Glossario (e riassunto) di Ingegneria del Software

Francesco Parolini

Indice

Premessa

Questo documento contiene i termini del glossario e talvolta spiegazioni di alcuni argomenti relativi al corso di Ingegneria del Software dell'anno 2017/2018. I significati sono stati dati in base alla mia interpretazione *personale* ($\Longrightarrow$ possibilmente sbagliata), a ciò che è scritto nelle slides e alle spiegazioni in aula($\Longrightarrow$ possibilmente mal recepite). Non c'è alcuna garanzia di correttezza ne tanto meno di completezza (andando verso gli ultimi argomenti in particolare ci saranno sempre meno spiegazioni e termini). Per questo motivo non ho caricato (e non voglio che *nessuno* carichi) questo documento su repository (Mega) al di fuori di quella in cui si trova, che terrò pubblica. Un grazie *al buon Ciprian* per le correzioni.

1 Introduzione

Progetto insieme di attività e compiti con le seguenti proprietà:

i. Devono raggiungere determinati obiettivi con specifiche fissate
ii. Hanno date di inizio e fine fissate
iii. Possono contare su limitata disponibilità di risorse
iv. Consumano risorse nel loro svolgersi

In particolare le attività sono Pianificazione → Analisi dei requisiti → Progettazione → Realizzazione. Attività $\neq$ compiti: i compiti sono i componenti elementari delle attività

Prodotto SW per...

commessa con forma, contenuto e funzione fissate dal committente. (Es. software per uso interno di un'azienda)

pacchetto con forma, contenuto e funzion idoneo alla replicazione. (Es. Vim)

componente con forma, contenuto e funzione adatte alla composizione. (Es. Plugin di Vim)

servizio con forma, contenuto e funzione fissate dal problema.
(Es. Google maps)

Ciclo di vita del prodotto SW gli stati che il prodotto assume dal concepimento al ritiro

Efficacia grado di conformità del prodotto. Misura la capacità di raggiungere gli obiettivi fissati. La strategia con il quale il fornitore garantisce *conformità* e pertanto *efficacia* viene fissata nel *Piano di Qualifica*

Efficienza quanto spreco *non* è stato fatto. Misura l'abilità di raggiungere gli obiettivi impiegando il numero minimo di risorse. È in contrasto con l'*efficacia* in quanto per perseguire quest'ultima in generale lo *spreco* di risorse va massimizzato

Best practice *way of working* noto che abbia mostrato di garantire i migliori risultati in circostanze note e specifiche; principi noti ed autorevoli

Manutenzione di tipo...

correttiva se si occupa della rimozione di difetti

adattiva se si occupa del raffinamento dei requisiti

evolutiva se si occupa dell'evoluzione del sistema

Ingegneria del software l'approccio *sistematico*, *disciplinato* e *quantificabile* allo sviluppo, l'uso, la manutenzione ed il ritiro (stati di uso) del SW, che garantisca *qualità* (misurata da *efficacia*) ed *efficienza*

Approccio sistematico è un metodo di lavorare metodico *(svolto secondo una precisa linea di condotta o procedimento tecnico, seguendo una lista di azioni predefinita)* e rigoroso. Che studia, usa ed evolve le *best practice* di dominio

Disciplinato che segue regole fissate

Quantificabile che permette di verificare efficienza ed efficacia *(i quantificatori di cui disponiamo ad ora)*

Stakeholder(portatore di interesse) l'insieme di coloro che a vario titolo hanno influenza sul prodotto, sul progetto, sui processi.
Es: la comunità degli utenti (utilizzatori), il committente (compratore), il fornitore (sostiene i costi di realizzazione), eventuali regolatori (verificano l'attuazione dei processi)

Way of working la maniera di rendere *sistematiche, disciplinate* e *quantificabili* (vedi Ingegneria del software) le attività di progetto → *ingegnerizzatore delle attività* (FP)

Le 4 P del SWE

- People
- Product
- Project
- Process

2 Processi Software

Possiamo vedere il ciclo di vita del software come una macchina a stati finiti, dove gli stati rappresentano lo stato nel quale si trova il prodotto (concepimento fino a ritiro) e gli archi da uno stato all'altro sono le attività che servono per portare uno stato in un altro.

Processi di ciclo di vita *way of working* per permettere le transizioni nel ciclo di vita del SW
(oppure) specificano le attività da svolgere per abilitare corrette transizioni di stato nel ciclo di vita di un prodotto SW

Modelli di ciclo di vita astrazioni per *possibili* cicli di vita. Descrivono come i processi si relazionino rispetto agli stati di ciclo di vita. Un particolare modello aiuta a pianificare, organizzare, eseguire e controllare lo svolgimento delle attività necessarie al ciclo di vita. É dunque uno strumento organizzativo di supporto. Disporre di molti modelli di

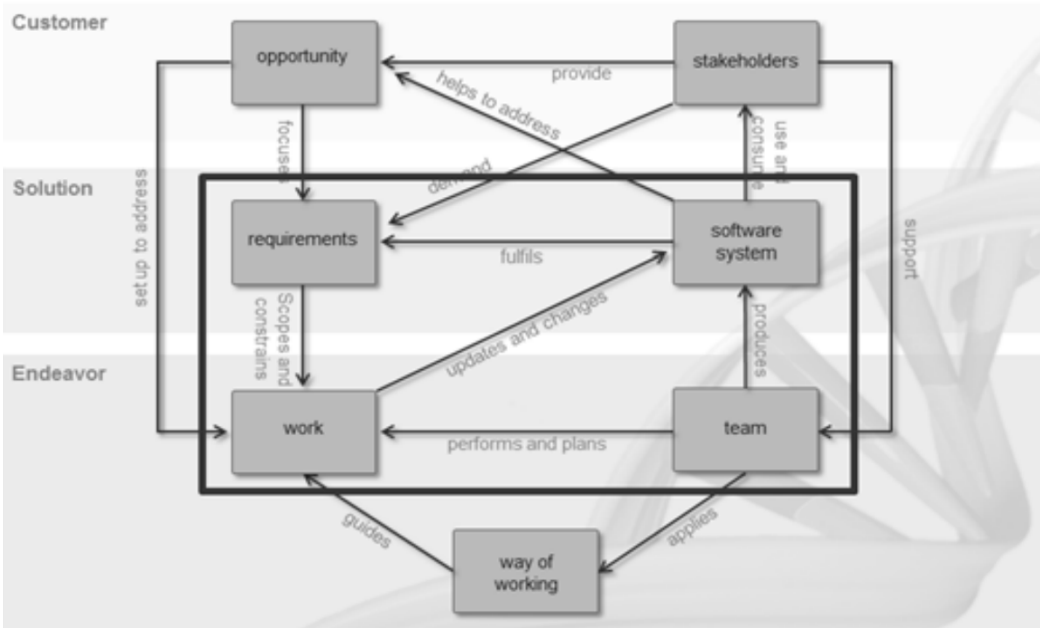

Figura 1: elementi di un progetto secondo SEMAT

cicli di vita ci aiuta a sviluppare un progetto SW. I modelli di cicli di vita sono *astratti*.

Iterazione una iterazione è un raffinamento o rivisitazione di qualcosa. Rappresenta il *tentativo* di svolgere un qualche tipo di incremento: la riuscita non è garantita. È paragonabile al ciclo while: a priori non si conosce quando verrà raggiunto un obiettivo.

Incremento è un particolare tipo di iterazione dove il successo è garantito. È paragonabile al ciclo for: il numero di incrementi è noto a priori.

Prototipo è *(astrattamente)* uno strumento per provare e scegliere una o più soluzioni ad un problema. Ha la caratteristica di poter essere "usa e getta", in quanto un prototipo non è una soluzioni finale ma una *prova* di soluzione. È uno strumento che va usato con cautela in quanto non da garanzia di successo o di un *incremento* futuro:

il primo tentativo fallisce $\nRightarrow$ il secondo avrà successo. Se utilizzo il prototipo nella soluzione esso andrà a rappresentare la *baseline*

Riuso utilizzo di un componente prodotto dal diretto interessato o terzi per un nuovo scopo. Ce ne sono di due tipi:

i. **Occasionale**, ovvero un copia e incolla opportunistico, dove l' utilizzatore non cerca veramente di comprendere il funzionamento del componente ma si limita ad utilizzarlo in modalità "monkey". Ha un basso costo dal punto di vista delle tempistiche, ma se in un futuro si dovesse riutilizzare ancora lo stesso componente il lavoro andrebbe ripetuto in modo uguale. A tal punto potrebbe convenire un riuso *sistematico*. Comunque questo tipo di riuso ha uno scarso impatto.

ii. **Sistematico**, dove si *capisce* realmente ciò che si usa. Ha un costo maggiore ma porta ad un maggior impatto

Controllo di versione sistema (*automatizzato*) per la gestione della storia del SW. Dato che un *buon* software non è immutabile è inevitabile dover fare della manutenzione: se volessi tornare indietro dopo averla fatta non potrei senza il controllo di versione.
Mantra: mai sovrascrivere, crea una nuova versione in cui dici cosa cambia

Configurazione atto di costruzione di *un solo* a partire da più *parti* in un *ordine*

Controllo di configurazione gestore delle parti che compongono il prodotto sw: definisce *quali* e *come* stanno insieme queste parti...

