Luca Cabibbo

Analisi e Progettazione del Software

Sviluppo iterativo ed evolutivo

Capitolo 2

marzo 2017

Lo sviluppo iterativo dovrebbe essere utilizzato solo per i progetti che si desidera vadano a buon fine.

Martin Fowler

2.2 Processi per lo sviluppo del software

Un **processo** (o **metodo**) **per lo sviluppo del software** – o **processo software** – definisce un approccio disciplinato per la costruzione, il rilascio e la manutenzione del software

- definisce **chi** fa **che cosa**, **quando** e **come** per raggiungere un certo obiettivo
- "che cosa" sono le *attività*, "chi" sono i *ruoli*, "come" sono le *metodologie*, "quando" riguarda l'*organizzazione temporale* delle attività

Esistono numerosi processi software

- il processo a cascata, UP, Scrum, XP, ...
- tutti organizzati attorno ad alcune attività fondamentali comuni

Attività nei processi software

Attività comuni nei processi software

- *specifica* (o *analisi*) dei *requisiti*
- *analisi*
- *progettazione*
- *implementazione*
- *validazione e verifica*
- *rilascio/installazione*
- *manutenzione/evoluzione*
- *gestione del progetto*

Differenziazione tra processi software

Perché esistono diversi processi software? In cosa si differenziano?

- nel “che cosa”?
- nel “chi”?
- nel “come”?
- nel “quando”?

- i processi si differenziano soprattutto nel “quando”
  - nell’ordine delle attività
  - nei criteri di transizione da un’attività alla successiva

2.1 Che cos'è UP

Unified Process (**UP**) è un processo *iterativo* per lo sviluppo di software OO

- l'OOA/D si applica al meglio proprio nei processi **iterativi** – e, se possibile, **agili**
- UP è aperto e flessibile – incoraggia l'adozione di pratiche da altri metodi iterativi
  - come Scrum, Extreme Programming (XP), Agile Modeling
  - ad es., TDD, refactoring, integrazione continua, …
  - dunque, UP consente di applicare le best practices più comunemente accettate nello sviluppo del software
- in ogni caso, le idee fondamentali del corso (sviluppo iterativo, progettazione guidata dalle responsabilità, casi d'uso, storie utente, OOA/D, pattern, ...) sono indipendenti da ogni particolare processo – e possono essere applicate anche nel contesto di altri processi

* Disclaimer – Processi vs. Persone

I principi, le tecnologie e i processi sono importanti – ma sono più importanti le persone che li mettono in pratica [Cockburn]

- i processi e le tecnologie hanno un impatto secondario nei risultati di un progetto
- le persone – con le loro individualità, sentimenti e qualità – sono molto più importanti

Le persone sono più importanti dei processi [Booch]

- persone buone che adottano un buon processo lavoreranno meglio, in ogni caso, di buone persone senza processo

Per realizzare qualcosa di valore, le persone devono mettere "anima e cuore" in tutto quello che fanno

- i "processi" sono solo per migliorare il modo in cui le persone lavorano insieme

2.3 Il processo "a cascata"

Il **processo a cascata** è un processo software "classico" – definito negli anni '60/'70 – ma purtroppo ancora diffuso

- attività svolte in modo sequenziale
  - pianificazione – con stime dettagliate per tutte le attività
  - analisi dei requisiti del software
  - progettazione
  - generazione del codice
  - collaudo
- inoltre
  - ciascuna attività produce documenti dettagliati – soggetti a revisione e ad approvazione formale
  - in accordo con il piano iniziale, il lavoro procede da un'attività alla successiva solo con l'approvazione dei documenti dell'attività precedente
  - spesso le diverse attività sono svolte da team differenti

Il processo "a cascata"

Il **processo a cascata** è un processo software "classico" – definito negli anni '60/'70 – ma purtroppo ancora diffuso

- attività svolte in modo sequenziale
  - pianificazione – con stime dettagliate per tutte le attività
  - analisi dei requisiti del sof
  - progettazione
  - gen
  - co
- i
  - spesso le diverse attivit da team differenti

Sequenziale vuole dire:

- ✓ prima faccio la pianificazione (mesi...)
- ✓ poi faccio tutta l'analisi (altri mesi)
- ✓ poi faccio tutta la progettazione (altri mesi)
- ✓ poi scrivo il codice (altri mesi)
- ✓ poi inizio a fare il collaudo (altri mesi)
- ✓ a questo punto ho in mano qualcosa di funzionante (forse)

Il processo "a cascata"

Il **processo a cascata** è un processo software "classico" – definito negli anni '60/'70 – ma purtroppo ancora diffuso

Che origini ha?

Può funzionare per lo sviluppo del software?

Potrebbe non funzionare?
In quali casi? Perché?
Si può fare di meglio?

a

tività

Il processo a cascata è poco efficace

Il processo a cascata è spesso poco efficace nello sviluppo del software

- è associato a un'elevata percentuale di fallimenti, a bassa produttività e a percentuali di difetti alte – perché?

Un motivo importante

- nello sviluppo a cascata, tutte le "buone idee" – sui requisiti, sull'architettura, sul progetto – dovrebbero venire all'inizio del progetto – in modo da poter essere incorporate nel "piano"
  - ma sappiamo che non è sempre così – le buone idee possono venire in qualunque momento
- ma il processo a cascata non consente l'introduzione di buone idee "ritardatarie"
  - nel processo a cascata, una buona idea ritardataria non è un'opportunità – anzi, è una minaccia!

Il processo a cascata è poco efficace

Un altro motivo

- il processo a cascata non consente una gestione efficace dei rischi
  - poiché affronta i rischi *tardi* nel ciclo di vita
  - inoltre, è molto costoso gestire errori introdotti nelle prime fasi – errori nei requisiti, nell'analisi, nel progetto o anche nell'implementazione – che spesso vengono rilevati solo durante il collaudo

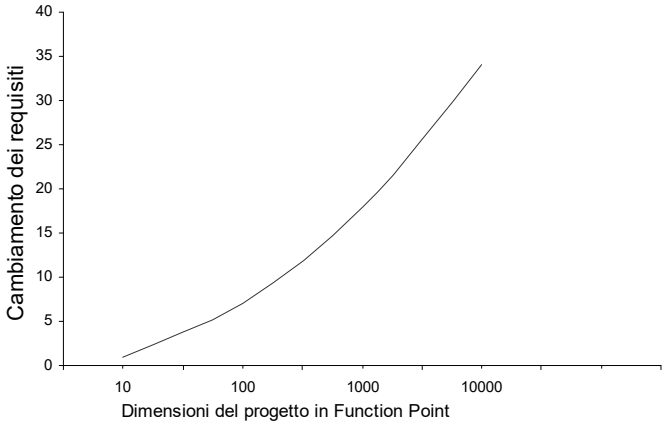
Ad esempio, un'ipotesi fondamentale del processo a cascata è che sia possibile definire correttamente e "congelare" i requisiti prima di procedere con le fasi successive

- ma questa ipotesi è realistica?

Il processo a cascata è poco efficace

Studi empirici hanno dimostrato che

- in un progetto software medio i requisiti cambiano del 25%
- i cambiamenti sono del 35%-50% per progetti grandi e/o innovativi

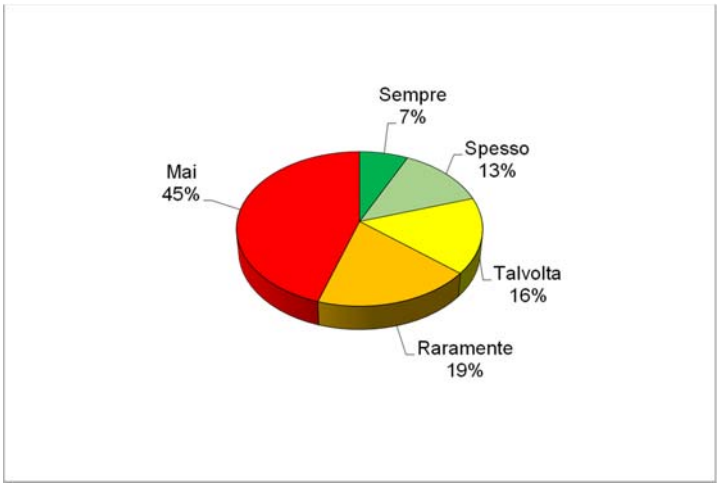


- una causa comune del fallimento dei progetti software è la mancata gestione del cambiamento

Requisiti che cambiano

Un'altra indagine

- se viene fatta un'analisi "a cascata" dei requisiti, quante caratteristiche identificate inizialmente saranno effettivamente utili nel prodotto finale?



Il processo a cascata è poco efficace

Il **processo a cascata** è un processo software "classico" – definito negli anni '60/'70 – ma purtroppo ancora diffuso

- attività svolte in modo sequenziale
  - pianificazione – con stime dettagliate per tutte le attività
  - analisi dei requisiti del software

Processo a cascata:
si può fare di meglio?

- i
  - ciascuna attività produce documenti dettagliati – soggetti a revisione e ad approvazione formale
  - in accordo con il piano iniziale, il lavoro procede da un'attività alla successiva solo con l'approvazione dei documenti dell'attività precedente
  - spesso le diverse attività sono svolte da team differenti

A P S 2.4 Lo sviluppo evolutivo ed iterativo

L'idea fondamentale dello **sviluppo evolutivo** – realizzare un'implementazione iniziale del sistema, esporla agli utenti e raffinarla attraverso diverse versioni, finché non si ottiene un sistema adeguato

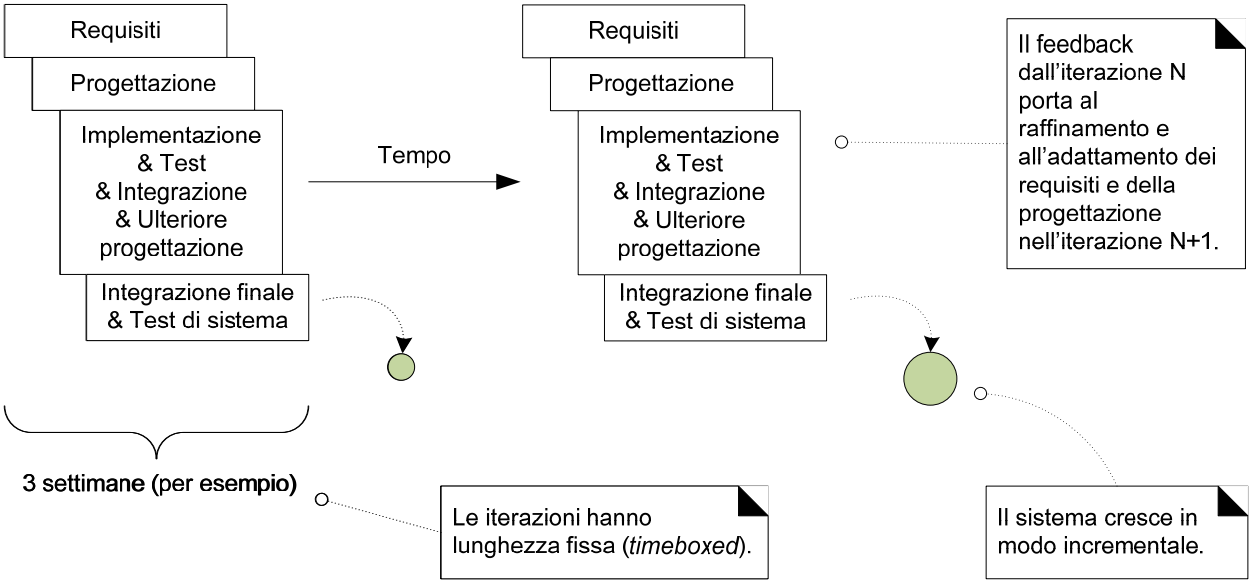
- l'evoluzione è guidata da esperienze pratiche dell'utilizzo del sistema
  - il sistema viene esposto ai suoi utenti, ed evolve sulla base del feedback ottenuto dal suo utilizzo
  - *sì, è più o meno quello che volevo – ma in effetti volevo qualcosa di un po' diverso…*
  - per ottenere un feedback veloce ed efficace, le diverse attività dello sviluppo del software sono intrecciate anziché separate

A P S Lo sviluppo iterativo

Nello **sviluppo iterativo** lo sviluppo del software è organizzato in una serie di mini-progetti brevi, di lunghezza fissa (ad es., 2-4 settimane) chiamati **iterazioni**

- ogni iterazione comprende lo svolgimento di attività di analisi dei requisiti, analisi, progettazione, implementazione, verifica, …
- il risultato di ciascuna iterazione è un ***sistema software eseguibile***, integrato e testato, anche se ***parziale***
- il sistema cresce in modo **incrementale** da un'iterazione alla successiva – adattandosi ai requisiti, in modo **evolutivo**, sulla base del feedback delle iterazioni precedenti
- il risultato di ciascuna iterazione è normalmente un sistema incompleto – che converge verso un sistema completo dopo varie iterazioni

Sviluppo iterativo e incrementale



Sviluppo iterativo, incrementale ed evolutivo

Lo sviluppo iterativo è **incrementale**

- il sistema cresce in modo incrementale, iterazione per iterazione

Lo sviluppo iterativo è **evolutivo**

- le specifiche e il progetto evolvono, in base a feedback ed adattamento

Sviluppo iterativo

L'idea di fondo dello sviluppo iterativo

- in ciascuna iterazione, considerare solo un piccolo insieme di requisiti
- anticipare nel tempo parte di alcune attività
- rimandare nel tempo parte di altre attività
- cercando di
  - massimizzare i benefici
  - minimizzare i rischi

Cambiamento e adattamento

Nello sviluppo del software, i cambiamenti sono inevitabili

- lo sviluppo iterativo accetta i cambiamenti
  - lo sviluppo iterativo "abbraccia" il cambiamento e l'adattamento, scegliendoli come guide essenziali
  - ma questo non vuol dire lavorare in modo caotico ("feature creep")
- come viene gestito il cambiamento nello sviluppo iterativo?

