Unità di apprendimento 5

Gestione di progetti informatici

Unità di apprendimento 5
Lezione 8

Modelli classici di sviluppo
di sistemi informatici

In questa lezione impareremo...

- i modelli di sviluppo del progetto software
- a scegliere le metodologie e le tecniche
- adeguate alle diverse situazioni

Generalità

Possiamo sintetizzare il ciclo del processo di produzione del software in tre fasi (o macrofasi):

- Progetto: costituito dalle fasi preliminari che portano alla stesura del contratto;
- Sviluppo: fasi realizzative del software fino alla relativa consegna;
- Manutenzione: interventi di modifica post-consegna.

Ciclo di vita del software

L'ingegneria del software ha specificato le seguenti fasi che caratterizzano il ciclo di vita del software.

- **Pianificazione del sistema**: in questa fase viene effettuata una definizione generale delle specifiche per **capire COSA deve essere realizzato**.
- **Analisi**: **individuazione dettagliata di COSA dovrà essere realizzato** senza alcun riferimento al come. L'output di questa fase è il documento di Specifica dei Requisiti Software (SRS).

Ciclo di vita del software

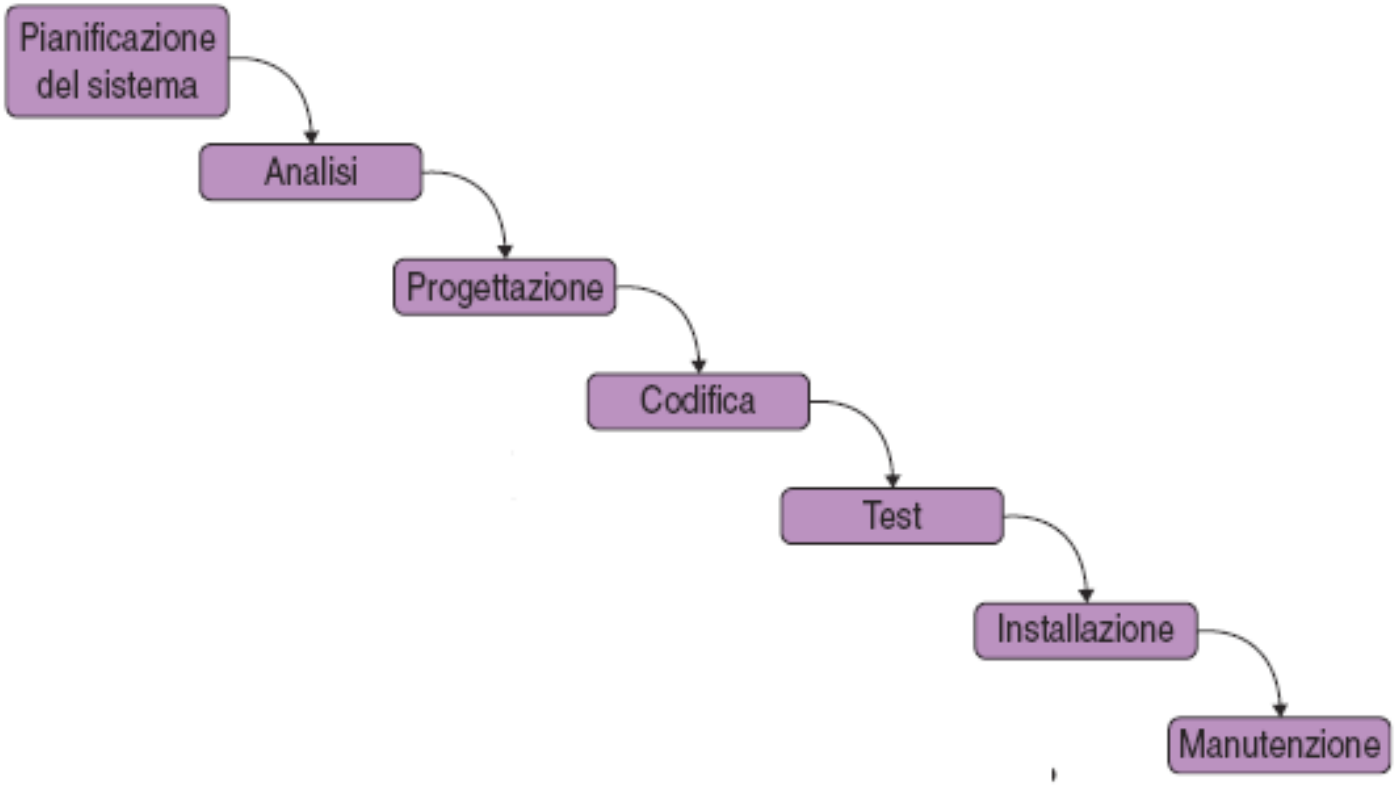
- **Progettazione**: definizione delle strategie per arrivare agli obiettivi fissati (architettura del sistema, individuazione delle strutture dati, delle singole classi o funzioni, ecc.), iniziando ad individuare **COME deve essere realizzato** il prodotto software.
- **Realizzazione**: codifica e compilazione dei moduli.
- **Test e debug**: verifica della correttezza del prodotto utilizzando un campione di dati di prova e, conseguente debug nel caso fossero rilevati dei *bug*.
- **Installazione, verifica e collaudo**: installazione presso il cliente, verifica del rispetto delle specifiche individuate nell'analisi e collaudo con dati reali.
- **Manutenzione**: fase permanente di supporto del sistema. Prevede la correzione di errori ancora presenti, di adattare l'applicazione ad altri sistemi e di migliorarla aggiungendo nuove funzionalità o ottimizzare quelle esistenti.