Processo *(Glossario ISO 9000)* insieme di attività *correlate* e *coese* che trasformano ingressi (o bisogni) in uscite (prodotti), perseguendo efficienza ed defficacia, secondo le regole date, consumando risorse nel farlo. È un concetto simile al thread: un filone principale che ha al suo interno diversi fili più piccoli che "vanno tutti nella stessa direzione". Una parte fondamentale del processo è il controllo che si

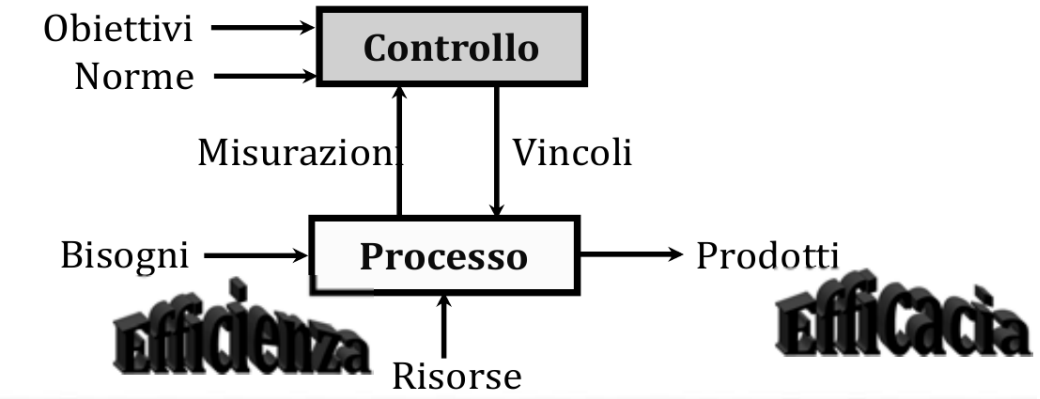

Figura 2: rappresentazione di processo

esegue su di esso: tramite delle *misurazioni* si acquisiscono dei dati ($\neq$ informazioni, opinioni). Inoltre il controllo del processo impone il rispetto dei *way of working* tramite dei vincoli. Il controllo ha come ulteriore input anche *obiettivi* e *norme*

Misurazione dato oggettivo rispetto ad una qualche grandezza. Ci sono strumenti specifici per fare tali misurazioni che desideriamo siano

i. **tempestive**, ovvero che possano dare velocemente dei dati aggiornati

ii. **accurate**, ovvero che rappresentino verosimilmente la realtà

iii. **non-intrusive**, ovvero trasparenti a colui che ha/sta sviluppando un prodotto

Per perseguire questi tre obiettivi le misurazioni devono essere *automatizzate*

Correlato è un componente rispetto ad un altro quando i due hanno un motivo per stare insieme, hanno un rapporto di reciprocità, uno scopo comune. Un insieme di attività è *correlato* se esse perseguono lo stesso scopo. Nota come il focus sia il *fine*. Correlato è un sistema

composto di parti le quali hanno tutte il fine ultimo di perseguire il medesimo scopo

Coeso è detto un sistema le parti di cui è composto sono **tutte e sole** quelle necessarie. Ad esempio una posatiera con dentro un martello non è coesa: non tutte le parti sono utili. Se togliessi il martello e anche il cucchiaio da burro non sarebbe comunque coesa poichè mancherebbe qualcosa. Similmente, uniforme. Il focus di questa parola è l'utilità totale. La coesione permette di ottenere sistemi a peso minimo. La coesione aumenta la manutenibilità, riusabilità, comprensibilità dei componenti del sistema ed aiuta ad evitare interdipendenza fra componenti. La coesione (buona) può essere di tipo:

1. Funzionale, se i componenti concorrono al medesimo scopo
2. Sequenziale, se le operazioni accorpate in un modulo devono essere eseguite sequenzialmente
3. Informativa, quando le parti agiscono sulla stessa unità di informazione

Attività macro-unità (*major unit* da qui) da completare per raggiungere gli obiettivi del processo. Ha precise date di inizio e di fine, incorpora un insieme di compiti che devono essere completati, consuma risorse e consegue lavoro prodotto. Può avere relazioni di precedenza con altre attività, per esempio finish-to-start, start-to-start, finish-to-finish. I processi sono separabili e componibili *(modulari)*, mentre le attività devono essere fra loro *correlate e coese* $\implies$ tutte necessarie

Compito è la più piccola unità di lavoro soggetta alla responsabilità della gestione. È un'assegnazione di lavoro ben definita, solitamente assegnato ad una singola persona. Compiti collegati si raggruppano in *attività* e si dice che i compiti *attuano* le attività

Metrica metodi e convenzioni con cui dare un significato alle misure

Raggiungimento di obiettivi [interni o esterni] è la metrica dell'*efficacia* di un insieme di attività. Essa va verificata in corso d'opera

Produttività $\frac{\text{Quantità di prodotto realizzato}}{\text{risorse utilizzate}}$, è la *quantità* e non la *qualità* ciò che consideriamo. Inoltre risorse $\neq$ tempo. È la metrica dell'*efficienza* di un insieme di attività

Economicità è l'insieme di efficienza ed efficacia. Governa il *way of working* ed è un valore benefico per tutti: sia per il committente che per l'esecutore

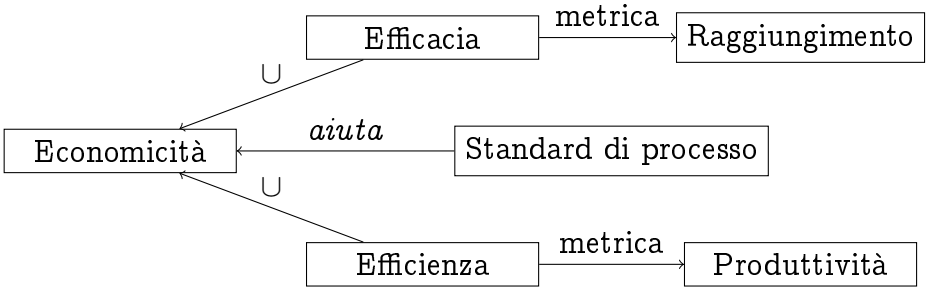

Figura 3: Relazioni fra efficienza, efficacia, economicità, raggiungimento obiettivi e produttività **per un insieme di attività**

Standard come

- **modello di azione** ovvero una sorta di ricettario che descrive le attività che compongono un processo. Può essere *imposto* oppure *proposto*. I processi vanno però *specializzati*, ovvero adattati alla specifica situazione reale. Come astrazione si può pensare alla relazione classe $\rightarrow$ istanza nei ringuaggi di programmazione: la classe è astratta e per esistere va istanziata. Un esempio di standard come modello di azione è ISO/IEC 12207
- **modello di valutazione** ovvero una valutazione del comportamento tramite delle metriche. Un esempio è l'esame universitario: non viene detto *come* studiare ma viene valutato lo studio tramite una prova, il quale esito rappresenta la metrica

ISO/IEC 12207 modello ad alto livello che identifica i processi di ciclo di vita del sw, ha una struttura modulare che richiede specializzazione,

specifica le responsabilità dui processi e indentifica i prodotti di questi ultimi. Si suddivide in tre macroaree

i. **Primari** $\exists$ un progetto $\Longleftrightarrow \exists$ istanze di questi processi

ii. **Secondari (di Supporto)** aiutano i processi primari, vengono istanziati solo se richiesto

iii. **Organizzativi** sono processi di organizzazione di *più* progetti e rendono produttiva un'organizzazione. Sono trasversali agli altri

Figura 4: Standard ISO/IEC12207

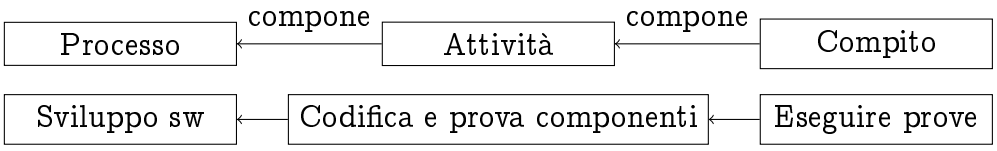

Figura 5: Esempio di processo, attività e compiti

Principio del miglioramento continuo *Ciclo di Deming* è un metodo di organizzazione interna dei processi composto da 4 fasi cicliche:

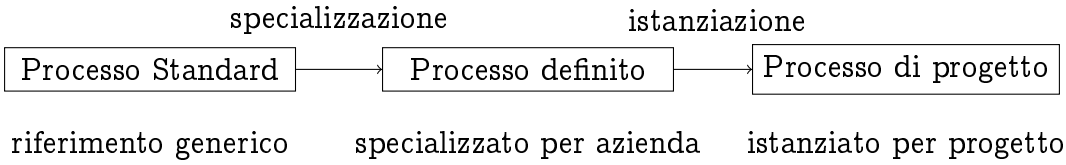

Figura 6: Processo di specializzazione

i. **Plan**: definisce attività, scadenze, responsabilità, risorse utili a raggiungere **specifici obiettivi di miglioramento**. Definisce due cose: obiettivi e strategie per perseguirli.

ii. **Do**: esegue le attività secondo il plan

iii. **Check**: verifica l'esito delle *azioni di miglioramento* ($\neq$ intero processo) rispetto alle attese

iv. **Act**: applica correzioni alle carenze rilevate e standardizza quanto è andato bene. Se i risultati sono troppo imbarazzanti rispetto a quanto ci si aspettava può essere utile un'analisi del fallimento

3 Modelli di cicli di vita

Fase è un segmento *contiguo* sull'asse del tempo che rappresenta lo stazionamento in uno stato del ciclo di vita o in una transizione fra stati. Gli stati del ciclo di vita/le attività che decidiamo di intraprendere non abbiano l'informazione relativa al tempo. Le fasi in generale possono essere sovrapposte una all'altra

Code n' Fix processo di realizzazione del software adottato da coloro i quali non procedono in maniera sistematica e disciplinata

Modello iterativo è una *proprietà* di un qualsiasi modello, consiste nel lasciare la possibilità ad un modello di tornare ad una fase precedente. Consente maggiore capacità di adattamento ma comporta il rischio di non convergenza all'obiettivo del progetto. In generale si decompone la *realizzazione* del sistema e si trattano prima le parti critiche

limitando superiormente il numero di iterazioni. I primi incrementi possono essere frutto di prototipazione. Nel modello sequenziale ogni ciclo di ritorno raggruppa sottosequenze di fasi

3.1 Modelli di cicli di vita

1. **Modello Sequenziale (Cascata)** è il più vecchio ed è caratterizzato dal fatto di essere un modello *iperdisciplinato*: le fasi sono rigidamente sequenziali e il ritono a fasi precedenti non è permesso. I prodotti sono principalmente documenti, si parla di modello *document driven*. Ogni stato, che in questo modello coincide anche con la fase, è caratterizzato da pre e post condizioni da soddisfare. Le fasi sono distinte e *non sovrapposte* nel tempo. I difetti principali sono che è un modello troppo rigido e che esprime una visione burocratica e poco realistica del progetto, non permette una rivisitazione dei requisiti e richiede molta manutenzione per garantire economicità. Questo modello non prevede rilasci multipli e successivi: tutte le feature in una botta sola *(bigbang integration)*. Le fasi fra loro hanno dipendenze causali: entrare, uscire e stazionare in una fase comporta lo svolgimento di di azioni specifiche. Correzioni: prototipazione e cascata con ritorni

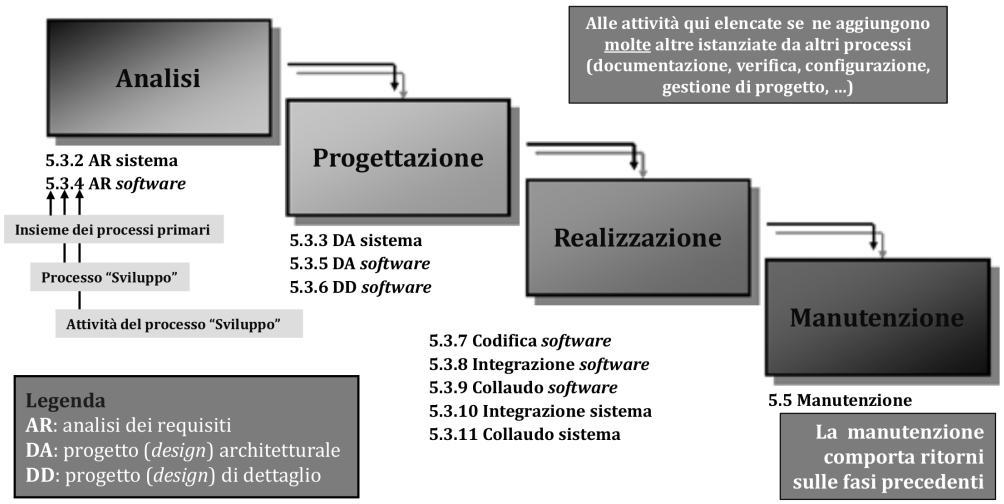

Figura 7: Modello a cascata

2. **Modello Incrementale** prevede rilasci multipli e successivi, dove ciascuno di essi realizza un incremento di funzionalità. I requisiti vengono trattati per importanza: le major features saranno le prime ad essere realizzate divenendo così presto stabili, mentre quelle meno importanti avranno più tempo per armonizzarsi con il resto del sistema. Analisi e progettazione architetturale non vengono tuttavia ripetute: i requisiti devono essere chiari sin dalla prima release e rimarranno tali per le successive, diventa così possibile pianificare i cicli di incremento. Ciò che rende incrementale questo modello è la *realizzazione*. I prototipi non sono visibili al committente

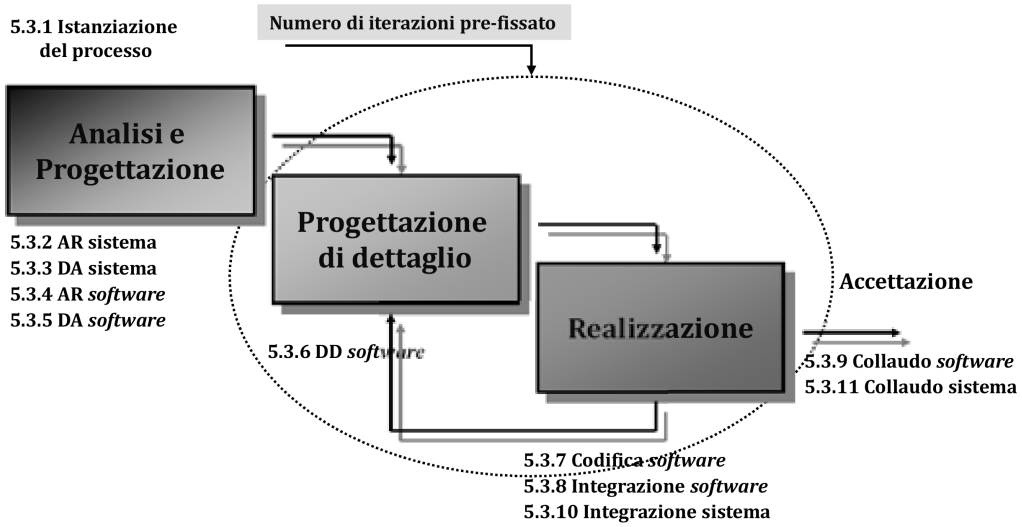

Figura 8: Modello incrementale, notare come nel ciclo ci siano solo progettazione in dettaglio e realizzazione in un numero fissato di iterazioni

3. **Modello Evolutivo** quando i bisogni evolvono nel tempo questo modello aiuta l'evoluzione del sistema. La grande differenza con il modello incrementale è che l'analisi e progettazione iniziali vengono poi raffinate con analisi e progettazioni successive. Questo modello comporta il rilascio e il mantenimento di più versioni parallelamente. Le fasi sono

   i. **Analisi preliminare** dove i requisiti e l'archiettura sono definiti in termini di massima. Inoltre si pianificano i passi, analisi e realizzazione evolutiva

   ii. **Analisi e realizzazione di un'evoluzione** che procede per raffinamento ed estensione dei requisiti che inizialmente non erano noti e relativa implementazione

   iii. **Rilascio prototipo** che diviene sempre più completo fino ad arrivare ad una *versione finale*

A parte la prima, le altre fasi ammettono iterazioni multiple e parallele. Il pregio è che permette di rispondere a bisogni non chiari inizialmente. I difetti sono: mantenimento e rilascio di più versioni parallelamente e riattraversamento di stati del ciclo di vita

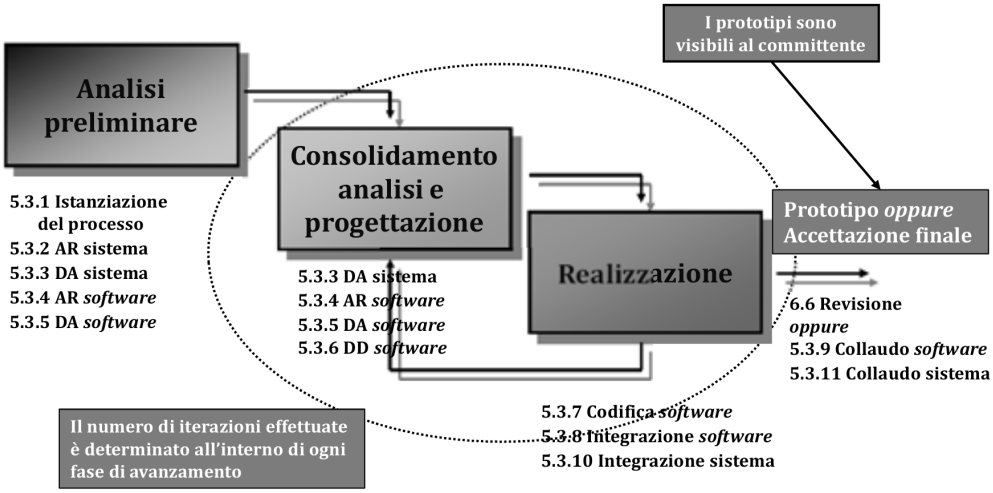

Figura 9: Modello evolutivo

4. **Modello a Componenti** dato che buona parte dei componenti che desideriamo usare è già stato scritto è utile riutilizzarlo adattandolo di volta in volta. Diventa fondamentale l'uso *sistematico* delle componenti. Il vantaggio è la rapidità nel produrre il software siccome la maggior parte è stata già scritta

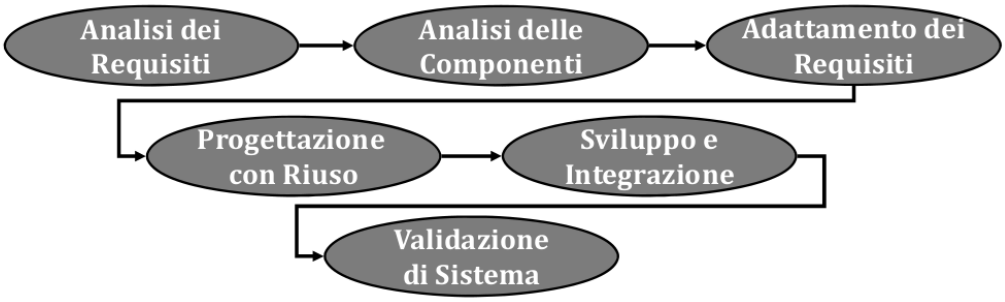

Figura 10: Modello a componenti

5. **Agile** nasce per rispondere al troppo lungo periodo di sviluppo del SW, utilizzando rilasci rapidi e incrementali, che si adatino dinamicamente ai nuovi requisiti. È indicato per dare rapidamente ai clienti un prodotto. Nasce per svincolarsi dall'eccessiva rigidità degli altri modelli di vita.

   Minuta prima stesura, provvisoria e suscettibile di mutamenti, di uno scritto, brutta copia

   User story un documento informale nel quale il cliente racconta cosa si aspetta che il prodotto faccia. È composta da una descrizione del problema, dai verbali di discussione con il cliente e da una possibile soluzione del problema

   Detto questo *agile* non è un modello di ciclo di vita ma piuttosto un insieme di principi che alcuni metodi di sviluppo del sw possono abbracciare (Sommerville capitolo 3, pagina 61). Comunque i principi fondanti sono

   i. Coinvolgimento del cliente
   ii. Accettare i cambiamenti
   iii. Consegna incrementale
   iv. Mantenre la semplicità
   v. Persone, non processi

vi. Poca documentazione, tanto software

Dunque i metodi agili sono metodi *incrementali*, con incrementi di piccole dimensioni e veloci, aperti al riconoscimento di nuovi requisiti.

oppure

i. Individuals and interactions over processes and tools

ii. Working sofware over comprehensive documentation

iii. Customer collaboration over contract negotiation

iv. Responding to change over following a plan

Vogliamo software non documenti, costruito insieme al cliente e capace di adattarsi alle situazioni. Il sw senza documentazione è un costo, senza un piano non si possono valutare rischi e cambiare si può, ma con la consapevolezza di costi/benefici. Un esempio di metodo agile è lo **Scrum** (Sommerville, p.75). I punti salienti sono:

- la presenza di un product backlog: una lista di cose da fare per il completamento del prodotto (es: refactoring interfaccia utente)
- la presenza di uno sprint backlog: lista di cose prese dalla product backlog che si andranno ad eseguire nel prossimo *sprint*(di durata dalle 2 alle 4 settimane)
- scrum quotidiano: riunione con tutti i membri sull'avanzamento del lavoro
- sprint planning, sprint review, sprint retrospective

3.2 Modelli iterativi vs modelli incrementali

I vantaggi degli **incrementali** sono:

1. producono valore ad ogni incremento: insieme di funzionalità presto disponibile. I primi incrementi possono essere frutto di prototipazione, che aiutano a fissare meglio i requisiti per gli incrementi successivi

2. ogni incremento riduce il rischio di fallimento (senza azzerarlo per il rischio di collassare in iterazione)

3. le funzionalità core del sistema vengono sviluppate durante i primi incrementi $\Longrightarrow$ più volte verificate $\Longrightarrow$ sempre più stabili nel tempo

I vantaggi degli **iterativi** sono:

1. flessibili: applicabili a qualunque modello di ciclo di vita

2. maggiore capacità di adattamento (il vantaggio di poter rivedere i requisti)

Gli svantaggi sono il rischi di *non convergenza*. Soluzione: limite superiore al numero di iterazioni, trattando le parti critiche prima. I modelli iterativi e incrementali coincidono quando un'iterazione raffina una parte ma non ha impatto sul resto del sistema.

4 Gestione di progetto

La gestione di un progetto viene effettuata dal project manager e fondamentalmente si occupa di istanziare processi aziendali (specializzati da processi standard, come nella Figura 6), stimare costi e risorse, pianificare le attività ed assegnarle a persone, controllare le attività e verificarne i risultati. I temi trattati sono:

1. Gestione qualità: posso avere qualità se applico rigorosamente i processi e applico il ciclo PDCA

2. Pianificazione di progetto: diagrammi di gantt, pert, wbs

3. Allocazione delle risorse

4. Stima dei costi di progetto

Notare come siano i *vincoli* temporali a decidere come distribuire le attività nel tempo e non viceversa (si parla di *pianificazione all'indietro*)

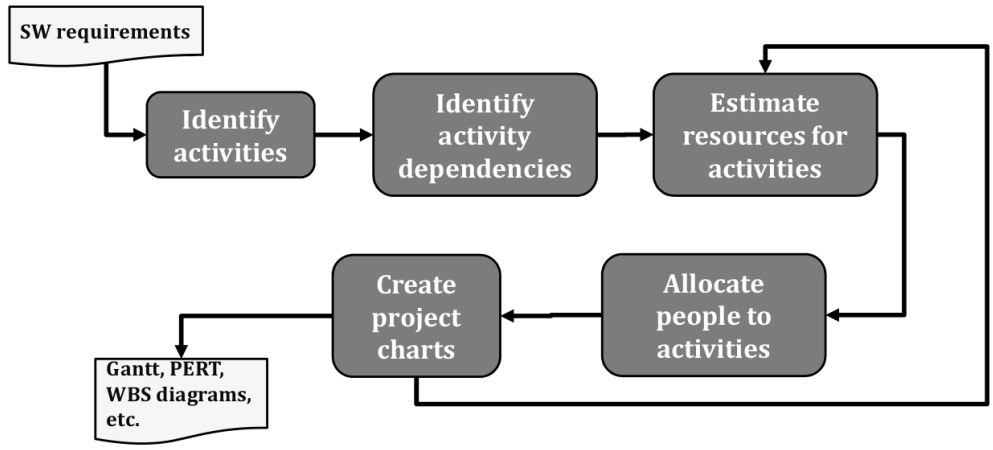

Figura 11: Attività di pianificazione, parte della gestione di un progetto

Ruolo funzione aziendale assegnata ad un progetto, ovvero per quel progetto una figura ben precisa si occupa di quel task, quando il progetto finisce il prossimo ruolo che assumerà potrebbe non essere lo stesso

Funzione aziendale ruolo fisso all'interno dell'azienda

Cammino critico sequenza di attività ordinata con prodotto importante e dipendenze temporale strette

Diagramma di Gantt formalismo grafico per rappresentare la dislocazione temporale delle attività. Rappresenta sequenzialità, parallelismo e permette di confrontare stime con progressi. In più di quello di wrike ha l'informazione riguardante la stima con l'effettività. Potrebbe avere il punto debole di non riuscire a rappresentare le sottoattività, come invece possono i diagrammi WBS

PERT permette di ragionare in modo particolare su scadenze fra attività e tempi di slack

Tempo/persona (o anche *tempo di lavoro*) il tempo di lavoro o produzione che impiega un'attività per svolgersi. Notare come è diverso dal

tempo di calendario, chiamato anche *tempo fisico*. Per esempio un task che per svolgersi richiede 10 giorni da una sola persona, ne può richiedere 5 giorni se svolto da 2 persone. Questi 5 giorni sono in termini tempo/persona poichè sono 5 solamente se 2 persone ci lavorano assieme. Tuttavia ciò dipende anche dalla qualità del lavoro prodotto

Milestone punto nel tempo al quale associamo un insieme di stati di avanzamento di una o più *baseline* che vogliamo assolutamente raggiungere. La milestone ci aiuta a pianificare come disporre le attività da svolgere nel tempo.

Baseline rappresenta tramite un insieme di stati di avanzamento strettamente sequenziali le attività che vanno svolte. È un insieme di stati di avanzamento *incrementali*. Ha la fondamentale caratteristica di essere *misurabile* (tempestivamente, accuramente e non-intrusivamente). Indica un punto di avanzamento in modo *sicuro* e con certezza per chi ci sta lavorando. La baseline deve stare nella repository del progetto. Secondo ISO 12207: *" A formally approved version of a configuration item, regardless of media, formally designated and fixed at a specific time during the configuration item's life cycle. "*

Piano di progetto è un docuemnto ufficiale redatto dal PM (project manager) che contiene fondamentalmente

1. La collocazione delle attività nel tempo
2. Le risorse disponibili e la loro assegnazione alle attività Serve per organizzare le attività con efficienza per produrre risultati efficaci. Inoltre facilita le misurazioni dell'avanzamento del tempo fissando delle *milestone*

E la sua struttura è:

- Introduzione (scopo e struttura)
- Organizzazione del progetto
- Analisi dei rischi

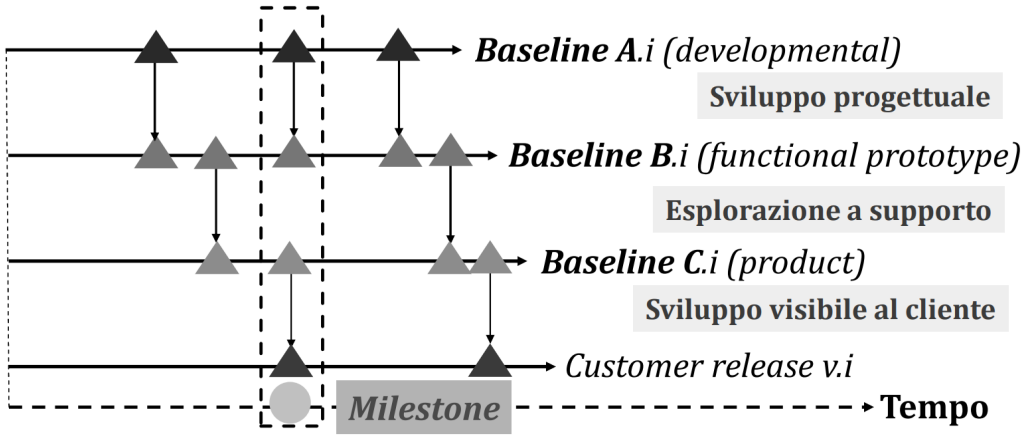

Figura 12: Rappresentazione di milestone e baseline

- Risorse disponibili (tempo + persone)
- Suddivisione del lavoro (work breakdown)
- Calendario delle attività (project schedule)
- Meccanismi di controllo e rendicontazione

Analisi dei rischi le tipologie di rischi sono:

1. tecnologie di lavoro e produzione SW
2. rapporti interpersonali
3. organizzazione del lavoro
4. tempi e costi
5. rapporti con gli stakeholders

I passi per affrontarli sono:

1. identificazione
2. analisi

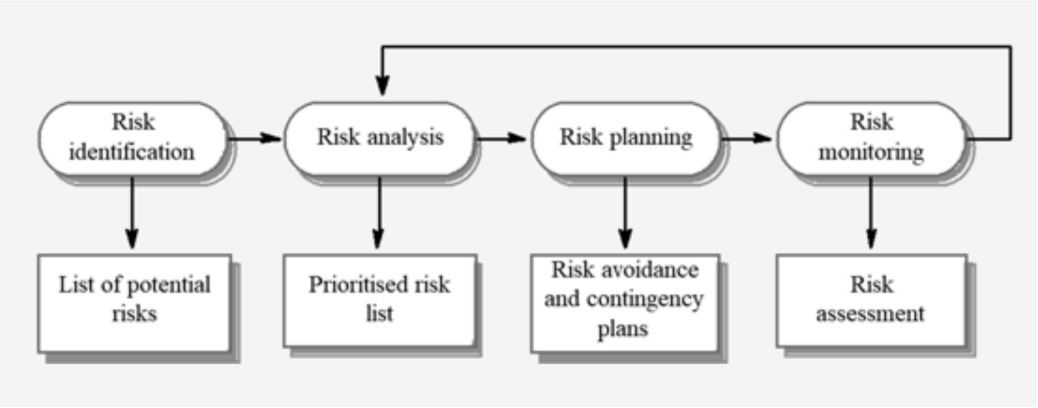

Figura 13: Ciclo di affronto dei rischi

3. pianificazione
4. monitoraggio

4.1 Ruoli

Analista il suo compito è determinare le richieste del cliente, capire il problema. Per farlo deve interagire molto con il cliente e il successo del progetto dipende in larga parte da lui. Generalmente sono pochi e seguono il progetto solo per le fasi iniziali.

Progettista deve tradurre il problema in una soluzione. Ha competenze tecniche aggiornate. Sono pochi e talvolta seguono il prodotto fino alla manutenzione.

Programmatori partecipano a realizzazione e manutenzione del prodotto, hanno competenze tecniche

Verificatori coloro che indicano cosa sta succedendo nel corso dello sviluppo del progetto verificando puntualmente come stanno andando le caratteristiche più importanti. Sono presenti per tutta la durata del progetto

Responsabile si occupa della gestione del progetto dall'inizio alla fine e gestisce pianificazione di attività, risorse umane, controllo coordinamento e relazioni esterne

Amministratore o in inglese *sys-admin* controlla l'ambiente di lavoro, gli strumenti utilizzati (per esempio configurazione e versionamento), gestisce la documentazione di progetto, risolve problemi legati alla gestione dei processi. Nelle aziende strutturate è una *funzione aziendale* e non un *ruolo*

5 Analisi dei requisiti

Requisito è interpretabile in tre diversi modi, tutti sotto il presupposto che un questo sia in qualche modo una *necessità*

i. **Requisito utente:** *dal punto di vista dell'utente*, come una capacità (*capability*) (del sistema) necessaria ad un utente per risolvere un problema o raggiungere un obiettivo. Sono vincoli contrattuali e specificano il *cosa*

ii. **Requisito SW:** *dal punto di vista della soluzione*, come una capacità che deve essere posseduta da un sistema per adempiere ad un obbligo. Specificano il *come*

iii. *dal punto di vista della documentazione*, come una descrizione documentata di una capacità interpretata come i. o ii.

Capitolato d'appalto *prodotto documentale* a carico del cliente che specifica i requisiti utente, ovvero *cosa*, e i requisiti sw, ovvero *come*. Viene successivamente tradotto nell'*analisi dei requisiti*, che definisce effettivamente e contrattualmente gli obblighi del fornitore (cioè chi deve produrre il software)

Studio di fattibilità *prodotto documentale* a carico del fornitore e riservato solamente ad esso (cioè interno). Deve specificare se il progetto proposto sia fattibile e deve essere redatto in un tempo relativamente

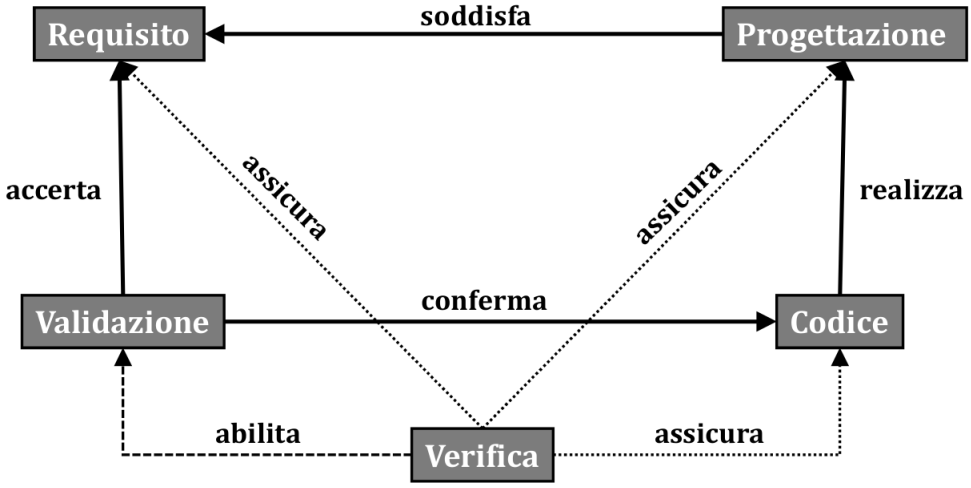

Figura 14: Relazioni fra requisiti, V&V

ristretto (tipicamente si tratta di 30 giorni) per questo è sempre redatto assieme all'analisi dei requisiti. Inoltre i due sono in qualche modo legati, studiano cose simili che si influenzano a vicenda. Lo studio di fattibilità deve valutare rischi, costi e benefici. Inoltre l'obiettivo ultimo del documento è decidere se un progetto SW è fattibile o meno. Deve anche valutare alternative ($\neq$ prendere decisioni!) riguardo strategie architetturali (sistema centralizzato o distribuito?), strategie realizzative (*make or buy?*) e strategie operative (avvio, esercizio e manutenzione del sistema, formazione e assistenza utenti).

Analisi dei requisiti *prodotto documentale* che definisce in modo formale i requisiti di un progetto. Ha un valore *contrattuale* e definisce ciò che realmente dovrà essere prodotto. Viene redatto dal fornitore ed approvato dal cliente. È lo stadio successivo del capitolato d'appalto. Definisce i requisiti SW

Specifica tecnica *prodotto documentale tecnico* di progettazione che ha un linguaggio formale o semi-formale che deve definire funzioni e profilo

operazionale: aggregati funzionali (pensa alle classi) e flusso comandi (interazioni) e dati tra essi

Progettazione top-down (approccio funzionale) progettazione che appare cattiva, in quanto non favorisce il riutilizzo, poichè il codice è già specializzato al massimo. Per esempio posso avere nel codice di un programma le entità studente e docente come classi separate

Progettazione bottom-up (approccio ObjOr) buon tipo di progettazione, in quanto costruisce per specializzazione e aggregazioni di parti. Pone attenzione sul riuso di soluzioni progettuali e alla realizzazione di componenti riutilizzabili

Tracciamento attività garante che i requisiti concordati siano *tutti e soli* quelli necessari e sufficienti: tutti per la completazza (o *sufficienza*); soli per la sinteticità (nulla di superfluo)

Brainstorming incontri di gruppo la definizione di nuove idee. Servono $N+2$ partecipanti, dove il $+2$ indica uno che appunti ciò che si dice e uno che passi la parola ad uno alla volta i partecipanti per evitare che i caratteri forti si impongano

Walkthrough tecnica che permette di fare della *verifica* che non richieda esecuzione (in quanto anche i documenti vanno verificati ed essi non possono essere paragonati a programmi che passano test). Consiste di una *lettura critica e a largo spettro* di un prodotto senza fare assunzioni preliminari. Questa tecnica viene eseguita da sviluppatori e verificatori in separata sede. Si tratta sostanzialmente di una traversata "a pettine", ovvero non so dove si trovi l'errore ma sospetto che esista, dunque riesamino tutto. È una tecnica molto costosa, meglio utilizzare la seguente tecnica

Ispezione come il *Walthrough* è una tecnica che permette di fare verifica, tuttavia non si occupa di esaminare tutto ciò che va verificato, ma essendo in possesso di una *checklist* basta controllare i punti uno alla volta. Presuppone un impegno minore rispetto alla precedente tecnica ma richiede di possedere una checklist organizzata, che si costruisce

col tempo (anche negli anni). In questo caso verificatori e sviluppatori sono ruoli ben distinti. Esegue una *lettura mirata*.

I seguenti termini rappresentano gli stati di progresso nell'analisi dei requisiti secondo SEMAT e una volta istanziate sono delle *milestones*

Conceived il committente è identificato e gli stakeholder vedono sufficienti opportunità per il progetto

Bounded i macro bisogni sono chiari, i meccanismi di gestione dei requisiti (configurazione e cambiamento) sono fissati

Coherent i requisiti sono classificati e quelli essenziali (obbligatori) sono chiari e ben definiti

Acceptable da qui in poi i requisiti sono esponibili anche al cliente. I requisiti fissati definiscono un sistema soddisfacente per gli stakeholder

Addressed il prodotto soddisfa i principali requisiti al punto da poter meritare rilascio e uso. In pratica è il fornitore ad essere convinto che il prodotto soddisfi i requisiti

Fulfilled Il prodotto soddisfa abbastanza requisiti da meritare la piena approvazione degli stakeholder. Dunque qui anche questi ultimi sono convinti che il prodotto soddisfi i requisiti

La *classificazione dei requisiti* è una attività molto importante in quanto suddividerli in classi aiuta a mantenerli, ritrovarli, tracciarli (quando e com'è nata la necessità di un requisito specifico) e comprenderli. Si dividono in

i. **Attributi di prodotto**, ovvero *cosa* devo fare, requisiti funzionali, prestazionali e di qualità

ii. **Attributi di processo**, ovvero *come* lo devo fare, requisito realizzativo, contrattuale o normativo

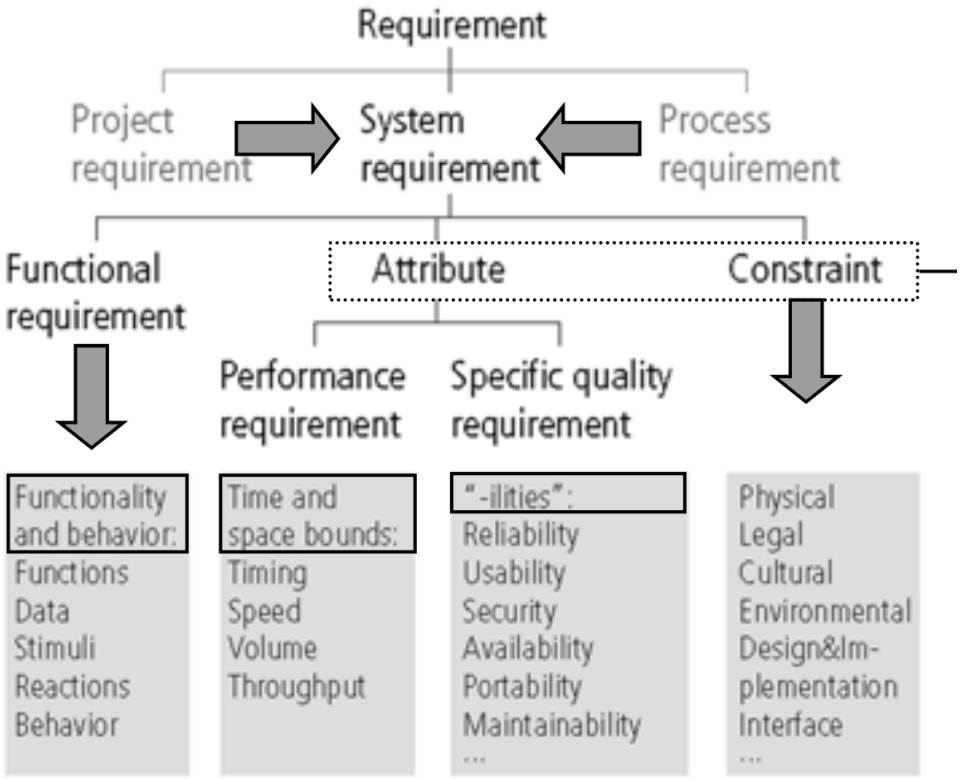

Figura 15: Classificazione dei requisiti

I requisiti in Figura 15 sono divisi in funzionali (ovvero le caratteristiche operative che il prodotto deve possedere), di attributo (ovvero quanto tempo e spazio occupa/consuma/deve entrare il sistema + *-ilities*) e costrizioni imposte da qualche entità esterna (culturali, legali, interfaccia...). Inoltre i requisiti possono essere obbligatori, desiderabili o opzionali. In generale ecco le classi dei requisiti e come verificare il rispetto del requisito in corso:

1. **Funzionali** $\rightarrow$ test, verifica formale o revisione
2. **Prestazionali** $\rightarrow$ misurazione
3. **Qualitativi** $\rightarrow$ verifica ad hoc

4. **Dichiarativi** *(di vincolo)* $\rightarrow$ revisione

6 Progettazione

Se l'analisi risponde alla domanda *"Che devo fare?"* la progettazione risponde alla domanda *"Come lo faccio?"*

Correttezza per correzione ottenere un risultato corretto continuando a sbagliare raffinando di volta in volta il risultato

Correttezza per costruzione quella che vogliamo ricavare noi: ottenere un risultato corretto progettandolo adeguatamente. Prima di implementare ciò che è stato progettato bisogna essere sicuri vada bene

Divide-et-impera approccio simile a quello dell'Impero Romano: per gestire un sistema troppo complesso si spezza un unico grande problema in tanti sottoproblemi più piccoli e semplici da risolvere

Architettura è uno strumento che permette di raggiungere un risultato ($\neq$ arte, qualcosa che dia soddisfazione). Rappresenta il primo passo della progettazione. L'architettura è:

1. la decomposizione del sistema in componenti *(che faccio?)*
2. l'organizzazione di tali componenti(ruoli, responsabilità, interazioni...) *(che fanno?)*
3. Le interfacce necessarie all'interazione tra le componenti tra loro e con l'ambiente *(come comunicano?)*
4. I paradigmi di composizione delle componenti *(come li metto insieme?)*

Design pattern architetturale soluzione progettuale generale a un problema ricorrente. Una descrizione o un modello da applicare per risolvere un problema che può presentarsi in diverse situazioni durante la progettazione e lo sviluppo del software. Organizza una responsabilità *architetturale* lasciando gradi di libertà nel suo uso

Framework struttura di supporto su cui un software può essere organizzato e progettato. Insieme integrato di componenti SW prefabbricate. Sono *bottom-up* in quanto fatti di codice già sviluppato. Possono essere anche *top-down* se impongono uno stile architetturale. Oggi per framework si intende una architettura generica riusabile

Unità nell'ambito della progettazione in dettaglio una unità è una "cosa singola", ovvero un blocco di lavoro realizzabile da un singolo programmatore

Prova di unità testing di unità, servono strumenti appropriati

Modulo aggregato di *unità*

Accoppiamento proprietà esterna di un componente, ovvero che non dipende da un componente stesso ma da coloro che lo utilizzano. È il grado di utilizzo reciproco di N componenti di un sistema. Si misura con **SFIN**, che è l'indice di utilità e va massimizzato e **SFOUT** che è indice di dipendenza e va minimizzato.

