Ingegneria del Software mod. A
Metodiche standard di sviluppo industriale

Docente: Tullio Vardanega
tullio.vardanega@math.unipd.it

Gradi di libertà

- Segmento di ciclo di vita
  - Determinato dall'intervallo temporale tra la prima e l'ultima revisione *esterna*
    - Nel nostro caso, l'intervallo tra **RR** (termine gara d'appalto) ed **RA** (fine del modulo 2 del corso)
- Modello di ciclo di vita *interno*
  - Adottato dal fornitore *entro* tale segmento
    - Determina il piano d'uso delle risorse disponibili
      - Persone, capacità, strumenti

Pianificazione - 1

- Relazione tra compiti, persone e sforzo richiesto
  - ***The Mythical Man-Month**, Frederick P Brooks, Jr (1975)*
  - Analisi delle componenti non riducibili
    - Compiti non partizionabili
      - Per necessità: p.es., un solo ambiente di prova
      - Per scelta: p.es., per preservare integrità concettuale
    - Compiti che richiedono comunicazione ed interazione tra membri del progetto
      - Ri-coordinarsi sistematicamente costa molto sforzo
    - Verifiche a livello sistema
      - Il sistema è uno ed è disponibile solo a fine sviluppo

Pianificazione - 2

- Aggiungere persone ad un gruppo di progetto
  - Aggiunge complessità
    - A tutti i compiti (*task*) che richiedono comunicazione ed interazione
  - Aggiunge inefficienza
    - A tutti i compiti non partizionabili
- Una buona pianificazione (insieme a buona analisi e buona progettazione architetturale)
  - Costa sforzo a monte
    - Quando ancora c'è tempo
  - Risparmia sforzo a valle
    - Quando le risorse ormai cominciano a scarseggiare

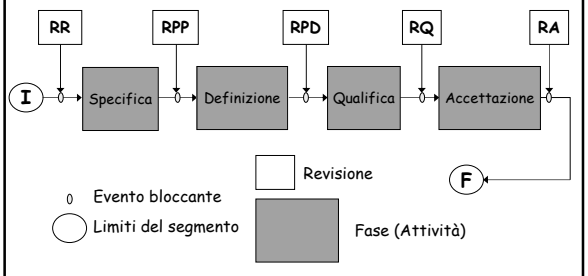
Modelli di ciclo di vita interno - 1

- Non tutti i modelli si adattano allo stesso modo agli adempimenti esterni richiesti dal progetto
- La scelta interna è libera, ma comporta oneri variabili
- La scelta è spesso per prodotto (per progetto)
  - Indipendente dall'organizzazione di appartenenza

Modelli di ciclo di vita interno - 2

- Sequenziale
  - Vantaggi
    - Impone disciplina al progetto
    - Comporta verifica sul completamento di ogni fase
  - Svantaggi
    - Richiede un notevole sforzo di documentazione
    - Allontana la percezione del prodotto (analisi e progetto) dalla sua realizzazione (codifica e qualifica)
  - Si presta bene a
    - Progetti a rischio contenuto, con poche dipendenze dall'esterno e limitati impatti sull'esterno

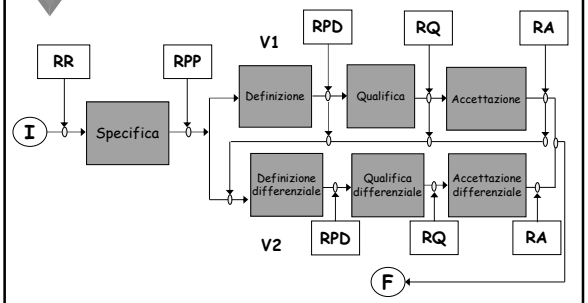
Modello sequenziale



Modelli di ciclo di vita interno - 3

- Incrementale
  - Vantaggi
    - Agevola il dialogo cliente-fornitore approssimando progressivamente il prodotto finale
    - Tollera una limitata fluttuazione (e maturazione) di requisiti
  - Svantaggi
    - Degenera facilmente nel circolo vizioso "*code and fix*"
    - È difficile decidere *quante* versioni intermedie siano utili o necessarie
  - Si presta bene a
    - Progetti nei quali l'integrazione del prodotto finale ha elementi di incertezza tecnica e finanziaria

