OO
* Definiscono i **contratti** in relazione al comportamento del sistema
* La loro stesura deve coinvolgere l’utente che userà il software; infatti, UC devono mettere in risalto gli obiettivi degli utenti e il loro POV, gli usi tipici
* Permettono dunque di scoprire e definire i requisiti del sistema concentrandosi sulla comprensione di ciò che l’attore considera un risultato di valore
* Formato a due colonne per enfatizzare interazione fra attore e sistema
* Devono essere scritti in uno stile essenziale (= narrazione), concentrandosi sull’obiettivo utente (= intenzione dell’utente del sistema) e le responsabilità del sistema (= cosa deve fare senza decidere come progettarlo, quello verrà fatto nella fase di progettazione)

**FORMATI**:

🡪 **Breve** = riepilogo conciso dello scenario principale di successo, utile a capire argomento e portata --------- (1 paragrafo)

Da non descrivere come un algoritmo ma narrazione delle interazioni fra attore e sistema che scatenano un’azione del sistema = requisito

🡪 **Informale** = testo informale, più dettagliato del formato breve ma con stesse finalità, riassume i vari scenari --------- (più paragrafi)

Descrive scenario principale (di successo) + scenari alternativi (di fallimento)

🡪 **Dettagliato** = descrive nel dettaglio tutti i passi e le variazioni, include pre-condizioni e garanzie di successo, si scrive partendo da uno dei due formati precedenti

Immagine che contiene testo, tavolo

Descrizione generata automaticamente

OSSERVAZIONI:

1. ***Scenario principale di successo***: descrizione (sequenza di passi) percorso comune che soddisfa gli interessi delle parti interessate.
   1. **Passi** = si possono ripetere ma non sono condizioni o diramazioni; il primo indica l’evento che scatena lo scenario; sono di tre tipi
      1. Interazione fra attori = attore/sistema, sistema/attore, sistema/altro sistema
      2. Cambiamento di stato del sistema
      3. Validazione (di solito fatta dal sistema)
2. ***Estensioni***: descrive tutti gli scenari alternativi al principale, sia di successo che di fallimento, sono azioni specifiche; dunque, ha passi che differiscono da quello principale = è diramazione dello scenario principale di successo. Suddiviso in:
   1. Condizione = descrizione situazione alternativa
   2. Gestione = passi differenti (anche uno solo), dopo l’ultimo si può tornare allo scenario principale

Sono usate per gestire tre situazioni tipiche:

1. Attore vuole che l’esecuzione del caso d’uso proceda in maniera diversa dallo scenario principale
2. Il sistema si accorge che il caso d’uso deve procedere diversamente da quanto previsto

**Numerazione estensioni**

* 1a, 1b 🡪 sono alternative al passo 1 dello scenario principale di successo
* 1a.1 🡪 primo passo dell’alternativa la passo 1 dello scenario principale
* (2-3)a 🡪 scenario alternativo che può occorrere al ai passi 2 oppure 3

**COME TROVARE UC**

1. Scegliere i confini del sistema
2. Identificare attori primari = sempre esterni al sistema, aiutano a definire confini
3. Identificare obiettivi di ogni attore primario
4. Definire UC che soddisfano gli obiettivi degli utenti, devono essere utili

**TEST DI UTILITA’ UC**

1. **Test del capo** = chiedersi se il capo sarà felice del nostro lavoro giornaliero
2. **EPB** (Elementary Business Process) = attività svolta da una persona in un determinato luogo e tempo in risposta ad un evento di business che aggiunge un valore di business misurabile e lascia i dati in uno stato consistente
3. **Dimensione** = valutare se UC contiene diversi passi e nel caso del formato dettagliato prevede dalle 3-10 pagine di testo

**LIVELLO**

* **OBIETTIVO UTENTE** = UC di tipo EPB perché permettono all’utente di raggiungere un proprio obiettivo di valore mediante il singolo utilizzo del sistema
* **DI SOTTO-FUNZIONE** = rappresenta funzionalità del sistema

UC e UP:

