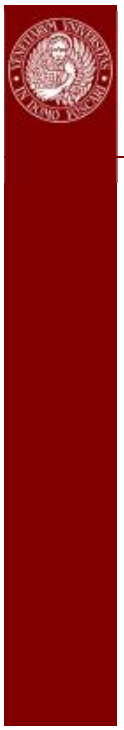
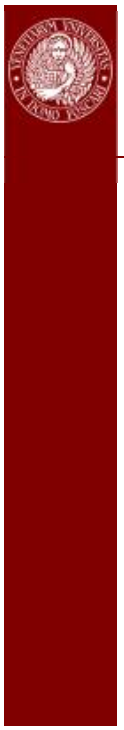


Corso di Project Management

I Modelli di Cicli di Vita del Software
(Software Lifecycle)

Roberto D'Orsi Anno Accademico 2013/2014



I modelli di Cicli di Vita del Software

Le fasi standard di un Ciclo di vita del Software

1. **Pianificazione e Programmazione (*Planning*):** individuare il problema da risolvere, identificare i requisiti principali, impostare il piano di massima, predisporre l'analisi costi/benefici e l'analisi dei rischi
2. **Elicitazione ed Analisi dei Requisiti (*Elicitation and requirements analysis*):** produrre una descrizione formale delle esigenze del Cliente sia in termini di cosa deve fare il software (requisiti funzionali), sia in termini di come il software deve essere utilizzato (requisiti non funzionali)
3. **Progettazione o Disegno (*Design*):** produrre, a partire dai requisiti, un modello formale e strutturato del sistema da implementare, che comprenda: l'architettura del sistema SW, i moduli e gli algoritmi da realizzare, il disegno dati, l'architettura tecnologica, le interfacce HW e SW
4. **Codifica (*Coding*):** tradurre nel linguaggio di programmazione prescelto le funzioni e gli algoritmi progettati

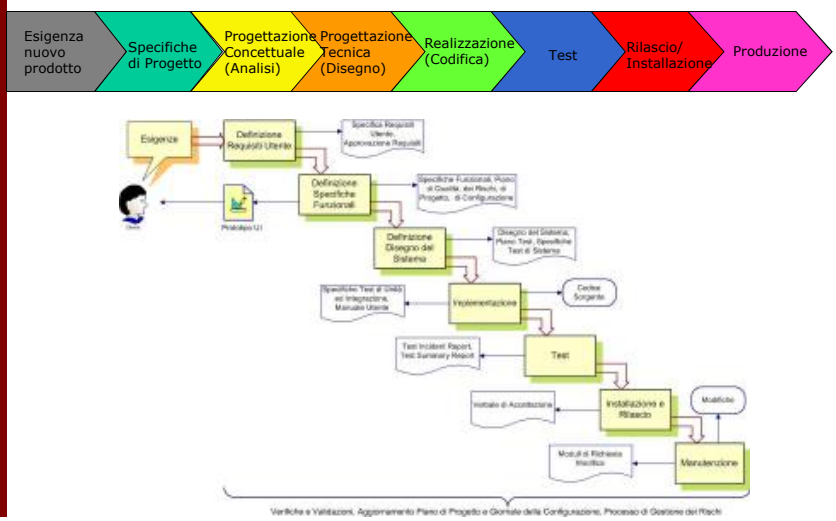
I modelli di Cicli di Vita del Software

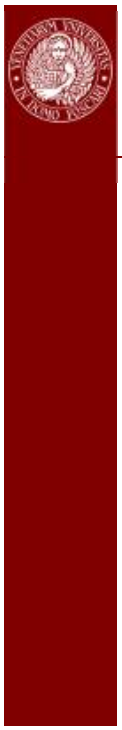
Le fasi standard di un Ciclo di vita del Software (segue)

5. ***Testing*:** anche se, dal punto di vista logico, il *test* viene posizionato a valle della codifica, prima della fase di *delivery*, in realtà l'attività di *testing* è distribuita lungo tutte le fasi ed ha l'obiettivo di verificare, passo passo, se i requisiti del Cliente sono soddisfatti
6. **Installazione e passaggio in produzione (*Delivery*):** il software viene installato nell'ambiente *target* ed integrato con l'ambiente esistente
7. **Manutenzione e Gestione (*Maintenance*):** riguarda sia la modifica del software già rilasciato per correggere errori o per tener conto di mutate esigenze, sia tutte le attività necessarie per la gestione ordinaria ed il supporto al prodotto (configurazioni, salvataggi, aggiornamenti, *tuning*, assistenza,…)
8. **Dismissione (*Phase-out o Retirement*):** fine della vita operativa del software a seguito di mutate esigenze o di sostituzione con un nuovo prodotto

I modelli di Cicli di Vita del Software

Il ciclo di vita a cascata (waterfall)

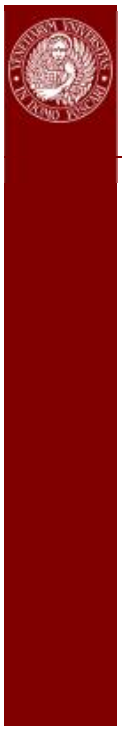




I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo waterfall (segue)

- Il ciclo di vita *waterfall* è il più utilizzato nei progetti, in particolare in quelli di grandi dimensioni
- Il ciclo di vita a cascata ha tra gli obiettivi quello della pianificazione di tutte le fasi del lavoro (infatti è anche detto "*plan driven*"): è un modello "predittivo" che ha l'obiettivo di cercare di non lasciare nulla al caso
- Per iniziare una nuova fase, a rigori, deve essere assolutamente conclusa quella precedente, in modo rigorosamente sequenziale
- In particolare la definizione dei requisiti è racchiusa in un'unica fase, all'inizio dell'intero processo (e spesso questo è un problema)
- Questo tipo di ciclo di vita può venire facilmente personalizzato, conglobando alcune fasi, pur rimanendo sostanzialmente sequenziale (vedi, ad esempio il caso del ciclo di manutenzione)
- Si presta bene ad un controllo efficace dei rischi, essendo i requisiti ben noti e definiti a priori
- Enfatizza gli aspetti organizzativi e gerarchici in quanto richiede la massima chiarezza su ruoli e responsabilità delle singole fasi. I controlli, le scadenze, i momenti di verifica, le dipendenze causali e temporali tra le fasi, sono molto ben definiti.

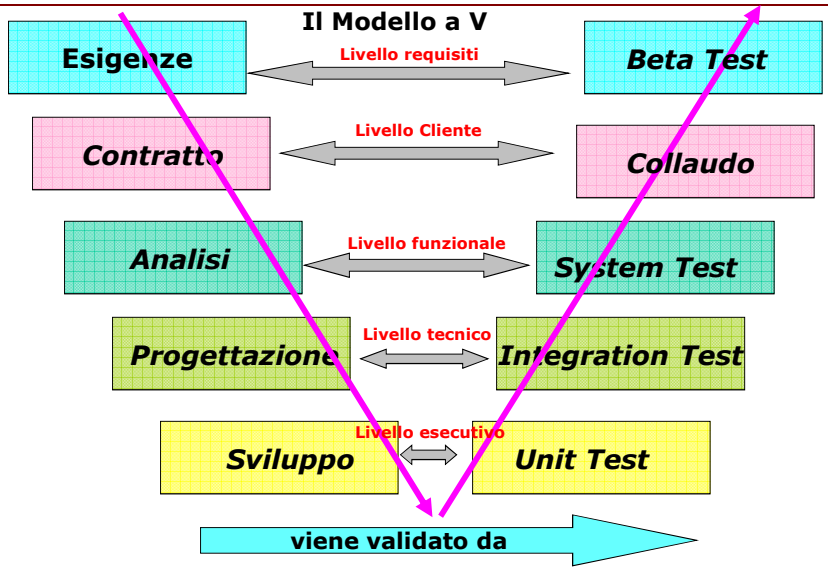


I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo waterfall

- La comunicazione tra le fasi è affidata ad una precisa documentazione standard (infatti è anche detto "*documentation driven*") che riporta ciò che è stato fatto e ciò che deve ancora essere completato
- Molto spesso è difficile per il Cliente dichiarare fin dall'inizio tutti i requisiti e il modello a cascata ha qualche difficoltà nel gestire la necessità di integrare e completare i requisiti in corso d'opera
- Inoltre il Cliente riesce a vedere una versione funzionante del sistema solo nelle ultime fasi del ciclo di vita: scoprire un errore concettuale così tardi potrebbe costare molto caro
- La rigorosa sequenzialità delle fasi può creare delle perdite di tempo nel gruppo di lavoro, dovuto ai tempi di attesa tra una fase e l'altra
- Non è adatto nei casi in cui i requisiti non siano chiari, stabili e completi fin dall'inizio
- Da un punto di vista generale quindi il modello *waterfall*, nella sua versione originale, è caratterizzato da una scarsa flessibilità, che male si adatta alla complessità di molti progetti
- In definitiva il ciclo *waterfall* è un modello ideale, che nella pratica viene spesso approssimato per poterlo applicare

I modelli di Cicli di Vita del Software



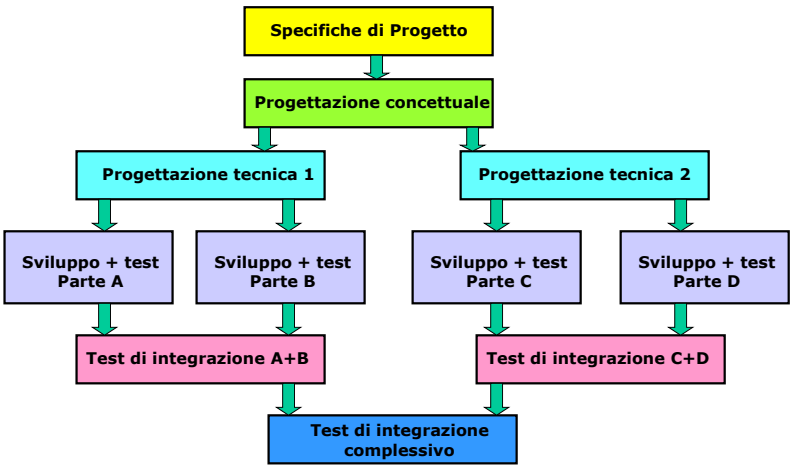
I modelli di Cicli di Vita del Software

Alcune considerazioni sul modello a V

- Il modello a V è nato per dare una rappresentazione, a partire dal classico modello a cascata, delle relazioni che intercorrono tra le varie componenti del ciclo di vita del software
- La visione del modello è orientata alla Verifica e Validazione di ogni fase, considerate attività essenziali per garantire la Qualità del prodotto SW che viene realizzato: ogni "strato" produce dei risultati che devono essere verificati in una specifica fase di Verifica/Validazione
- Un modello di questo tipo si adatta particolarmente bene, ad esempio, a gestire i progetti di sviluppo SW nei quali le responsabilità del ciclo di vita è ripartita tra Azienda committente (che si occupa del ramo ascendente del modello a V) e Azienda fornitrice (alla quale è affidato il ramo discendente), o comunque nei quali è stata predisposta una netta separazione di responsabilità tra *software factory* e *test factory*

I modelli di Cicli di Vita del Software

Il ciclo di vita parallelo



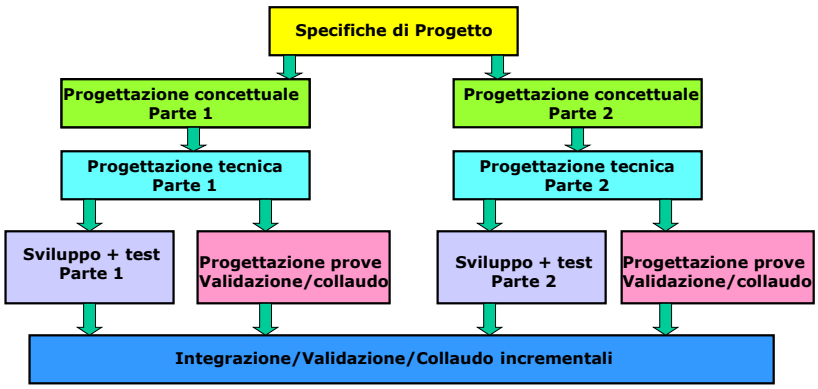
I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo di vita parallelo