Modelli di sviluppo

I modelli di sviluppo definiscono la modalità con cui si succedono le diverse fasi del progetto software. Si dividono i tre gruppi distinti:

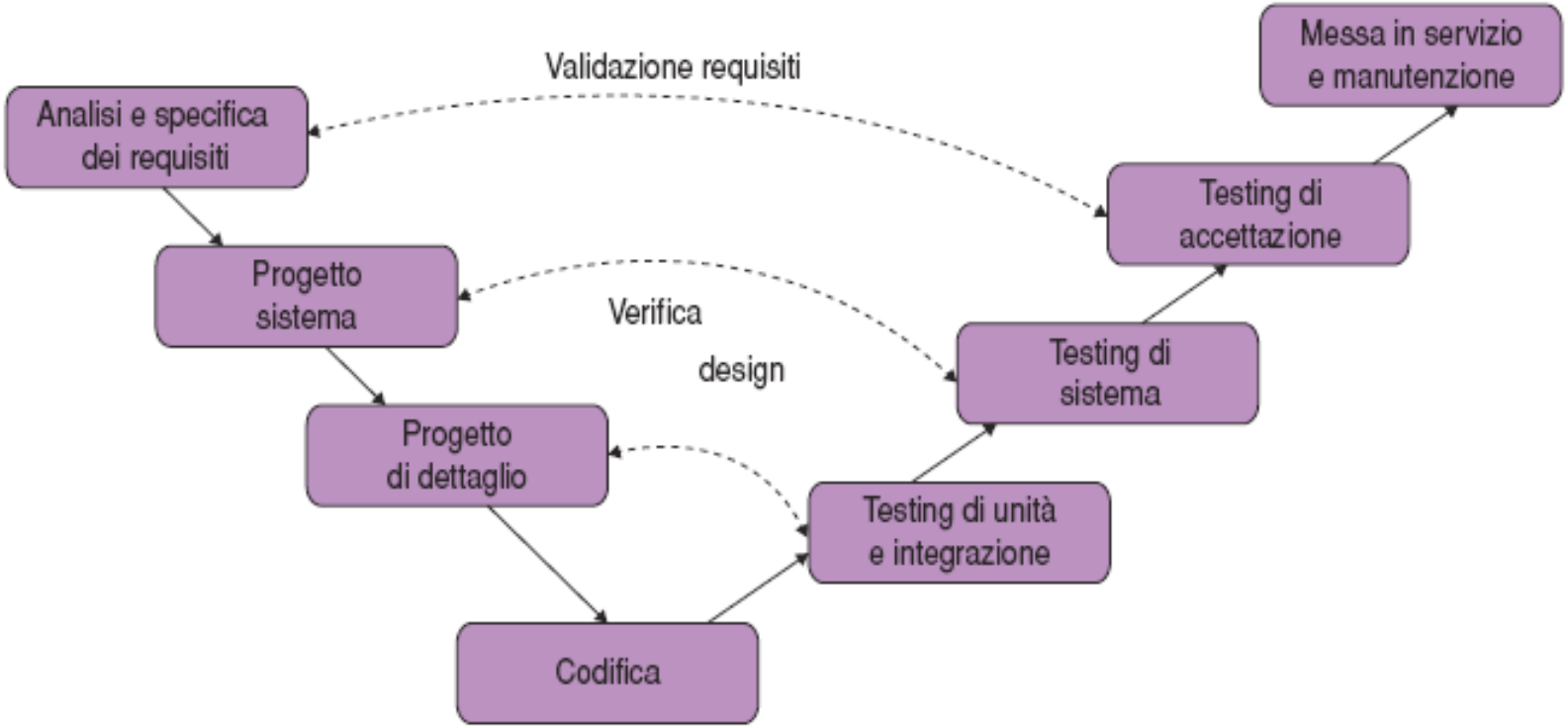
- Sequenziali
  - a cascata
  - a V
- Incrementali
  - RAD (Rapid Application Development)
  - Incrementale
- Evolutivi
  - a spirale
  - modelli agili

