Il ciclo di vita del software

Il ciclo di vita del software

- Definisce un modello per il software, dalla sua concezione iniziale fino al suo sviluppo completo, al suo rilascio, alla sua successiva evoluzione, fino al suo ritiro
- Definisce dunque il processo attraverso cui il software evolve
  - si parla di software lifecycle o software process

I temi che affronteremo

- Perche` avere un modello di ciclo di vita?
- Il modello ideale (e tradizionale) a cascata
- Perche` il modello ideale a cascata non funziona (quasi mai)
- Dall'idea del "modello ideale" all'idea di tanti modelli possibili, tra cui scegliere in funzione delle circostanze
- Modelli evolutivi e flessibili

Argomenti trattati

- Motivare la necessita` di un modello di riferimento per la vita del prodotto software
- Introduzione del modello "a cascata"
- Inizio di discussione delle fasi in cui il ciclo puo` essere scomposto

La mancanza di un modello

- Non esiste un processo pianificato
- Lo sviluppo avviene in maniera spontaneistica (caotica?), e cio` causa problemi di
  - controllo dei tempi e dei costi
  - qualita` dei prodotti
- Talora si parla ironicamente di modello "code&fix" (vale a dire, l'assenza di modello)

La necessita` di un modello

- Qualunque "industria" ha un modello per la produzione dei beni
- Il modello consente di pianificare le attivita` e le risorse necessarie
- Consente di prevedere e controllare i costi del processo e la qualita` dei prodotti

La prima proposta di soluzione

Il ciclo di vita "a cascata"

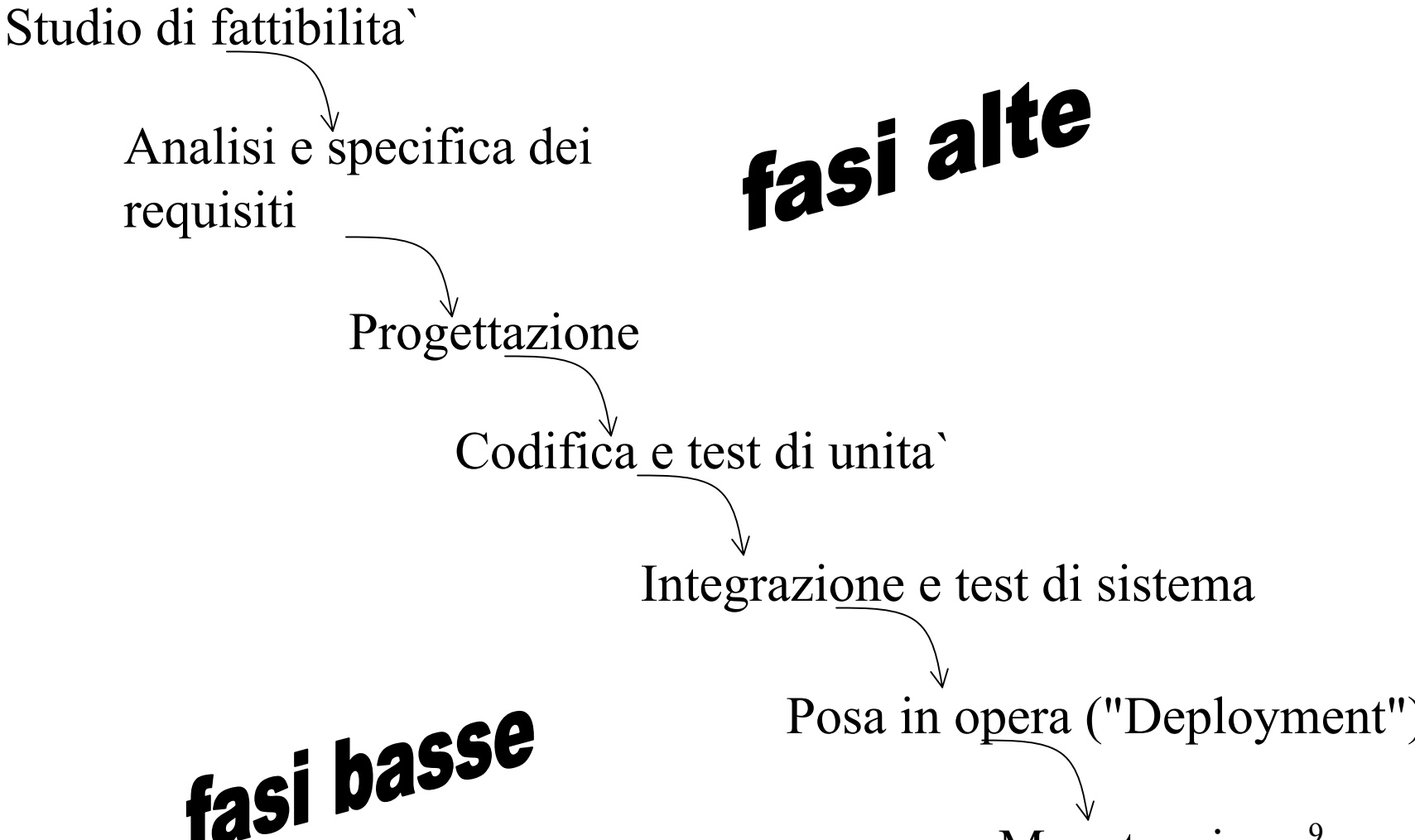
(fine anni '60—inizio anni '70)