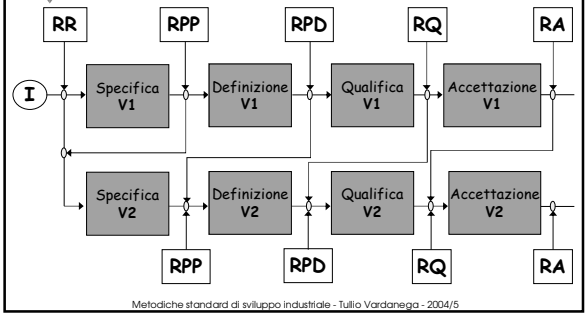
Modello incrementale



Modelli di ciclo di vita interno - 4

- Per evoluzioni successive
  - Vantaggi
    - Quelli del modello incrementale, ma all'interno di un contesto più formale
  - Svantaggi
    - Aggiornamento dei documenti di specifica e definizione (AR, ST, DP) per ogni versione successiva
  - Si presta bene a
    - Progetti nei quali la definizione dei bisogni è particolarmente laboriosa

Modello ad evoluzioni successive



Modelli di ciclo di vita interno - 5

- A spirale
  - Vantaggi
    - Favorisce l'esplorazione di alternative e promuove pratiche di riuso
  - Svantaggi
    - L'analisi delle alternative e dei rischi ed il contenimento dei costi di prototipazione richiedono esperienza
  - Si presta bene a
    - Progetti nei quali l'analisi dei rischi beneficia e coinvolge anche il cliente

Progettazione software - 1

- Comprende specifica e definizione del prodotto
  - Definizione IEEE
    - Il processo di definizione dell'architettura, delle componenti, delle interfacce e delle altre caratteristiche di un sistema o di una sua componente
    - Il prodotto di tale processo

Progettazione software - 2

- Processo interno al processo di sviluppo
  - Procede dall'analisi dei requisiti
  - Produce una descrizione della struttura interna e dell'organizzazione del sistema
    - Fornisce la base della realizzazione
  - Architettura *software* (logica e di dettaglio)
    - Decomposizione ed organizzazione del sistema in componenti ed interfacce tra esse
    - Il livello di dettaglio deve essere sufficiente a guidarne la realizzazione

Progettazione software - 3

- Attività 1 : Progetto (*design*) architetturale
  - Definisce la struttura e l'organizzazione del sistema secondo una visione ad alto livello
  - Identifica le componenti
    - Entità funzionalmente coese e suscettibili di implementazione mediante ulteriore decomposizione
- Attività 2 : Progetto di dettaglio
  - Ciascuna componente è descritta ad un livello sufficiente per determinarne la codifica

Nozioni fondamentali

- Limiti intrinseci del disegno
  - Non tutti i problemi hanno una soluzione
  - Occorre comprendere
    - Obiettivi
    - Vincoli
    - Alternative
    - Rappresentazioni del problema e delle sue soluzioni
- Contesto del disegno
  - Fattibilità e verificabilità
    - Tecnica ed economica

Tecniche abilitanti - 1

- Astrazione
  - Dimenticare informazione (attributi specifici) per applicare operazioni uguali ad entità diverse
    - P.es., calcolare l'area di una figura piana
    - La base di una gerarchia di classi astrae rispetto alle classi più specializzate
    - Ad una astrazione corrisponde una concretizzazione
      - Mediante parametrizzazione
        - Una *generic unit* in Ada; un *template* in C++
      - Mediante specializzazione
        - Una *classe* in Java