Cambiamento e adattamento

Nello sviluppo iterativo

- ogni iterazione opera su un piccolo insieme di requisiti
- il lavoro procede mediante una serie di iterazioni di costruzione-feedback-adattamento
- nel corso delle iterazioni, grazie al feedback e all'adattamento, la deviazione tra sistema realizzato e sistema desiderato (in termini di requisiti e di progetto finali) tende a diminuire nel tempo
  - il sistema realizzato converge verso quello desiderato
- i cambiamenti più significativi dovrebbero essere sollecitati soprattutto nelle iterazioni iniziali

Feedback e adattamento

Ingredienti chiave per il successo dello sviluppo iterativo sono feedback e adattamento

- feedback proveniente dalle attività di sviluppo – in relazione ai requisiti e alla loro comprensione da parte del team di sviluppo
- feedback proveniente dai test
- feedback proveniente dagli sviluppatori che raffinano i modelli e il progetto
- feedback per raffinare le stime dei tempi e dei costi
- feedback proveniente dai clienti e dal mercato – sulla priorità da assegnare alle caratteristiche da sviluppare
- feedback sull'applicazione del processo software

Vantaggi dello sviluppo iterativo

Vantaggi dello sviluppo iterativo – dimostrati da ricerche sullo sviluppo iterativo e incrementale

- minor probabilità di fallimento del progetto, miglior produttività, percentuali più basse di difetti
- riduzione precoce anziché tardiva dei rischi maggiori
- visibilità del progresso
- feedback precoce, impegno degli utenti, adattamento
- gestione della complessità
- miglioramento continuo del processo stesso

Attenzione: ci sono anche dei rischi...
… ma conoscendoli è possibile affrontarli in modo opportuno

Timeboxing

Una buona pratica dello sviluppo iterativo è il **timeboxing** – ogni iterazione deve avere una durata *prefissata* (di circa 1-6 settimane, a seconda del processo specifico)

- iterazioni *brevi* consentono un adeguato feedback e adattamento ai cambiamenti
  - iterazioni *troppo brevi*, invece, non consentono di svolgere lavoro sufficiente in un'iterazione
  - iterazioni *troppo lunghe*, invece, non consentono di ottenere tutti i benefici dello sviluppo iterativo, aumentando così i rischi
- la durata di un'iterazione, una volta fissata, non può cambiare – le iterazioni sono **timeboxed**
  - ciascuna iterazione deve produrre un sistema eseguibile (anche se parziale)
  - è meglio ridurre i requisiti di un'iterazione, piuttosto che non rispettarne la scadenza

Progetti iterativi vs. pensare a cascata

Un rischio comune

- dire di adottare un processo iterativo… ma in realtà pensare a cascata
  - in questa iterazione scrivo i requisiti
  - in questa iterazione faccio il progetto
  - solo dopo scrivo il codice
  - il test lo faccio in un'altra iterazione
- purtroppo visto anche in libri, tirocini e tesi ☹

In questo caso

- il pensiero a cascata ha inficiato il progetto
- non si tratta di un progetto iterativo o UP sano

Sviluppo iterativo e qualità del codice

Se si adotta un processo di sviluppo iterativo, è bene che il software possegga alcune qualità

- in particolare, la struttura del software deve essere **flessibile** – in modo tale che l'impatto dei cambiamenti sul sistema sia basso
  - a tal fine, il codice – ma anche il progetto sottostante – deve essere facilmente **modificabile**
  - inoltre, come prerequisito, il codice – ma anche il progetto sottostante – deve essere facilmente **comprensibile**

Attenzione, queste osservazioni – apparentemente banali – hanno un impatto fondamentale sul modo di fare analisi e progettazione del software

Sviluppo iterativo e qualità del codice

Se si adotta un processo di sviluppo iterativo, è bene che il software possegga alcune qualità

- in particolare, la struttura del software deve essere **flessibile** – in modo tale che l'impatto dei cambiamenti sul sistema sia basso
  - a tal fine, il codice – ma anche il progetto sottostante – deve essere facilmente **modificabile**
  - inoltre, come prerequisito, il codice – ma anche il progetto sottostante – deve essere facilmente **comprensibile**

Come sostenere queste qualità?

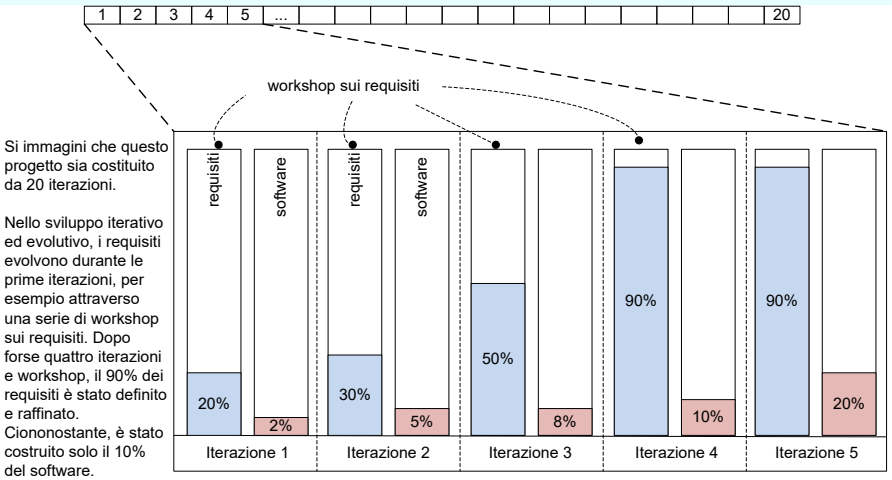
- applicando principi di progettazione opportuni – salto rappresentazionale basso, modularità, separazione degli interessi, ...
- utilizzando strumenti (metodologici) opportuni – tecnologie OO, refactoring, sviluppo guidato dai test, tracciatura dei requisiti e gestione delle modifiche, pianificazione iterativa, ...