Obiettivo del un modello

- Cercare un metodo sistematico che consenta di
  - identificare fasi e attivita` attraverso cui procedere
  - standardizzare gli output di ciscuna fase (semilavorati—"artifacts")
  - forzare un procedimento lineare di passaggio tra una fase e la successiva, a completamento avvenuto della fase
  - nessun ritorno all'indietro, considerato dannoso
    - impedisce una buona pianificazione e controllo

la produzione di software come catena di montaggio

Un modello a cascata ("waterfall model")



Esaminiamo le fasi

- Le fasi si ritroveranno anche in altri modelli di ciclo di vita
- Cio` che li distinguera` sara` la modalita` di organizzazione del processo, ma esse dovranno comunque essere effettuate

Studio di fattibilita`

- Effettua un'analisi dei costi/benefici
- Determina se il progetto debba essere avviato (buy vs make), le alternative, le risorse di massima necessarie
- Produce il documento "Studio di fattibilita`"
  - Definizione del problema problem description
  - Possibili soluzioni
  - Possibili costi e tempi per la soluzione

Analisi e specifica dei requisiti

- Analizza il dominio applicativo (il contesto per l'applicazione, il problema)
- Identifica i requisiti desiderati
- Deriva una specifica per il software (la "macchina" da realizzare
  - Necessaria l'interazione col committente/utente
  - Presuppone una conoscenza approfondita delle proprieta` del dominio
- Produce il documento "Analisi e specifica dei requisiti (ASR)"

Le 5 W

- Who
  - Chi usera` il sistema
- Why
  - Perche` svilupparlo? Perche` gli utenti dovrebbero volerlo usare?
- What (vs How)
  - Che cosa fornirà?
- Where
  - Dove verrà usato? Per quale architettura?
- When
  - Quando (e per quanto) verrà usato?

Il documento ASR

- Il dominio
- Le proprieta` richieste
  - Preciso
  - Completo
  - Consistente
- Potrebbe includere
  - Manuale utenmte vers.0
  - Piano per il test di sistema

Progettazione ("design")

- Definisce la "architettura software" complessiva
  - Componenti
  - Relazioni e interazioni tra componenti
- Obiettivo
  - Supportare lo sviluppo parallelo ("concurrent engineering")
  - Suddividere le responsabilita`
- Produce il "Documento di progetto"