Modello a cascata (waterfall)



Ogni singola attività deve essere completata prima del passaggio alla successiva. I limiti di questo modello risiedono nel fatto che spesso, in un progetto software, risulta necessario tornare alle fasi precedenti perché un elemento, divenuto indispensabile, era stato precedentemente ritenuto superfluo.

Modello a V



Rispetto al modello waterfall sono state aggiunte le dipendenze tra le diverse fasi di progetto e quelle di collaudo, a enfatizzare il fatto che i test devono essere progettati e predisposti preventivamente.

Modelli incrementali e evolutivi

I modelli sequenziali si sono rivelati inadeguati per la maggior parte dei progetti software, proprio a causa dell'impossibilità di ritornare ad attività precedenti:

- spesso i requisiti cambiano nel corso del progetto;
- spesso il prodotto necessita di revisioni prima ancora di essere terminato.

Di conseguenza sono nati diversi modelli fondati sull'idea di:

- procedere per passi successivi;
- sviluppare dei prototipi da sottoporre al cliente;
- poter tornare su fasi precedenti del ciclo di vita.

Modello a prototipazione rapida

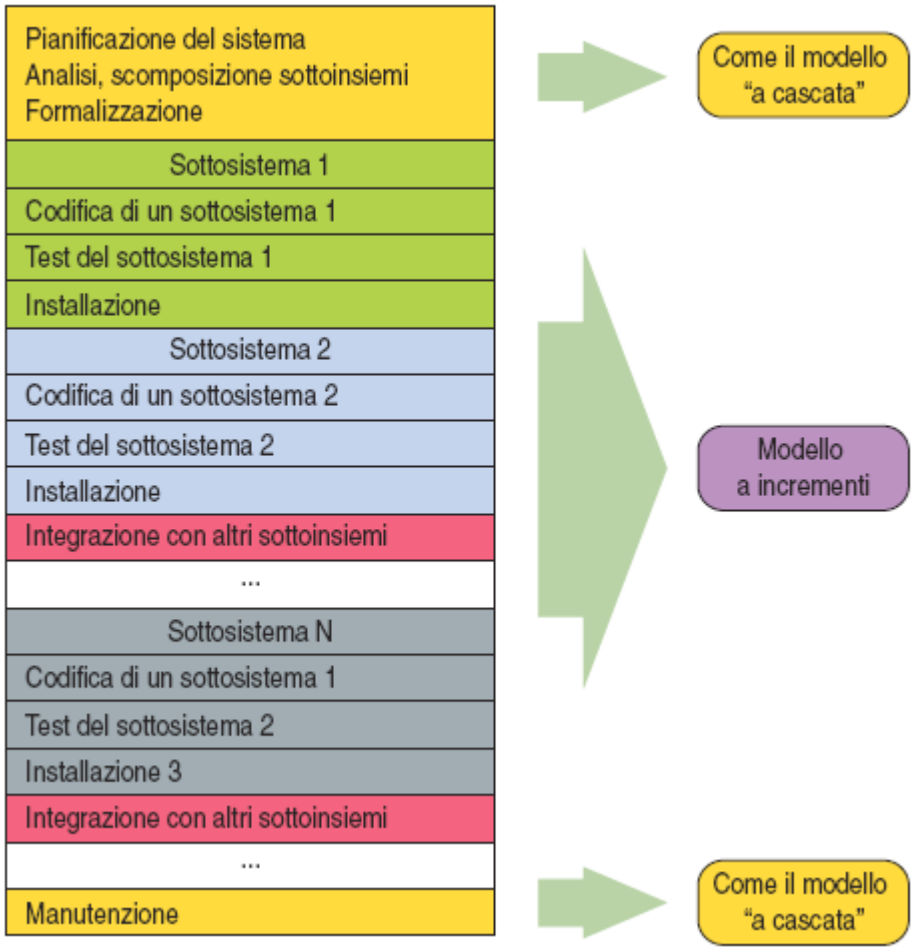
Si segue un processo iterativo che raffina di volta in volta il prodotto, fino al raggiungimento di un prodotto che soddisfa utente e sviluppatore.