2.5 OOA/D iterativa ed evolutiva

1	2	3	4	5	...													20

Si immagini che questo progetto sia costituito da 20 iterazioni.

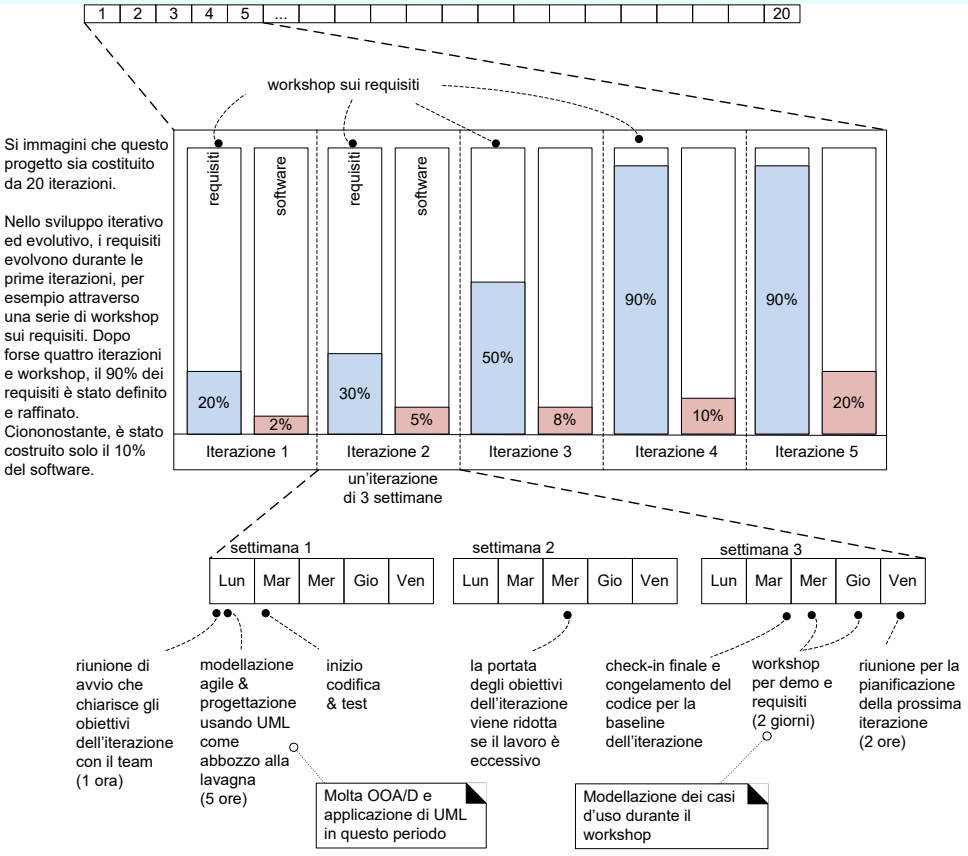
OOA/D iterativa ed evolutiva



Si immagini che questo progetto sia costituito da 20 iterazioni.

Nello sviluppo iterativo ed evolutivo, i requisiti evolvono durante le prime iterazioni, per esempio attraverso una serie di workshop sui requisiti. Dopo forse quattro iterazioni e workshop, il 90% dei requisiti è stato definito e raffinato. Ciononostante, è stato costruito solo il 10% del software.

OOA/D iterativa ed evolutiva



Si immagini che questo progetto sia costituito da 20 iterazioni.

Nello sviluppo iterativo ed evolutivo, i requisiti evolvono durante le prime iterazioni, per esempio attraverso una serie di workshop sui requisiti. Dopo forse quattro iterazioni e workshop, il 90% dei requisiti è stato definito e raffinato. Ciononostante, è stato costruito solo il 10% del software.

A P S * Parentesi: Principio di Pareto

Principio di Pareto

- la maggior parte degli effetti è dovuta a un numero ristretto di cause

Una legge empirica – detta anche "legge 80/20"

- la maggior parte degli effetti (ad es., l'80%) è dovuta a un numero ristretto di cause (ad es., il 20%)
- i valori 80% e 20% sono solo indicativi
- applicazioni – economia, informatica, qualità, ...

Quale interpretazione della figura precedente?

- le prime iterazioni si possono occupare solo del 20% dei requisiti o del sistema
- ma non si devono occupare di un 20% a caso – ma proprio del 20% dei requisiti o del sistema che contribuiscono all'80% del valore complessivo!