- Nel ciclo di vita parallelo, una volta definiti in modo preciso i requisiti, si divide in modo logico l'attività di progettazione e sviluppo in più parti sufficientemente indipendenti, per poi procedere all'integrazione progressiva
- Questo modo di procedere è possibile a due condizioni: che sia possibile frazionare e parallelizzare abbastanza facilmente le attività e che si abbiano le Risorse Umane sufficienti ed adeguate per poter far lavorare i gruppi in parallelo
- Il ciclo di vita parallelo è particolarmente indicato nei casi in cui è necessario compattare i tempi di progetto, anche a rischio di maggiori costi di integrazione
- Per poterne utilizzare in modo completo i vantaggi, richiede comunque un controllo accurato del progetto

I modelli di Cicli di Vita del Software

Il ciclo di vita segmentato o incrementale



I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo di vita segmentato (segue)

- Si tratta di un approccio nel quale i requisiti complessivi del prodotto vengono suddivisi in più blocchi, ciascuno contenente un certo numero di requisiti: l'applicazione viene costruita in modo progressivo
- Ciascun blocco viene sviluppato indipendentemente dagli altri in tempi diversi (sottoprogetti con parallelismo sfalsato): i rilasci sono quindi multipli e ad ogni rilascio il sistema si arricchisce di funzionalità
- La suddivisione può, a seconda delle esigenze del Cliente, essere effettuata per area applicativa, per priorità di requisiti funzionali, per priorità di requisiti non funzionali, ecc.
- In qualche altro caso invece la priorità è di tipo architetturale (sviluppo prima A perché B ha bisogno di A per poter funzionare)
- L'integrazione avviene in modo incrementale (nuovi blocchi integrati con quelli precedentemente realizzati), man mano che i singoli blocchi sono pronti

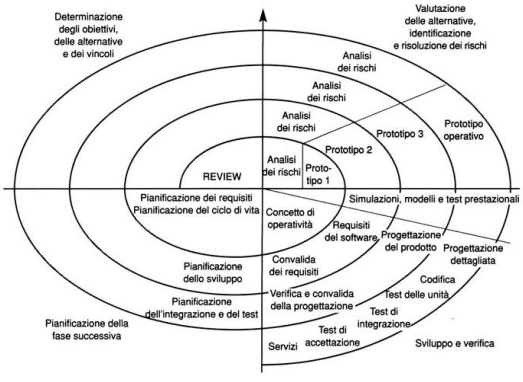
I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo di vita segmentato

- Quindi il modello incrementale richiede di scomporre il progetto in sottosistemi e di costruire il sistema per consolidamento successivo dei blocchi che vengono rilasciati
- Per esempio si può iniziare dalle parti più critiche o dalle più urgenti, dove serve un immediato *feedback* da parte del Cliente
- Così facendo il Cliente può iniziare ad utilizzare un *set* di funzionalità del sistema, mentre si stanno ancora sviluppando le altre
- Questo tipo di ciclo di vita è particolarmente adatto a gestire i progetti ICT particolarmente complessi (*effort* di alcuni anni/uomo), nei quali il rilascio complessivo può avvenire per fasi successive, riducendo così il *time to market*
- Richiede un intenso lavoro di verifiche/validazioni con gli utenti
- Il rischio principale è quello di dover ri-lavorare quanto è stato già realizzato, se non sono stati definiti con cura a priori gli *standard* di interfacciamento tra i vari blocchi

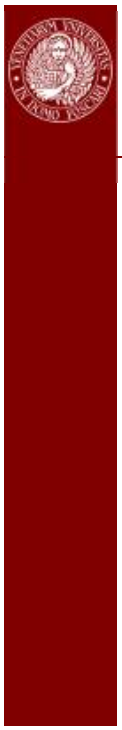
I modelli di Cicli di Vita del Software

Il modello di ciclo di vita di Boehm del 1988 (evolutivo, iterativo o a spirale)



Fasi principali del modello a spirale

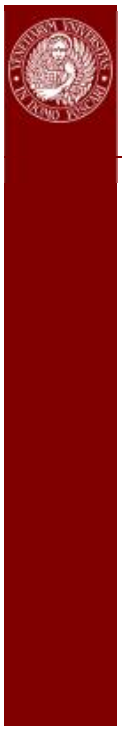
1) Determinare obiettivi e vincoli
2) Valutazione e mitigazione dei rischi e valutazione alternative
3) Progettazione e testing
4) Rilascio di una versione del prototipo
5) Pianificazione fase successiva