Tutte le fasi si sviluppano per affinamenti successivi, generando un prototipo per ogni ciclo, da far valutare al cliente, fino al raggiungimento del prodotto finale.

Svantaggi:

- i prototipi iniziali riguardano aspetti poco significativi;
- il prototipo viene usato come base di sviluppo e non come chiarificatore di requisiti, inficiando l'intero progetto con i limiti del prototipo stesso.

Modello incrementale



Le fasi preliminari e finali seguono il modello a cascata.

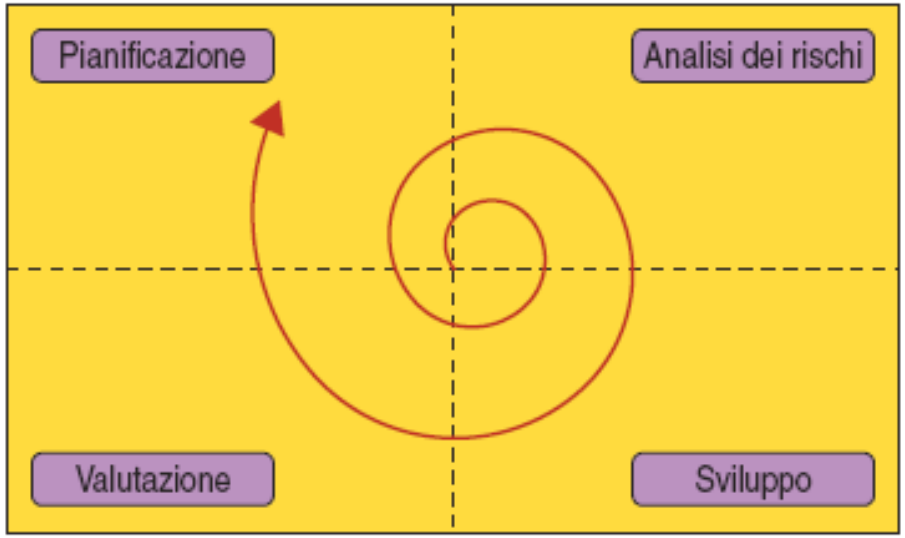
Durante l'analisi si devono individuare i vari sottosistemi da sviluppare in modo incrementale partendo da quelli con criticità maggiore, fino a quelli meno critici.

Ogni sottosistema soddisfa una parte dei requisiti.

Ogni sottosistema deve essere il punto di partenza per quello successivo, in base al concetto di riuso de sottosistemi.

Ogni sottosistema deve essere integrato con quelli precedenti, testando ogni volta quanto ottenuto.

Modello a spirale



Questo modello include i precedenti: stesse fasi del modello a cascata ma con tempi più ridotti, ritornando nuovamente alla pianificazione per un nuovo ciclo più dettagliato.

- Pianificazione: committente e fornitore interagiscono per determinare i requisiti.
- Analisi dei rischi: includono quelli derivanti da fattori di costo, di tempo e di variazione delle specifiche, cercando di individuare delle strategie per controllarli.
- Sviluppo: codifica e verifica, con dei tempi molto lunghi.
- Valutazione: il committente valuta se il sistema realizzato risponde alle sue esigenze. In caso negativo si parte con un nuovo ciclo.

Il modello presuppone che ci si possa fermare dopo un solo giro, ricadendo nel modello a cascata, o usando il modello a prototipi.

Metodologie agili

Le metodologie agili sono metodologie adattative e non predittive come le precedenti. Si adattano ai requisiti utente invece di cercare di prevedere come evolverà il sistema software.