A P S 2.6 Pianificazione iterativa

Una pratica fondamentale dello sviluppo iterativo è la **pianificazione iterativa**

- che cosa fare nella prossima iterazione?

UP incoraggia la pianificazione iterativa **guidata dal rischio** e **guidata dal cliente**

- obiettivo delle prime iterazioni
  - identificare e affrontare i rischi principali
  - implementare caratteristiche visibili a cui il cliente è particolarmente interessato
- compatibile con la pratica dello sviluppo **centrato sull'architettura**
  - le prime iterazioni si concentrano sulla costruzione, il test e la stabilizzazione del nucleo dell'architettura

Iterazioni e requisiti bloccati

Durante ciascuna iterazione, i requisiti su cui operare vengono prefissati (stabiliti all'inizio dell'iterazione) e bloccati (non possono cambiare durante l'iterazione)

- il team di sviluppo, nell'ambito di ciascuna iterazione, può lavorare al suo meglio
  - poiché i requisiti sono bloccati, il team non può essere interrotto o disturbato dai committenti durante un'iterazione – con un impatto positivo sulla tranquillità e produttività del team e sulla possibilità di raggiungere gli obiettivi dell'iterazione
- d'altra parte, i committenti possono interagire con il team al termine di ciascuna iterazione (che sono brevi)
  - possono vedere il progresso effettivo dello sviluppo e fornire un feedback per guidare lo sviluppo successivo
  - possono fare nuove richieste da prendere in considerazione nelle iterazioni successive

* Pianificazione iterativa e backlog

Il **backlog** ("lavoro arretrato") è uno strumento per l'organizzazione del lavoro che deve essere ancora svolto nello sviluppo iterativo

- il backlog è un elenco di requisiti che devono essere ancora implementati e di problemi che devono essere ancora risolti
- la gestione del backlog è iterativa e guida la pianificazione e l'evoluzione del processo di sviluppo
- prima di iniziare una nuova iterazione
  - le voci del backlog vengono aggiornate – facendo cancellazioni, aggiunte o modifiche – anche sulla base del feedback dell'iterazione che si sta concludendo
  - a ciascuna voce viene assegnata una priorità – sulla base del valore di business e dei rischi associati a quella voce
  - per l'iterazione che sta per iniziare viene scelto come compito l'implementazione di voci del "lavoro arretrato" tra quelle che hanno priorità maggiore

A P S 2.7 Altre pratiche fondamentali di UP

UP promuove diverse discipline fondamentali (best practices)

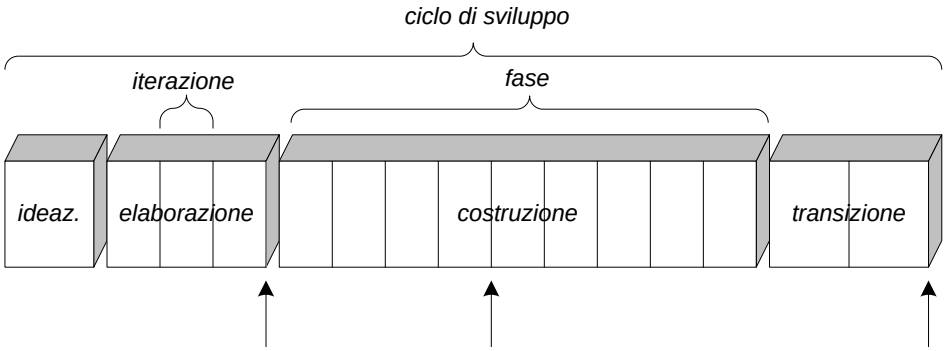
- sviluppare software in modo iterativo, evolutivo ed adattivo
- sviluppo guidato dal rischio
- sviluppare il cuore dell'architettura nelle prime iterazioni
- coinvolgimento continuo degli utenti
- verifica continua delle qualità – testare presto, spesso, e in modo realistico
- gestire attentamente i requisiti
- applicare i casi d'uso – se appropriato
- uso delle tecnologie a oggetti (OOA/OOD/OOP)
- modellare in modo visuale
- gestire le richieste di cambiamento e le configurazioni
- sviluppo basato su componenti

A P S 2.8 Fasi di UP

Un progetto UP organizza il lavoro e le iterazioni in quattro **fasi** principali

- **ideazione** (**inception**) – avvia il progetto
  - visione approssimativa del sistema, stime vaghe dei costi
- **elaborazione** (**elaboration**) – realizza il nucleo dell'architettura
  - visione raffinata del sistema, implementazione iterativa del nucleo dell'architettura, risoluzione dei rischi maggiori, identificazione della maggior parte dei requisiti, stime realistiche dei costi
- **costruzione** (**construction**) – realizza le capacità operative iniziali
  - implementazione iterativa, risoluzione dei rischi minori
- **transizione** (**transition**) – completa il prodotto
  - verifiche finali (beta test) e rilascio agli utenti

Fasi di un progetto UP



milestone

La fine di un'iterazione in cui si verifica una decisione o una valutazione significativa.

release

Un sottoinsieme stabile ed eseguibile del prodotto finale. La fine di ogni iterazione produce una release minore.

incremento

La differenza (delta) tra le release di due iterazioni successive.

release produzione finale

A questo punto il sistema viene rilasciato e consegnato ai clienti.