I termini seguenti sono le definizioni degli stati(sequenziali) della progettazione secondo SEMAT. Rappresentano delle possibili *milestones*

Architecture selected ho pensato ad architettura, tecnologie, dettagli come *build, buy e make* e posso difendere le mie scelte

Demonstrable so enunciare le proprietà buone della mia architettura e gli *stakeholders* concordano

Usable il sistema seppur grezzo è utilizzabile (da eventuali utenti) e corretto: le parti disponibili sono integrabili

Ready anche documentazioni e parti mancanti sono pronte. Gli *stakeholders* hanno accettato il prodotto e desiderano che diventi operativo

I benefici dell'incapsulamento sono

1. l'esterno non può fare assunzioni sull'interno

2. cresce la manutenibilità
3. al diminuire delle dipendenze aumenta la possibilità di riuso
4. algoritmi e strutture dati nascoste

7 Qualità del software

Qualità insieme delle caratteristiche di un'entità che ne determinano la capacità di soddisfare esigenze espresse e implicite [ISO9000]. La qualità tuttavia ha più punti di vista: da parte di chi usa, di chi fa e chi valuta. È profondamente legata al concetto di *valutazione*. Le visioni della qualità sono:

1. Intrinseca: conformità ai requisiti, idoneità all'uso
2. Relativa: soddisfazione del cliente
3. Quantitativa: misura del livello di qualità per confronto

è fondamentale che la qualità sia *quantificabile* nel swe, non esiste approssimazione, anche se ottenere ciò è molto difficile. Le *attività* per l'assicurazione della qualità sono (per ISO 12207):

1. *Implementazione di processo di garanzia qualità:* dove si stila una pianificazione della qualità apposta per il progetto. Questo processo dovrebbe essere coordinato con il processo di verifica e con quello di validazione. Va quindi formalizzato un *piano di qualifica*: esso va implementato, documentato e manutenuto durante il ciclo di vita del SW. Quello che il piano deve includere sono: standard di qualità, metodologie di raccolta delle misurazioni e strumenti con cui fare ciò. Inoltre dovrebbe programmare le misurazioni, queste dovrebbero essere regolari e visibili anche all'acquirente
2. *Garanzia di prodotto:* Si deve assicurare che i prodotti software e la relativa documentazione siano conformi al contratto e a prima dei piani. Inoltre prima della consegna del prodotto dovrebbe essere verificato che questo soddisfi tutti i requisiti

3. *Garanzia di processo:* Si deve assicurare che tali processi del ciclo di vita del software (sviluppo, funzionamento, manutenzione, e processi di supporto incluso l'assicurazione di qualità) impiegati per il progetto sono conformi con il contratto e aderire ai piani. Va anche qui assicurato che lo staff che produce il SW abbia capacità necessarie per farlo, eventualmente ricorrendo al processo di training
4. *Garanzia del sistema qualità:* addizionali attività possono essere inserite se così richiesto da contratto

Valutazione Giudizio oggettivo che indica un'interpretazione, positiva o negativa, delle misure

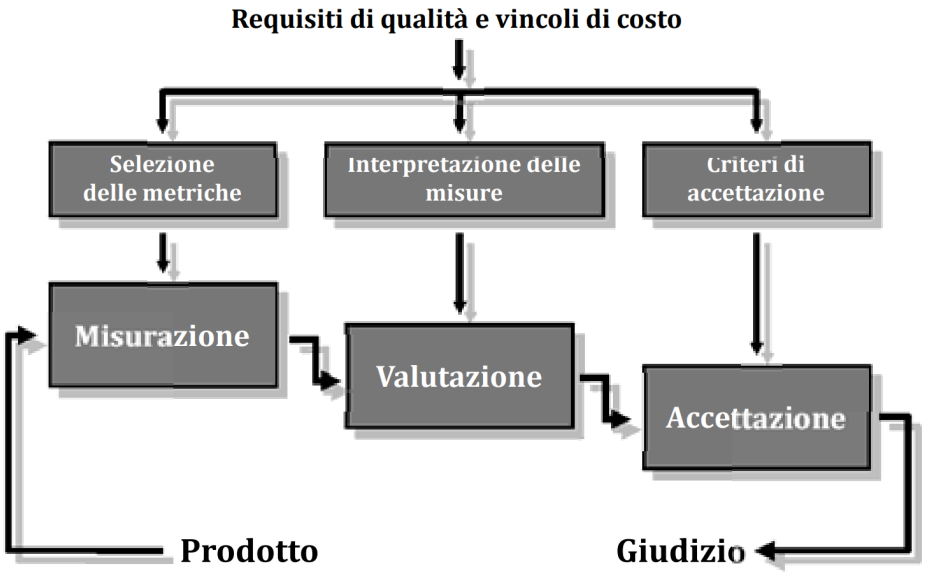

Figura 16: Processo di valutazione

Sistema di qualità *struttura organizzativa*, responsabilità (c'è qualcuno che si assume la *responsabilità* di controllare il sistema di qualità), procedure, e risorse (4 cose) (essere organizzati e responsabili → *way of*

working) messe in atto per il perseguimento della qualità [ISO 9000]. Agisce su pianificazione (definizione di politica e obiettivi), controllo e miglioramento continuo. È in pratica un insieme di azioni ben organizzato che mira a creare qualità

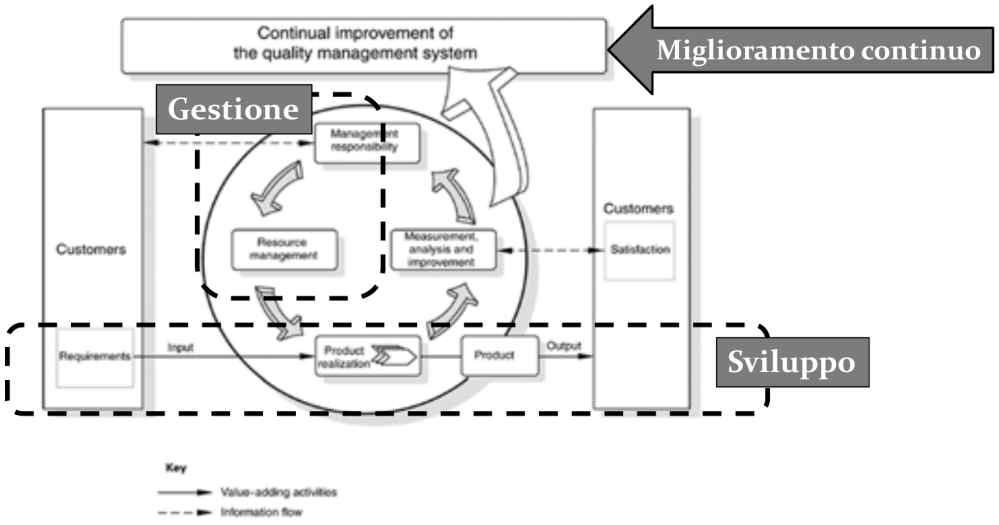

Figura 17: Sistema di qualità

Miglioramento continuo processo (non inteso come processo swe) il cui scopo è migliorare il *way of working*. È una necessità se si punta ad ottenere *qualità*. Il miglioramento continuo è fattibile se si fanno delle misurazioni su piccoli frammenti su attività ripetitive, così da poterle migliorare quando verranno ripetute. La chiave per il miglioramento continuo sono gli standard di qualità.

Pianificazione della qualità le attività del *sistema qualità* mirate a fissare gli obiettivi di qualità, e i processi e le risorse (queste due stesse sono parti del sis.qualità) necessarie per conseguirli [ISO9000].

Piano della qualità è un documento che fissa le politiche *aziendali* ($\neq$ di progetto) per il perseguimento della qualità (visione orizzontale). Inol-

tre determina gli obiettivi di qualità del singolo prodotto (visione verticale).

Controllo di qualità le attività del *sistema qualità* pianificate e attuate per assicurare che il prodotto soddisfi le attese [ISO9000]. E' il controllo dell'adesione del prodotto alla qualità. Vogliamo controllo *preventivo*, non *retrospettivo*.

Quality assurance controllo della qualità preventivo. E' un impegno preso all'inizio del progetto vincolante. $\neq$ quality insurance: per capire la differenza la quality insurance dice che se ti fai male verrai risarcito a dovere. La quality assurance dice che *sicuramente non ti farai male*. Il "caso brutto" viene garantito che non si presenti

Misurazione quantitaiva il *processo* tramite cui, secondo regole definite, simboli o numeri sono assegnati ad attributi di una entità, l'uso di una metrica per assegnare un valore (numero o categoria) su una scala predefinita

Modello di valutazione il *modello* tramite cui alle *misurazioni quantitative* vengono assegnate *valutazioni*

ISO/IEC 9126 modello di *definizione* della qualità. agisce su tre assi come si vede nella figura sotto.

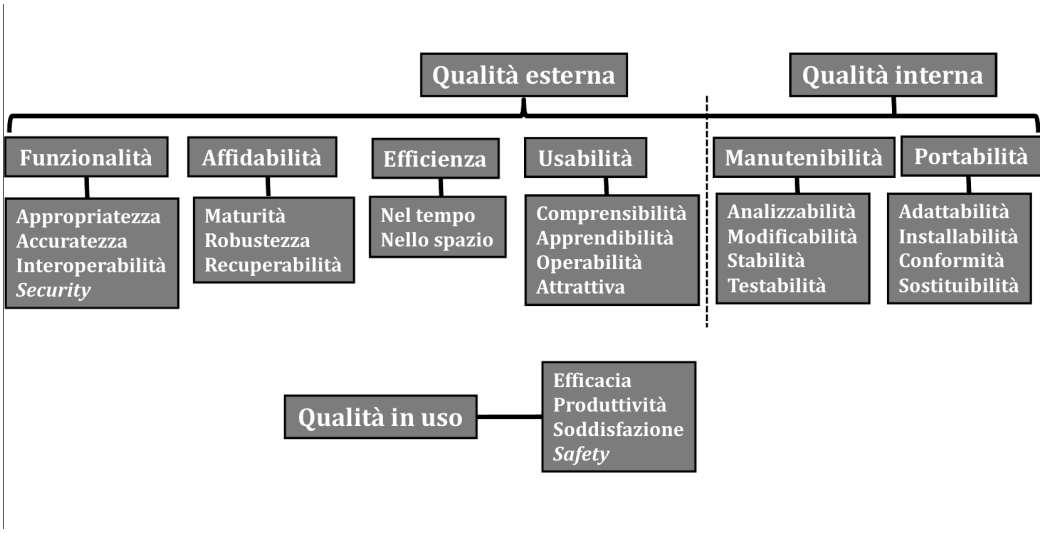

Figura 18: Standard ISO/IEC 9126: definizione di qualità

8 Qualità di processo

Manuale della qualità il documento che definisce il sistema di gestione della qualità di un'organizzazione: si integra con i processi e procedure aziendali, fissa obiettivi di qualità, strategie per perseguirli e specifica le modalità per la sua evoluzione. Contiene i principi *orizzontali* all'azienda del SGQ(sistema gestione qualità). Contiene il *way of working* dell'azienda

Piano di qualità è il documento di gestione della qualità rispetto al *singolo progetto*, visione verticale della qualità. Contiene degli obiettivi da raggiungere, motivazioni per le quali li vogliamo raggiungere, modalità di misurazione e quali esiti saranno positivi

Capability misura dell'adeguatezza di *un processo* per gli scopi ad esso assegnati. Determina efficienza ed efficacia raggiungibile da quel preciso processo. Il focus è sull'adeguatezza del *singolo* processo

Maturity è la misura di qualità per un *insieme di processi*. Sono processi di qualità quelli che applicano i principi di miglioramento continuo (vedi PDCA). Risulta dall'effetto combinato (il bottom) delle capability dei processi considerati.