I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo di vita evolutivo (segue)

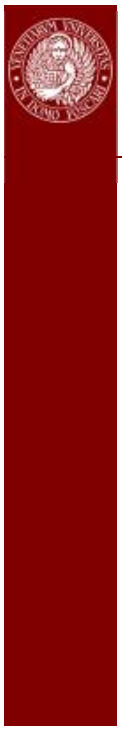
- Il ciclo di vita evolutivo è particolarmente adatto per realizzare quei prodotti per i quali è molto difficile definire in modo chiaro e completo i requisiti iniziali, per cui conviene procedere per sviluppi successivi sequenziali, avvicinandosi al risultato atteso per approssimazioni successive
- D'altra parte molto spesso lo stesso Project Manager non ha, fin dall'inizio, una definizione ben chiara ed esaustiva del dominio dell'applicazione, per non parlare di quei requisiti che si chiariscono solo in corso d'opera
- Comprende l'utilizzo di prototipi, che consentono una verifica immediata ed efficace della corretta interpretazione dei bisogni dell'utente, con affinamento progressivo dei requisiti
- L'abbinamento con l'approccio prototipale offre il vantaggio di poter sviluppare il software attraverso varie versioni del prodotto, via via più complete, da presentare all'utente, il cui coinvolgimento è più che mai fondamentale
- La sua natura iterativa fa sì che la stessa sequenza di fasi di lavoro del modello *waterfall* venga ripetuta più volte



I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo di vita evolutivo (segue)

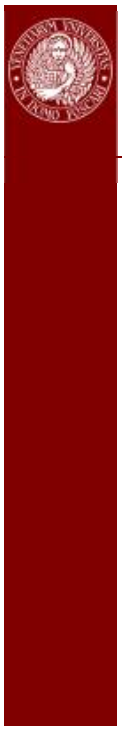
- Ad ogni iterazione vengono costruite nuove porzioni del sistema che vengono man mano integrate con quelle precedenti e verificate con il Cliente
- In letteratura, è sconsigliato superare le 8-10 iterazioni
- Questo approccio consente quindi di scoprire per tempo eventuali fraintendimenti con il Cliente, oltre che far emergere in modo spontaneo esigenze non espresse in precedenza o chiarire meglio funzionalità già analizzate
- Inoltre, per sua stessa natura, prevede la possibilità che i requisiti cambino in corso d'opera e/o che nascano nuovi requisiti da parte del committente
- Consente di rilasciare in tempi abbastanza rapidi una prima versione del sistema da realizzare, anche se ridotta, che da fiducia al Cliente sul buon esito del progetto
- Contrariamente all'approccio prototipale "usa e getta" (*throw-away prototyping*) in uso alcuni anni fa, il *prototyping* evolutivo *(evolutionary prototyping)* fa evolvere progressivamente il prodotto e consente di ridurre il *time to market*
- L'utilizzo di prototipi riduce il numero degli errori residui nel software consegnato al Cliente, in particolare quelli di analisi



I modelli di Cicli di Vita del Software

Alcune considerazioni sul ciclo di vita evolutivo

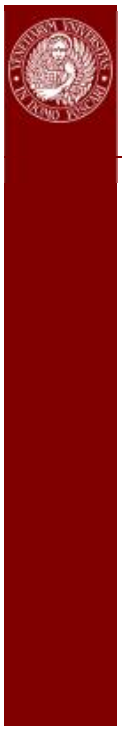
- Per lo stesso motivo, si riducono anche i rischi di progetto
- Si adatta particolarmente bene alla tecnologia di sviluppo ad oggetti, nella quale è abbastanza facile integrare i rilasci precedenti senza dover rimettere mano a componenti già realizzate
- Concentra l'attenzione sugli obiettivi e si presta alla gestione di progetti anche di notevole complessità
- Al contrario è assolutamente sconsigliato nel caso di sviluppo in ambienti tradizionali di tipo *legacy* perché porterebbe ad una lievitazione dei costi inaccettabile dovuto alla necessità di rifacimenti, anche parziali, delle parti già realizzate ad ogni giro del ciclo
- Richiede però un'eccellente gestione del progetto, se si vuole evitare il rischio di una spirale "senza fine": infatti il *work-plan* di un progetto iterativo evolve durante tutta la durata del progetto stesso e richiede quindi un controllo molto attento degli avanzamenti