2.9 Discipline di UP

In UP, le attività lavorative (compiti) sono organizzate in discipline

- **disciplina** – l'insieme delle attività, e dei relativi elaborati, relative a una determinata area
- **elaborato** (*artifact*) – termine generico che indica un prodotto di lavoro

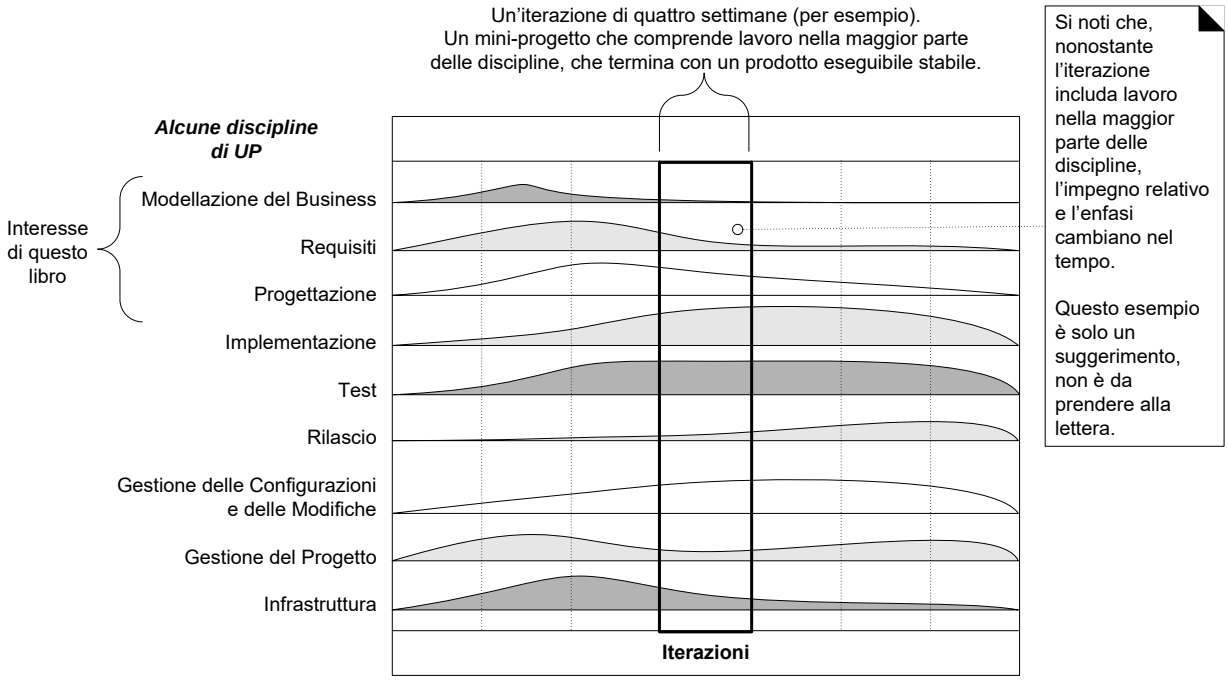
Tra le discipline di UP, tre sono di interesse per questo corso

- **requisiti**
- **modellazione del business** (business modeling)
- **progettazione**

Alcune ulteriori discipline di UP

- *implementazione*
- *test*
- *gestione del progetto*

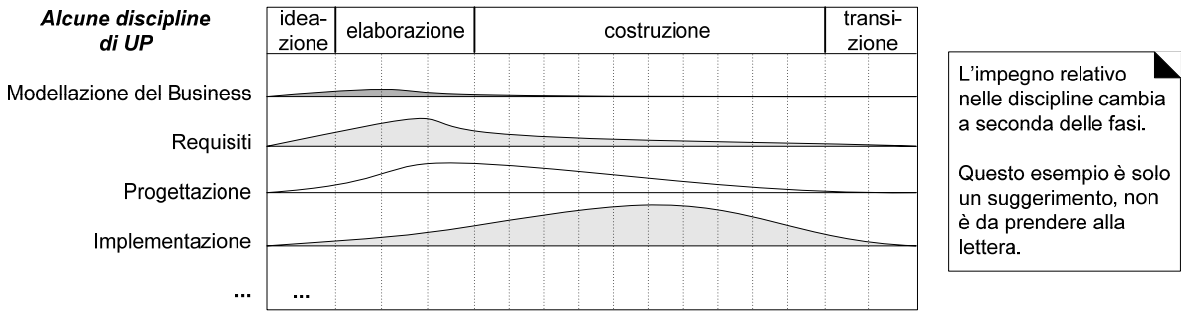
Discipline di UP e sviluppo iterativo



Discipline di UP e sviluppo iterativo

Nello sviluppo iterativo

- ciascuna iterazione coinvolge molte o tutte le discipline
- lo sforzo richiesto da ciascuna disciplina cambia durante lo svolgimento delle varie fasi e iterazioni

