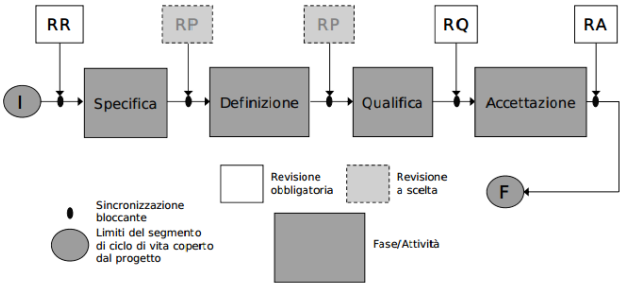
Considerazioni strategiche(11)

Il segmento di ciclo di vita attivato nel progetto didattico va dalla RR *Revisione dei Requisiti* alla RA *Revisione Accettazione*.
Per il committente non è positivo avere un modello sequenziale, perché non avrà mai visione del prodotto se non alla fine, non riceve prototipi e quindi perde interesse nel progetto, e noi vogliamo un proponente che sia **attivo** e interessato al nostro prodotto. Una volta acquisita l'esperienza necessaria allora posso usare un modello un po' più agile. Il **modello di sviluppo interno** serve a noi per trovare il miglior compromesso tra esigenze diverse. E' una scelta autonoma del fornitore e determina il piano strategico di utilizzo delle risorse disponibili (persone, capacità , strumenti).

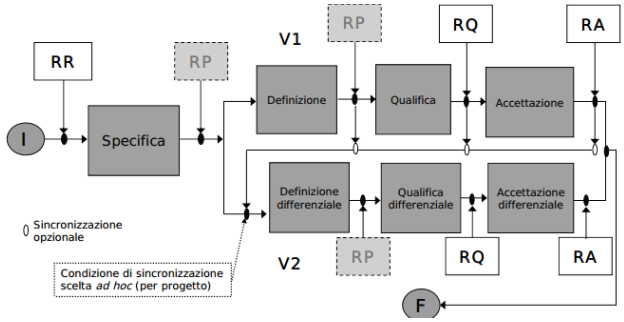
Ciascuno di noi deve formare un gruppo portando un proprio **calendario** già fatto, fissando dei vincoli e delle previsioni strategiche. Bisogna far emergere una disciplina ed avere alcune accortezze. Si parla dunque di **pianficazione**. Un *mese-persona* vale circa 142 ore. Il modo in cui gestisco tempo e persone è molto delicato:

- Vi sono componenti di impegno **non comprimibili**, ovvero non posso mettere più persone sullo stesso compito, non posso svolgerlo in parallelo. Non posso frantumare in piccole parti da parallelizzare (es. programmazione o verifica). Le cose sulle quali riesco a comprimere sono poche;
- Vi sono compiti **non partizionabili**;
- La verifica a livello di sistema si fa **solo alla fine**, perché non sono test parallelizzabili e il sistema diventa disponibile solo alla fine dello sviluppo.

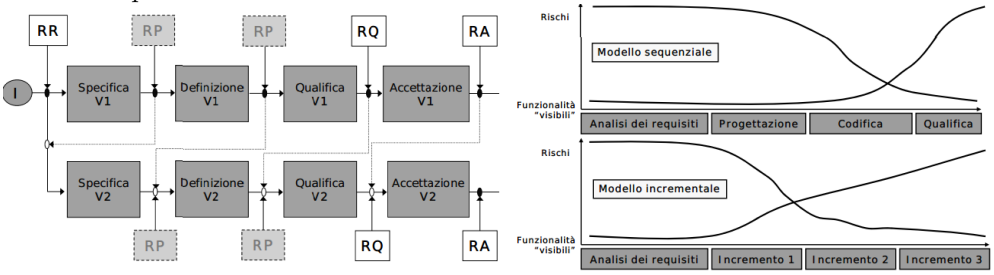
Occorre avere una pianificazione che abbia margini e che sia completamente consapevole dei vincoli. Una buona progettazione consente di non cadere nella iterazione non controllata. In questo modo, con queste tecniche si migliora la *mitigazione dei rischi*.
Che tipo di modello di ciclo di vita devo scegliere? Dobbiamo scegliere una strategia che sia buona per noi. Il modello sequenziale è esattamente coerente con quello che si aspetta il committente, ma non per quanto riguarda il proponente.