I modelli di Cicli di Vita del Software

Metodologie di sviluppo Agili (segue)

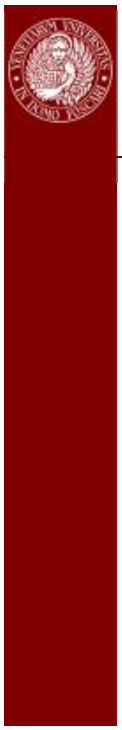
- Le metodologie di sviluppo "classiche" hanno l'obiettivo di organizzare il processo di produzione del SW per renderlo prevedibile, affidabile, efficiente e ripetitivo. Si tratta di metodologie di tipo predittivo
- Per ottenere questi risultati richiedono di applicare in modo rigoroso una serie di linee guida, di controlli, di adempimenti "burocratici", tra cui molta documentazione
- In contrapposizione alla rigorosità e alla pesantezza delle metodologie "classiche" sono nate le metodologie "agili":
  - ✓ Poca documentazione da produrre
  - ✓ Adatte a piccoli *team* di sviluppo
  - ✓ Obiettivo di massima semplicità del SW da produrre
  - ✓ Sviluppo del SW in modalità incrementale, a piccoli rilasci successivi, con rilasci sempre più frequenti
  - ✓ Il SW deve avere la caratteristica di essere facilmente modificabile
  - ✓ L'attività di *testing* va svolta in modo intensivo ed approfondito durante tutte le fasi del progetto in modo da garantire in ogni momento la stabilità del sistema



I modelli di Cicli di Vita del Software

Metodologie di sviluppo Agili (segue)

- Una delle prime metodologie di progettazione agile è stata lo SCRUM, nata all'inizio anni '90, con un nome ereditato dal termine del *rugby* che indica il pacchetto di mischia ed è una metafora del *team* di sviluppo che deve lavorare insieme in modo che tutti gli attori del progetto spingano nella stessa direzione, agendo come un'unica entità coordinata
- Il lavoro complessivo viene diviso in "*sprint*", che rappresentano blocchi di attività con obiettivi precisi e durata limitata (una trentina di giorni). Lo Scrum Team è composto da al massimo 5-9 persone
- Ogni iterazione o "*sprint*" copre l'analisi, lo sviluppo, il *testing* e il rilascio di una *release* dell'applicazione
- Le metodologie di sviluppo agili partono dalla considerazione che i requisiti utente cambiano in continuazione durante il progetto, per cui non ha molto senso svolgere una pesante attività di analisi, di progettazione e di documentazione prima di iniziare la codifica, in quanto molto di questo lavoro potrebbe risultare sprecato



I modelli di Cicli di Vita del Software

Metodologie di sviluppo Agili

- In alternativa si può mantenere la documentazione essenziale all'interno del codice sorgente stesso, oppure utilizzando dei *wiki*: così quando si modifica il codice a seguito delle variazioni di specifiche richieste dal Cliente, si modifica, in modo sincronizzato, anche la documentazione
- Una delle più note e diffuse metodologie agili è denominata eXtreme Programming (XP). Vedere i siti *www.extremeprogramming.org* e *www.xprogramming.com*

I modelli di Cicli di Vita del Software

www.agilemanifesto.org

Manifesto per lo Sviluppo Agile di Software

Stiamo scoprendo modi migliori di creare software,
sviluppandolo e aiutando gli altri a fare lo stesso.
Grazie a questa attività siamo arrivati a considerare importanti:

Gli individui e le interazioni più che i processi e gli strumenti

Quindi la risorsa più importante di un progetto è costituita dalla comunicazione e dalle relazioni tra gli attori in gioco

Il software funzionante più che la documentazione esaustiva

Bisogna rilasciare nuove versioni funzionanti del software ad intervalli frequenti, il codice deve essere semplice in modo da ridurre al minimo la documentazione