Si basano su un processo iterativo di progettazione, sviluppo e test molto brevi.

Principali caratteristiche delle metodologie agili:

- Iteratività e incrementalità: le fasi sono compresse in tempi molto brevi, concentrandosi sulla soluzione di pochi problemi, piccoli e ben definiti.
- Rilasci frequenti: i tempi ridotti permettono di produrre rilasci più frequenti.
- Testing: verifica continua del codice, dei dati, dei modelli e della documentazione.

Le metodologie agili si adattano a progetti dove i requisiti utente cambiano continuamente.

Metodologie agili

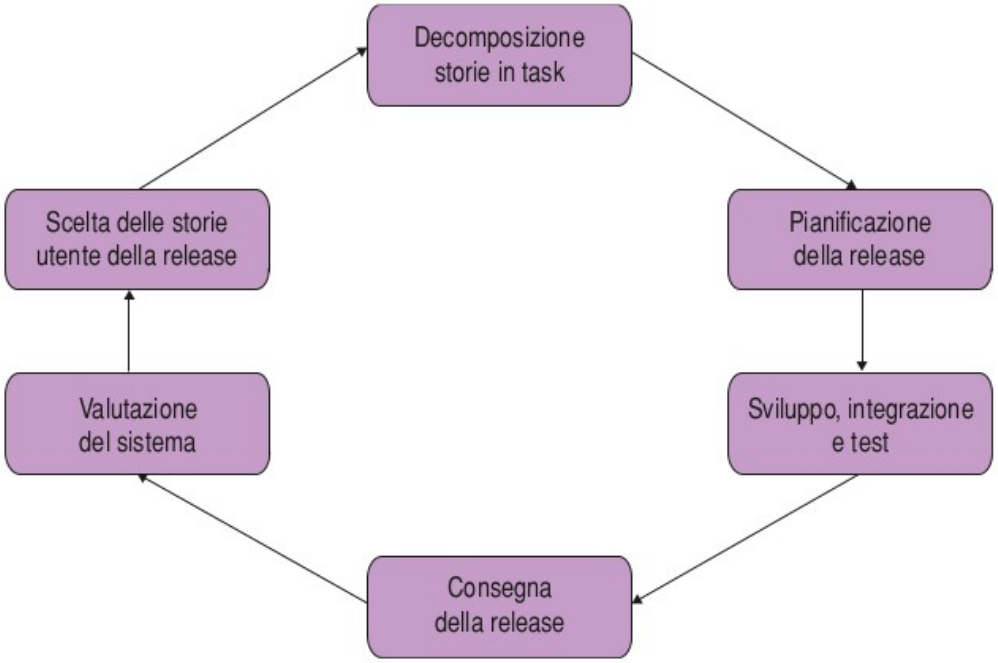
Regola	Descrizione
Pianificazione realistica	I clienti devono prendere le decisioni sulla funzionalità, i programmatori devono prendere le decisioni tecniche. Aggiornate il piano di sviluppo quando è in conflitto con la realtà
Piccoli stati di avanzamento	Fornite velocemente un sistema utilizzabile, e fornite aggiornamenti in tempi brevi
Metafora	Tutti i programmatori dovrebbero condividere un racconto che illustri il sistema in fase di sviluppo
Semplicità	Progettate ogni cosa in modo che sia la più semplice possibile, invece di predisporre tutto per future complessità
Collaudo	Sia i programmatori sia i clienti devono preparare casi di prova. Il sistema deve essere collaudato continuamente
Riprogettazione	I programmatori devono continuamente ristrutturare il sistema per migliorare il codice ed eliminare parti duplicate
Programmazione a coppie	I programmatori devono lavorare a coppie e ciascuna coppia deve scrivere codice su un unico calcolatore
Proprietà collettiva	Tutti i programmatori devono poter modificare qualsiasi porzione di codice quando ne hanno bisogno
Integrazione continua	Non appena un problema è risolto, mettete insieme l'intero sistema e collaudatelo
Settimana di 40 ore	Non usate piani di lavoro poco realistici, riempiendoli di sforzi eroici
Cliente a disposizione	Un vero utilizzatore del sistema deve essere disponibile in qualsiasi momento per la squadra di progettazione
Standard per la scrittura del codice	I programmatori devono seguire degli standard di codifica che pongano l'accento sul codice autodocumentato