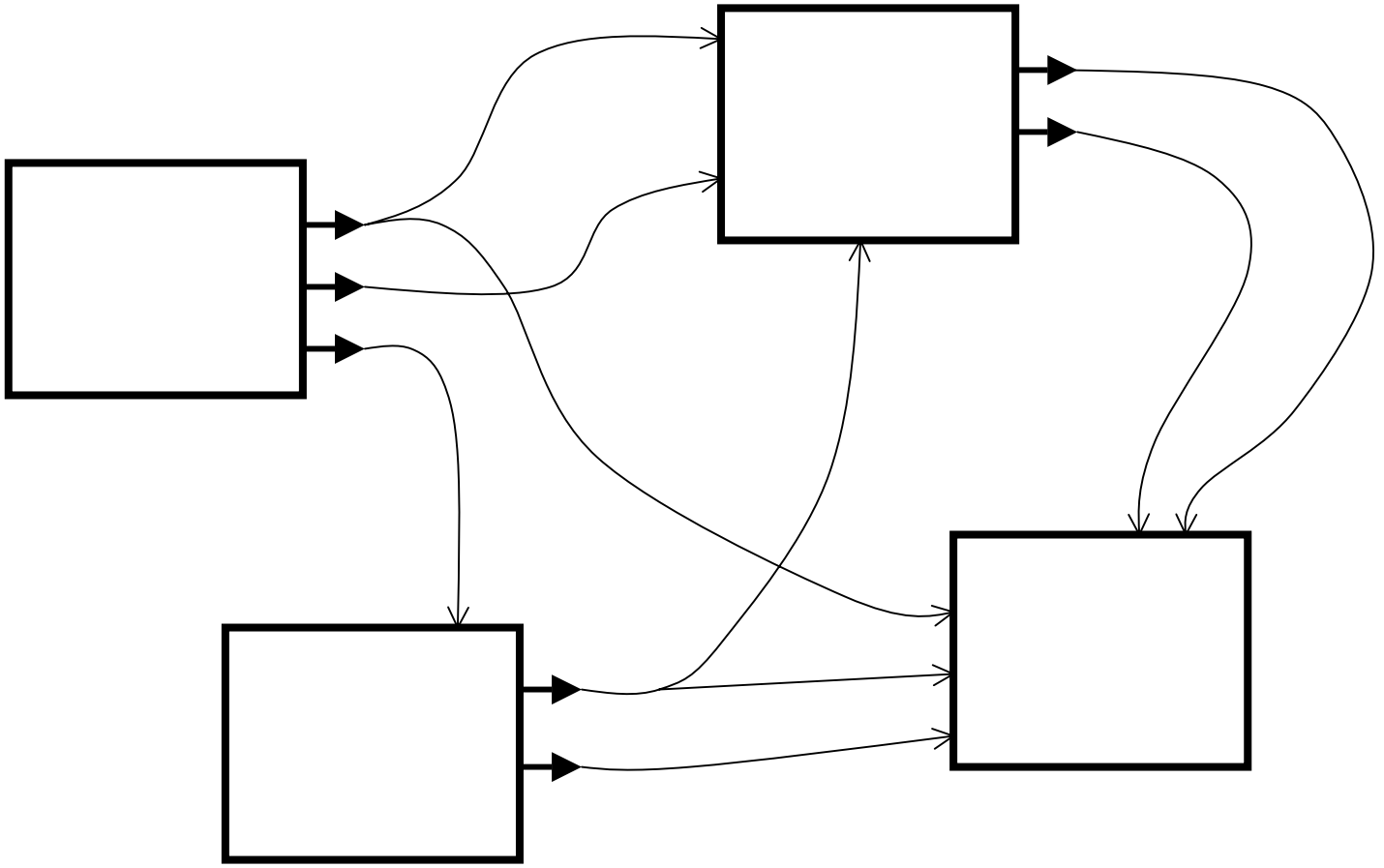
Livelli di progetto

- Dall'architettura complessiva in termini di sottosistemi
- Fino all'identificazione dei singoli moduli che comporranno il sistema
  - in C, definizione dei file ".h" che contengono le definizioni delle risporse esportate dai moduli verso altri moduli
  - meccanismi molto efficaci nei linguaggi orientati agli oggetti (Java)
    - classi e packages

L'idea del progetto modulare

- Il sistema complessivo e` suddiviso in parti
  - *divide et impera*
- Ogni parte (modulo) deve realizzare una funzionalita` logica autosufficiente nel sistema complessivo
- Deve essere chiaro che cosa un modulo offre agli altri moduli in termini di funzionalita'
  - *interfaccia del modulo*

Una visione grafica intuitiva



Modulo

- La scomposizione in moduli deve facilitare lo sviluppo concorrente e le successive modifiche
- Deve essere possibile modificare ciascun modulo senza impattare sugli altri moduli
- Un modulo deve "incapsulare" decisioni di progetto che si posso cambiare senza che se ne modifichi l'interfaccia

"Information hiding"

- Concetto introdotto da Dave Parnas, 1972
- Un modulo e` caratterizzato da cio` che nasconde agli altri moduli, e cio` da cio` che puo` essere modificato senza che gli altri moduli ne risentano