Tecniche abilitanti - 2

- Grado di accoppiamento e di coesione
  - L'accoppiamento è misura dell'intensità della relazione tra moduli (*inter*)
    - La modifica di uno comporta la modifica dell'altro
      - Indesiderabile effetto domino
    - Forte accoppiamento → scarsa modularità
  - La coesione è misura dell'intensità della relazione tra i costituenti di un modulo (*intra*)
    - Forte coesione → buona caratterizzazione

Tecniche abilitanti - 3

- Decomposizione e modularità
  - Una buona decomposizione architetturale identifica componenti tra loro indipendenti
    - Disaccoppiati
    - Autosufficienti (funzionalmente coesi)
- Incapsulazione (*information hiding*)
  - Separare l'astrazione dal dettaglio implementativo
    - L'astrazione è pubblica (specifica di interfaccia)
    - Il dettaglio è noto solo all'autore

Tecniche abilitanti - 4

- Sufficienza
  - La definizione dell'astrazione è sufficiente a caratterizzare l'entità desiderata
- Completezza
  - L'astrazione esibisce tutte le caratteristiche richieste
- Atomicità
  - La definizione dell'astrazione non può essere convenientemente decomposta in astrazioni più primitive

Aspetti cruciali - 1

- Concorrenza
  - Se e come decomporre il sistema in entità attive concorrenti (processo, *task*, *thread*) assicurando
    - Efficienza di esecuzione
    - Atomicità di azione
    - Integrità di dati condivisi
    - Comunicazione e sincronizzazione
    - Predicibilità di ordinamento (*scheduling*)

Aspetti cruciali - 2

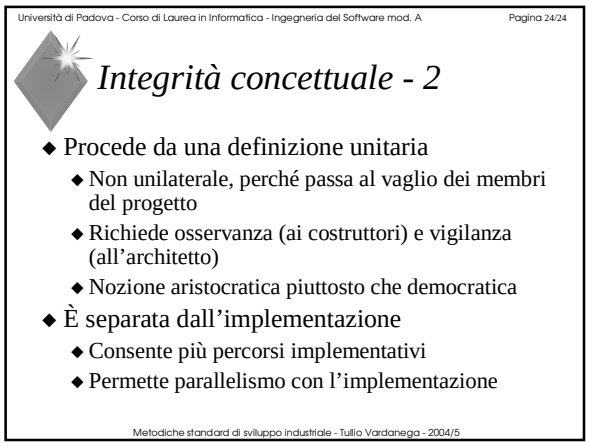
- Controllo e gestione degli eventi
  - Evento
    - Relativo al flusso dei dati
      - La disponibilità di un dato (dall'interno o dall'esterno)
    - Relativo al flusso di controllo
      - L'ingresso del sistema (o di una sua componente) in un particolare stato
    - Relativo al trascorrere del tempo
  - Come organizzare il flusso di dati e di controllo
  - Come trattare gli eventi temporali

Aspetti cruciali - 3

- Distribuzione
  - Se e come componenti software sono disseminate su più nodi di elaborazione
  - Come tali componenti comunicano fra loro
- Gestione degli errori e delle eccezioni
  - Come prevenire, trattare e tollerare eventi anomali (guasti, difetti interni, errori d'uso)

Integrità concettuale - 1

- Facilmente riconoscibile in una architettura fisica (p.es., un edificio)
  - Suggerisce uno stile uniforme, coerentemente applicato a tutte le parti del sistema ed alle loro interazioni
  - Bilancia capacità funzionale con semplicità d'uso e di concezione
    - L'una cresce senza penalizzare le altre
- Desiderabile in ogni architettura di sistema


Università di Padova - Corso di Laurea in Informatica - Ingegneria del Software mod. A
Pagina 24/24
Integrità concettuale - 2
Procede da una definizione unitaria
Non unilaterale, perché passa al vaglio dei membri del progetto
Richiede osservanza (ai costruttori) e vigilanza (all'architetto)
Nozione aristocratica piuttosto che democratica
È separata dall'implementazione
Consente più percorsi implementativi
Permette parallelismo con l'implementazione
Metodiche standard di sviluppo industriale - Tullio Vardanega - 2004/5