La collaborazione col cliente più che la negoziazione dei contratti

La collaborazione diretta con il cliente offre risultati migliori di un contratto

Rispondere al cambiamento più che seguire un piano

Il team di sviluppo dovrebbe essere autorizzato a suggerire modifiche al progetto

Ovvero, fermo restando il valore delle voci a destra,
consideriamo più importanti le voci a sinistra

I modelli di Cicli di Vita del Software

Alcune pratiche dell'Agile Programming (segue)

- ✓ Ricerca della soluzione più semplice
- ✓ Estrema semplicità del modo in cui il codice viene costruito
- ✓ Standard di codifica rispettati da tutti: chiunque deve poter comprendere il codice scritto da un altro membro del gruppo
- ✓ Progettazione dei test prima di iniziare la codifica
- ✓ Programmazione in coppia *(pair programming)*: chi scrive il codice è focalizzato sul modo migliore per realizzare un metodo, una classe, un algoritmo, mentre il collega è focalizzato sui controlli da effettuare, su cosa potrebbe non funzionare, su possibili strade alternative. Questa modalità consente, tra l'altro, di condividere le conoscenze. I ruoli devono essere tutti interscambiabili
- ✓ Frazionamento in piccoli sottoprogetti, con iterazioni brevi e frequenti
- ✓ Ripianificazione del progetto, con ridefinizione di priorità e requisiti ad ogni iterazione
- ✓ Non ci si preoccupa di pianificare attività lontane nel tempo, ma si pianifica in dettaglio solo l'iterazione successiva

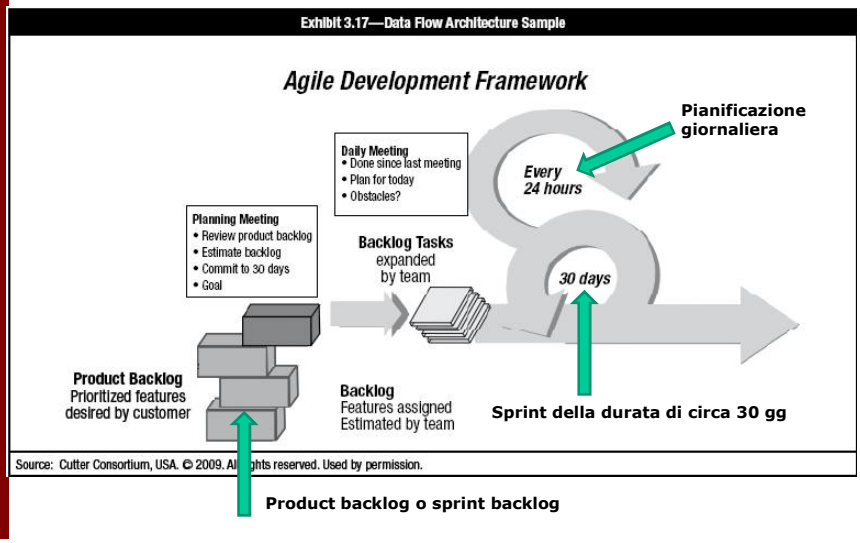
I modelli di Cicli di Vita del Software

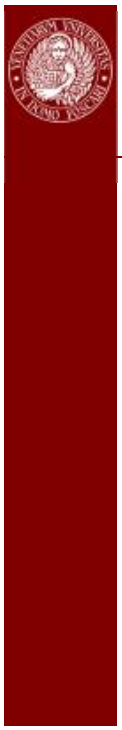
Alcune pratiche dell'Agile Programming

- ✓ Focus sulla conoscenza implicita delle persone anziché sulla esplicita documentazione del progetto
- ✓ Integrazione continua: il *testing* intensivo e continuo garantisce che ad ogni rilascio successivo il sistema rimanga comunque stabile
- ✓ *Refactoring* del codice, cioè sua ristrutturazione per contrastare il possibile degrado dovuto alle continue implementazioni
- ✓ Cliente sempre disponibile: un suo rappresentante deve essere sempre a disposizione del *project team* per rispondere alle domande
- ✓ Riunioni in piedi: veloci ed efficaci
- ✓ Proprietà collettiva del codice: chiunque, all'interno del *project team* può modificare qualunque parte del codice in qualsiasi momento
- ✓ In contrapposizione ai processi definiti e ripetibili dei cicli di vita classici, adattamento progressivo delle fasi di sviluppo in base alle risultanze dei momenti di verifica interna