Il **modello incrementale** non è iterativo, l'iterazione *distrugge* per sostituzione, potenzialmente pericolosa. La posso avere solo in situazioni di emergenza. Un ciclo incrementale può essere visto come un "*for*". Ogni passaggio di questo ciclo aggiunge cose, mi avvicino alla soluzione per approssimazioni non distruttive, quindi additive. Posso anche non incrementare ma l'importante è che il numero di iterazioni sia noto. Dentro una chiamata di RP posso portare più di un incremento, tuttavia il progetto didattico prevede una singola occorrenza di ogni revisione. Il fornitore deve realizzare altre verifiche interne senza il coinvolgimento del committente.



Modello evolutivo. E' un modello che approssima la soluzione finale ammettendo tante iterazioni, abortendo le versioni intermedie. Per poter attuare un modello evolutivo ho bisogno di tanta energia. E' una tecnica molto interessante ma con un enorme costo, infatti revisioni successive possono avere come oggetto versioni di prodotto diverse.



Con un modello sequenziale combatto per abbattere i rischi mandando molto avanti le funzionalità visibili, l'avanzamento infatti viene fatto solamente quando si è sicuri.
Con un modello incrementale riesco ad avere delle funzionalità molto prima, tuttavia ho molti più rischi possibili.
Il **modello agile** non è facilmente rappresentabile. Si ragiona sulle cose da fare (**backlog**). Fra le cose da fare in un modello agile le persone prendono liberamente quello che faranno (ciascuno pesca un post-it a seconda del proprio estro). In un modello agile l'essenziale è che per ogni cosa fatta l'effetto sia visibile (**incremental build**). Ogni aggiunta rende il prodotto sempre più vicino alle aspettative, anche se non ha un ordine particolarmente ovvio. L'unico ordine è che ci siano tante cose che posso vedere e che rappresentano ciò che il prodotto sarà . E' un modello molto interessante ma difficile da gestire.
Non tutti i problemi hanno una (buona) soluzione. Bisogna fissare con la massima chiarezza:

- **Obiettivi**;
- **Vincoli**;
- **Alternative**;
- **Rappresentazione del problema e delle sue soluzioni**.

la qualità cardine resta comunque **Fattibilità** e **Verificabilità**
Tecniche progettuali:

- **Decomporre** in modo modulare e senza dipendenze. Una buona decomposizione identifica componenti tra loro indipendenti (a basso accoppiamento e funzionalmente coesi). La seconda cosa che voglio fare è nascondere il dettaglio implementativo (**incapsulamento**).
- **Accoppiamento:** è la misura dell'intensità della relazione tra parti distinte.
- **Coesione:** è la misura dell'intensità della relazione all'interno di una singola parte.

- **Astrazione:** omettere informazione per poter applicare operazioni simili ad entità diverse. Non importa la forma esatta che ha una cosa ma che abbia informazioni e funzionalità utili, non voglio sapere tutto, ma le cose importanti in un determinato contesto.
- **Atomicità:** è un altro criterio molto importante. L'utilità dell'astrazione non migliora se divido ulteriormente, perché scomporre più del dovuto ha un costo.
- **Concorrenza:** è molto importante garantire al sistema una concorrenza, ma va ben gestita e se la uso in modo inconsapevole faccio solo danni.
- **Distribuzione:** se e come i componenti sono disseminati su più nodi di elaborazione e come comunicano tra di loro.