Metodologie agili

Tra i metodi agili ricordiamo:

- Extreme Programming XP (Beck & Cunningham)
- Feature Driven Development FDD (De Luca & Coad)
- SCRUM (Sutherland)
- Crystal (Cockburn)
- DSDM (Dynamic System Development Method)
- Lean Software Development (Poppendieck)

Extreme Programming



Approccio estremo allo sviluppo software:

- User story: requisiti espressi come scenari; raggruppati per l'implementazione in tempi brevi.
- Simple design: sviluppo concentrato solo sugli aspetti semplici, procedendo per incrementi.
- Unity test: i test delle user story vengono implementati prima del codice (chiarisce i requisiti) ed eseguiti quotidianamente.
- Refractoring: continuo miglioramento incrementale iterativo del codice senza alterarne il comportamento esterno.
- Pair programming: i programmatori lavorano a coppie, sinergicamente.
- Insite customer: il cliente collabora a tempo pieno con lo sviluppatore.

Extreme Programming

Linee guida:

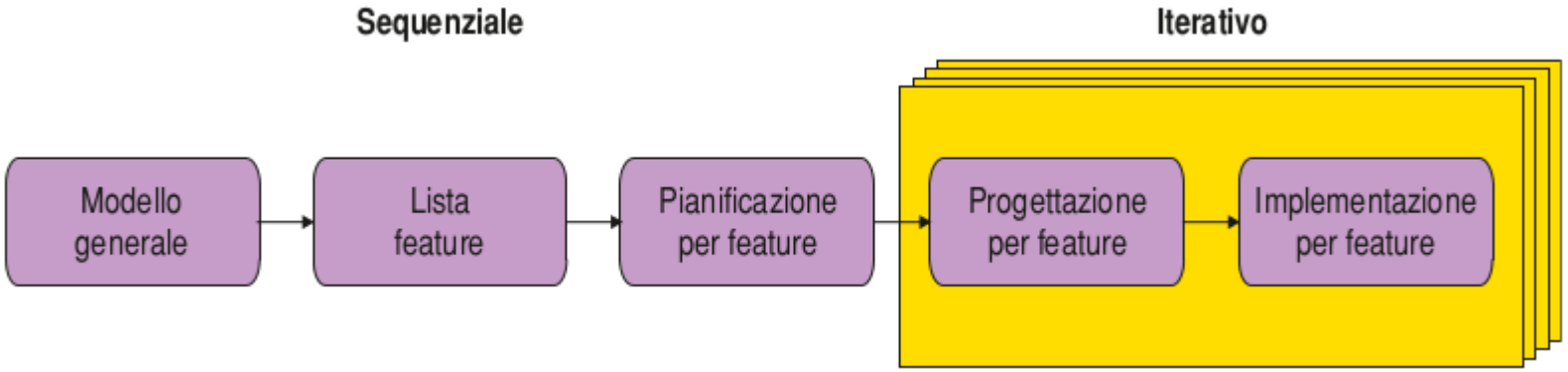
- Comunicazione: tutti possono parlare con tutti, persino l'ultimo dei programmatori con il cliente.
- Semplicità: gli analisti mantengano la descrizione formale il più semplice e chiara possibile.
- Verifica: sin dal primo giorno si prova il codice.
- Coraggio: si dà in uso il sistema il prima possibile e si implementano i cambiamenti richiesti man mano.

Extreme Programming

Problemi:

- Cliente non disponibile: difficilmente il cliente può essere sempre a disposizione e rappresentativo di tutti gli utenti finali.
- Refractoring troppo oneroso: scelte errate nelle prime fasi possono rendere troppo oneroso il progetto e portarlo al fallimento.
- Test troppo semplici: essendo spesso automatici possono essere troppo semplici e poco significativi.
- Scarsità di documentazione: eccessiva interazione verbale.
- Difficile riutilizzo: il codice è molto specifico alla user story implementata.

Feature Driven Development FDD



La metodologia è organizzata in cinque fasi di sviluppo (processi) suddivise in due parti:

- Parte sequenziale: composta dalle prime tre fasi che servono a stabilire una forma per il modello complessivo.
- Parte iterativa: composta dalle fasi di progettazione e implementazione che vengono **iterate** per ciascuna feature (piccolo pezzetto di una funzionalità dell'applicazione che ha valore per il committente) fino al loro completamento.