I modelli di Cicli di Vita del Software

Il ciclo di sviluppo agile

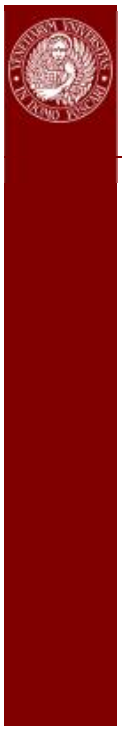




I modelli di Cicli di Vita del Software

Lo sviluppo a componenti (segue)

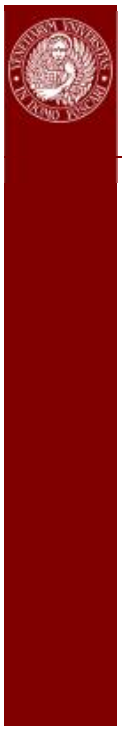
- Lo sviluppo a componenti (*Component Based Development*) si applica a qualsiasi modello di ciclo di vita del SW e si basa sul riuso di parti già realizzate da altri con l'obiettivo di ridurre tempi e costi di realizzazione
- Anziché costruire ex-novo, si assemblano pezzi già pronti
- Per componente si intende una parte di un'applicazione (quella che il *Configuration Management* chiama *Configuration Item*), che ha una funzione chiaramente identificabile
- Esempi di componenti sono: porzioni di codice sorgente (programmi, *routines*, classi Java, script Unix,…), interfacce utente, prototipi, documentazione, dati, architetture, specifiche tecniche, diagrammi, casi di test, stime dei costi, piani di progetto, ecc.
- Naturalmente il riuso ha senso solo se risulta meno costoso che sviluppare ex-novo: quindi bisogna tener conto degli eventuali costi di adattamento e/o di personalizzazione



I modelli di Cicli di Vita del Software

Lo sviluppo a componenti (segue)

- Il *Component Based Development* è legato alla tecnologia ad oggetti e consiste nel costruire applicazioni usando pacchetti eseguibili che rendono disponibili i loro servizi tramite interfacce definite, indipendentemente dal linguaggio utilizzato per scriverli e dal sistema operativo in cui si eseguono
- Lo sviluppo per componenti consente di:
  - Ridurre il tempo di sviluppo
  - Migliorare la Qualità
  - Focalizzarsi sulle funzionalità di business, slegandosi dalla programmazione a basso livello
  - Dare maggiore enfasi alla modularità
  - Rendere più semplice il riutilizzo del software
  - Ridurre i costi complessivi di sviluppo
- Il software riutilizzato ha inoltre il vantaggio di avere una minore densità di errori, in quanto è già passato per più fasi di *testing* e più utilizzi



I modelli di Cicli di Vita del Software

Lo sviluppo a componenti

- La politica del riuso, per essere veramente efficace, ha un costo, in quanto vanno sommati i costi aggiuntivi necessari per produrre componenti riusabili, i costi di gestione del catalogo del riuso, i costi dei *tools* necessari per supportare il processo di riuso
- Lo sviluppo a componenti rende talvolta più complessa la definizione della proprietà del software, in particolare quando vengono utilizzati componenti protetti da *copyright*
- Uno dei fattori inibitori del riuso molto frequente è quello legato alla mentalità dello sviluppatore, che ritiene più rapido e più divertente scrivere ex-novo un componente, piuttosto che "perdere tempo" a cercarlo, capire cosa fa, se è adatto o meno al suo caso, adattarlo, ecc.
- Quando il riuso, lo sviluppo a componenti, l'utilizzo di prototipi evolutivi, si abbinano ad un lavoro in parallelo su *team* piccoli e ben addestrati, rigorosamente fasati tra loro, che utilizzano strumenti integrati di modellazione e prototipazione, con l'obiettivo di ridurre drasticamente i tempi di rilascio, si parla di *Rapid Application Development* (RAD)