Codifica e test di unita`

- Ogni modulo viene implementato usando il linguaggio di programmazione scelto
- Ogni modulo viene testato isolatamente
- Ogni modulo include la propria documentazione

Integrazione e test di sistema

- I moduli vengono integrati in (sotto)sistemi e questi vengono ulteriormente testati
- Questa fase può essere integrata con la precedente in uno schema di implementazione incrementale
- Il test di sistema complessivo deve verificare le proprietà globali
- Si distingue a volte tra *alpha test* e *beta test*

Commento

- Le attivita` di "Codifica e test di unita`" e "Integrazione e test di sistema" potrebbero essere organizzate in maniera piu` flessibile e non in una rigida sequenza
- E` proprio il ciclo di vita "a cascata" che forza un processo sequenziale
  - *torneremo su questo argomento…*

"Tipica" distribuzione degli sforzi

- 18% requisiti e specifica
- 19% progettazione
- 34% codifica e test di unita`
- 29% integrazione e test di sistema

ATTENZIONE

- ampia variabilita`(anche piu` del 10%)

Posa in opera ("deployment")

- Obiettivo:
  - Distribuire l'applicazione e gestire le diverse installazioni e configurazioni sui computer dei clienti

Manutenzione

- Definizione
  - Le modifiche post-rilascio
- termine infelice
  - il software non e` soggetto a usura!
  - se succede un malfunzionamento, la causa era presente fin dall'inizio!
- Dato inquietante: e` responsabile per piu` del 50% dei costi totali
  - Indagine recente per le imprese di software dell'UE
    - 80% del budget IT speso in manutenzione

Manutenzione

- Include diversi fattori di cambiamento
  - correzione
  - evoluzione
- Terminologia (e fenomenologia)
  - Manutenzione correttiva $\approx$ 20%
    - non sono rispettate le specifiche
  - Manutenzione adattativa $\approx$ 20%
    - Far fronte a cambiamenti
      - dell'ambiente
      - delle condizioni al contorno (e.g., normative)
  - Manutenzione perfettiva $\approx$ 50%
    - Aggiunta nuove funzionalità (Eliminazione di quelle inutili)
    - Miglioramento di aspetti non funzionali (efficienza, interfaccia)

I malfunzionamenti (e gli errori che li causano) osservazioni dalla pratica

- Ci si puo' aspettare che il software consegnato contenga un numero di errori = 10% degli errori trovati durante il test
- Errori introdotti presto (p. es. errata analisi) sono scoperti tardi, e **piu` tardi vengono scoperti piu` costosa e` la loro rimozione**
- Numero di errori dipende dalla complessita` della struttura interna (intesa come flusso di controllo)
- Eliminare errori da software di grande dimensione e maturo (risultato di molti interventi manutentivi) costa di piu` (fino a 5-10 volte) di quanto costi intervenire su software piccolo e nuovo
- La rimozione di errori causa l'introduzione di nuovi errori
- Sistemi di grande dimensione tendono a stabilizzare il numero di errori presenti

Perche` il software evolve?

- Mutamenti nel contesto (man. evolutiva)
  - introduzione dell'EURO
- Mutamenti nei requisiti o requisiti non noti precedentemente (man. perfettiva)
  - L'applicazione introdotta induce nuovi fabbisogni
    - innovazione genera innovazione…
  - Recente indagine a livello EU
    - 20% dei requisiti utente sono obsoleti dopo 1 anno
- Specifiche errate (requisiti non colti, dominio non compreso) (*di quale manutenzione si tratta?*)

Re-ingegnerizzazione dei prodotti

Come gestire l'evoluzione?

- Se l'evoluzione del software e` cosi` connaturata con il tipo di prodotto

Il problema

- Le modifiche progressivamente deteriorano il software
  - I programmi perdono la loro iniziale struttura logica
  - Sorgono variabili inutili, sottoprogrammi che non vengono piu` chiamati, parti di codice irraggiungibile
  - La documentazione non e` piu` coerente con i programmi

La soluzione

- "Reverse engineering"
  - riportare il sistema in uno stato consistente
  - condizione necessaria per poter mettere mano al software
- "Re-engineering"
  - effettuare le modifiche necessarie, ad esempio re-implementare certe parte
  - talora si parla di "manutenzione preventiva"

Come gestire prodotti che evolvono continuamente e rapidamente?