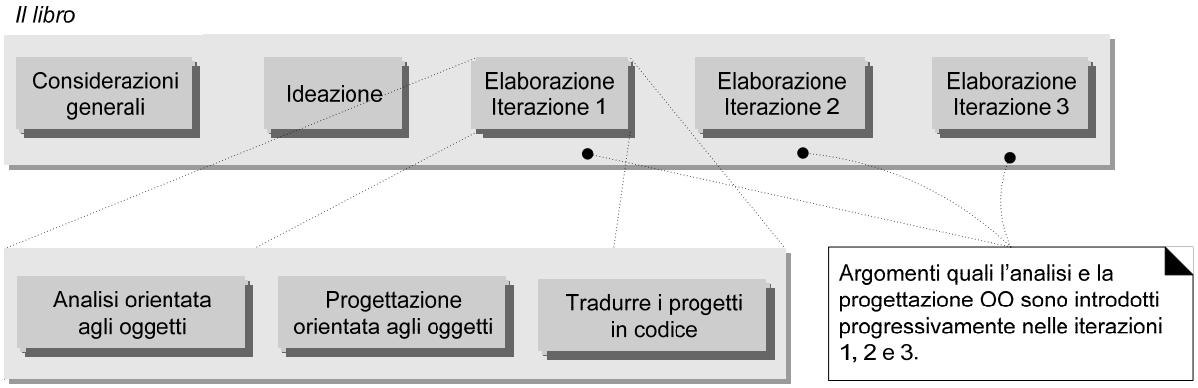


Struttura del corso, fasi e discipline

Questo corso enfatizza

- soprattutto la fase di elaborazione
- alcune attività ed elaborati delle discipline di modellazione del business, requisiti, progettazione

che sono la fase e le discipline in cui vengono principalmente applicate l'analisi dei requisiti, l'OOA, l'OOD, i pattern, UML



2.10 Personalizzare UP

Alcuni principi e pratiche di UP sono indispensabili

- ad eccezione del codice, tutte le attività e gli elaborati sono opzionali
- ciascun progetto va basato sugli elaborati e le attività che sono effettivamente di valore per quel particolare progetto

La scelta degli elaborati di UP per un progetto viene scritta in un breve documento

- lo **Scenario di sviluppo** (development case) – disciplina di infrastruttura (environment)

Scenario di sviluppo per lo studio di caso

Disciplina	Pratiche	Elaborato	Ideaz. (I1)	Elab. (E1..En)	Costr. (C1..Cn)	Trans. (T1..T2)
Modellazione del business	agile modeling workshop requisiti	Modello di dominio (Domain Model)		s		
Requisiti	workshop requisiti vision box dot voting	Modello dei casi d'uso (Use-Case Model)	s	r		
		Visione (Vision)	s	r		
		Specifiche supplementari (Supplementary Specification)	s	r		
		Glossario (Glossary)	s	r		
Progettazione	agile modeling test-driven dev.	Modello di progetto (Design Model)		s	r	
		Documento dell'architettura software (SW Architecture Document, SAD)		s		
		Modello dei dati (Data Model)		s	r	
Implementazione	test-driven dev. pair programming integrazione continua standard di codifica	Modello di implementazione (Implementation Model)		s	r	r
Gestione del progetto	agile PM daily meeting	Piano di sviluppo software (SW Development Plan)	s	r	r	r
...						

- s = start (inizia) r = refine (raffina)

2.11 Non hai capito lo sviluppo iterativo o UP se...

- pensi che ideazione=requisiti, elaborazione=progettazione e costruzione=implementazione
- pensi che lo scopo dell'elaborazione è di definire modelli, che vengono implementati (in codice) nella fase di costruzione
- provi a definire la maggior parte dei requisiti prima di iniziare la progettazione e l'implementazione
- provi a fare la maggior parte della progettazione prima di iniziare l'implementazione
- provi a definire e scegliere una architettura prima di iniziare la programmazione e il test in modo iterativo
- viene speso molto tempo per i requisiti e la progettazione prima di iniziare la programmazione
- credi che la lunghezza giusta per le iterazioni sia tre mesi – non tre settimane
- pensi che la creazione di diagrammi UML e le attività di progettazione devono definire il progetto in dettaglio, e che la programmazione è semplicemente traduzione di questi diagrammi
- pensi che adottare UP significhi svolgere quante più attività e produrre quanti più elaborati possibili
- provi a pianificare il progetto in dettaglio dall'inizio alla fine, in modo speculativo
- vuoi piani e stime dei costi credibili prima che sia conclusa la fase di elaborazione