Governance è la capacità di processi più evoluti di una organizzazione ad elevare i processi ancora rudimentali. Esempio: l'Italia che ha copiato dai paesi nordici il metodo di dichiarazione dei redditi. L'organizzazione sarebbe l'Europa.

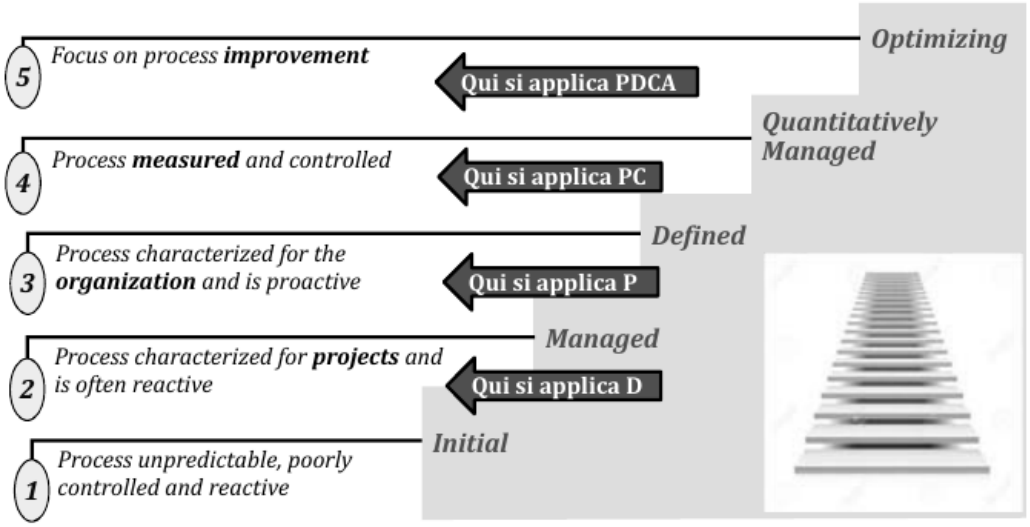

Figura 19: I 5 gradini dello stadard CMMI

9 Verifica e validazione

Verifica è un processo che si occupa di fornire evidenza oggettiva che i risultati ottenuti come output di una particolare fase del ciclo di vita di sviluppo del software soddisfino i requisiti. Per fare ciò la verifica viene eseguita durante lo sviluppo prestando attenzione alla consistenza, alla completezza, alla correttezza del software e della documentazione ad esso relativa. La verifica si occupa dunque di accertare che nella

fase in esame non siano stati introdotti errori ed è la base su cui poter passare alla validazione.

Attività di supporto che verifica *in itinere* che il lavoro svolto sia stato sviluppato in modo *corretto*. Viene eseguita sui prodotti dei processi per accertare il rispetto di regole, convenzioni e procedure vigenti. Il focus rivolto al *way of working*. Se facendo verifica osservo che sto facendo le cose in modo corretto il mio prodotto finale sarà fatto bene.

Validazione è un processo per confermare in modo definitivo che le caratteristiche del software siano conformi ai bisogni dell'utente e all'uso previsto, fornendo evidenza oggettiva a seguito di analisi del prodotto che mostri come i requisiti siano implementati dal software in maniera consistente. La validazione non si applica ad un particolare segmento temporale ma è una conferma finale, una self-fulfilling prophecy.

Attività anch'essa di supporto la quale accerta che il prodotto dei processi rispetti le specifiche. Il focus è rivolto sui prodotti finali. La verifica è l'ultima cosa da fare, *verifica finale*.

Qualifica $=$ verifica $+$ validazione. Nota anche come V&V

Unità è la più piccola unità di SW testabile e degna di essere testata. E' producibile da un singolo programmatore. Si può pensare all'unità come un *singolo pezzo*. I linguaggi di programmazione non hanno una keyword per denotare un'unità, poichè sta a chi valida determinare la dimensione dell'unità (è dunque un concetto *astratto*).E' possibile comunque che un'unità corrisponda ad una classe, ma questo $\Leftrightarrow$ la classe è *coesa* (vedi definizione)

Modulo parte dell'entità, è inteso con il classico significato di *modulo*, da internet: "*a module consists a single block of code that can be invoked in the way that a procedure, function, or method is invoked*". Dalle slides: componente elementare di architettura di dettaglio

Componente entità che integra più *unità*

Integrazione $\int$ (di unità) è l'atto di unione di due unità. Ha un relativo *test di integrazione*

Driver component SW fittizia per pilotare l'esecuzione di test di unità: esse non avendo il main non hanno un chiamante e dunque il driver simula la chiamata. E' usa e getta

Stub (ceppo) componente fittizia che simula una parte sottostante al sistema che si sta testando. Esempio: sto testando un sito web ma non ho ancora il database vero e proprio SQL, dunque preparo un componente che lo simuli per eseguire i test.

Logger componente non intrusivo di registrazione dei dati di esecuzione per analisi dei risultati. Il programma deve dunque scrivere in modo automatico e *persistente* i risultati su un file analizzabile in un secondo momento.

Fault (colpa) manifestazione concreta di un errore umano. E' la causa concettuale ed algoritmica degli errori.

Failure (fallimento) il risultato del fault. Accade quando il sistema fa qualcosa di insapettato.

Error quantificazione di quanto scorretto è il risultato. Gli errori sono uno stato particolare del sistema: si può pensarli come una mera sequenza di 0 e di 1, ma causati da un *fault*

$$\text{La sequenza è Mistake} \rightarrow \text{Fault} \rightarrow \text{Error} \rightarrow \text{Failure}$$

Test funzionale (Black box) è il test che si esegue senza vedere che sta succedendo veramente all'interno del programma. Dati gli input l'unica cosa che interessa veramente è avere degli output corretti

Test strutturale (White box) è il test che verifica la struttura interna dell'*unità*, cercando la massima *copertura*. Ciò vuol dire che si cerca di eseguire ogni singolo branch e istruzione possibile nell'unità

Condizione Espressione booleana semplice non contenente ulteriori condizioni combinate da operatori booleani

Decisione Espressione composta contenente condizioni combinate da operatori booleani

Esercitazione

Tecnology Baseline rappresenta la *baseline* delle tecnologie scelte (fra molte) per implementare un progetto. Ogni scelta deve essere motivata. Deve spiegare come si integrano tecnologie (librerie con framework per esempio) e prototipi

POC (Proof of concept) utilizzo delle tecnologie selezionate nella *technology baseline* per la realizzazione di un punto critico all'interno del progetto. Da qui proof (prova) of concept (di qualcosa di astratto). Fa parte della technology baseline

Product Baseline successiva temporalemente alla technology baseline, è la baseline di prodotto, presentazione documentale (e non dal proof of concept) del prodotto, solamente nel momento in cui ho un'*architettura* consolidata e realizzata. Da notare come ciò sia diverso dal prodotto completo: ci siamo vicini ma non siamo ancora arrivati

SOLID

L'acronimo SOLID

1. **Single responsability principle**: una classe dovrebbe avere un solo asse di cambiamento
2. **Open-close principle**: il codice deve essere aperto all'estensione ma chiuso alla modifica. **Significa** variabili private, niente downcasting o RTTI e niente variabili globali

3. **Liskov substitution principle**: funzioni che usano puntatori a classi base devono essere capaci di utilizzare sottoclassi senza saperlo. **Significa** utilizza la programmazione con `contratti`: con pre e post condizioni di ogni metodo. Le sottoclassi non possono fare assunzioni più forti sulle pre e nemmeno assunzioni più deboli sulle post.

4. **Interface segregation principle**: i client non dovrebbero essere dipendenti da interfacce che non usano. **Significa** usa adapter o ereditarietà multipla.

5. **Dependecy inversion principle**: i moduli di alto livello non dovrebbero dipendere da quelli di basso livello e viceversa. Entrambi dovrebbero dipendere da astrazioni. Le astrazioni non dovrebbero dipendere dai dettagli e i dettagli dovrebbero dipendere dalle astrazioni. **Significa** usa parecchia astrazione, decisioni di design importanti nelle astrazioni e usa il template method