Risposte (1)

- Metodi di progetto
  - Cercare di anticipare i cambiamenti
  - Progettare il software in modo da facilitare le modifiche successive in maniera economica ed affidabile
    - isolare le parte soggette a cambiamento all'interno di moduli, nascondendole all'esterno (incapsulamento, information hiding)
      - p.es. funzione C di cui mi basta conoscere nome e parametri, ma l'algoritmo e` nascosto
      - concetto di tipo di dato astratto

Risposte (2)

- Processo
  - alternative al ciclo di vita waterfall, che spesso ha grossi problemi (vediamo ora perche`)

Ciclo a cascata: problema 1

- La manutenzione, pur cosi` importante, e` nascosta in una fase finale che non viene disciplinata da un processo (non viene strutturata in termini di attivita` elementari)

Ciclo a cascata: problema 2

- Il ciclo a cascata richiede comunque un adattamento del processo allo specifico progetto
- Ne devono esistere variazioni per diverse tipologie, p.es.:
  - software sviluppato per uso personale
  - il committente appartiene alla stessa organizzazione
  - software "su misura" ("custom") sviluppato da un produttore per un committente specifico
  - applicazione ("pacchetto") per il mercato

Ciclo a cascata: problema 3

- La "cascata" presuppone che il dominio sia perfettamente noto, che il committente conosca esattamente cio` che vuole, che i requisiti vengano colti correttamente e che siano stabili
- Cio` capita molto raramente
- Non e` possibile evitare i ricicli: sono parte essenziale del nostro mondo

Trasparenza e flessibilita`

- Occorre che il processo consenta di verificare continuamente se cio` che si sta facendo corrisponde a cio` che desidera il committente
- Non appena possibile, correggere rotta
  - prima che sia troppo tardi
  - prima di aver spesso tutte le risorse…

Flessibilita`

- Adattarsi ai cambiamenti, in particolare nei requisiti
- Flessibilita` attraverso incrementalita`
  - processi incrementali in cui si fornisce qualcosa al committente, che cosi` puo` fornire un feedback
    - L'idea generale puo` incarnarsi in molte alternative pratiche

Prototipazione

- Un prototipo e` un modello approssimato dell'applicazione
- Viene usato per ottenere feedback o valutare qualche concetto
- “Che cosa” prototipare dipende da cio` che e` critico valutare (p. es., l'interfaccia per l'utente)
- Un prototipo puo` essere "EVOLUTIVO" o "USA E GETTA" ("throw-away")

Prototipi evolutivi

- Questa e` la soluzione di norma usata per il software (mentre in altri settori dell'ingegneria e` comune il prototipo *throw away*)
- Nel prototipo possono mancare parti oppure i componenti posso essere realizzati in maniera preliminare

Rilascio incrementale ("incremental delivery")

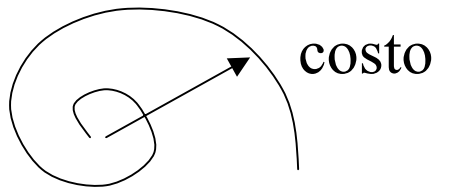
- Si identificano fin dall'inizio subset dell'applicazione da realizzare
- Su questi subset si ottiene un feedback dai committenti
- Spesso i subset iniziali sono quelli piu` critici per il successo dell'applicazione

Il ciclo di vita dei prodotti innovativi

- Esempio: i nuovi prodotti per Internet
- Qui l'incrementalita` e` fondamentale
- Incrementalita` implica rapida uscita sul mercato e fidelizzazione dei clienti
- Non esistono, spesso, requisiti da cogliere
- Spesso si tratta di "creare" una domanda
- Si creano versioni iniziali di prova da lanciare come "versioni beta"
- La rete stessa e` sia la "vetrina" dei prodotti che il mezzo distributivo

Il modello a spirale

- Il processo viene visto come un ciclo nel quale si iterano le attività di analisi, progetto, realizzazione/test e valutazione per successivi cicli
- Ad ogni ciclo, il costo (raggio della spirale) aumenta



Cicli di vita e attivita`

- I diversi cicli differiscono nel modo e nell'ordine in cui le diverse attivita` vengono eseguite
- Le diverse attivita`

  analisi, specifica, progettazione, implementazione, test

  in modi diversi e secondo fasi diverse, vanno comunque fatte