

## → Secondo Blocco

Per realizzare un prodotto software vengono svolte una serie di attività in un determinato tempo, con determinati costi e con determinate caratteristiche, tutto ciò viene descritto con il termine Processo Software.

Questo processo prevede l'uso di metodi, tecniche e strumenti, la creazione di prodotti intermedi e finali, il controllo gestionale, la garanzia della qualità e la gestione delle modifiche.

Precedentemente abbiamo parlato di come il ciclo di vita di un Prodotto Software si pianifica in tre stadi.

Nel primo stadio si possono riconoscere due tipi di fasi:

- Fasi di tipo definizione, si occupa di “cosa” il software deve fornire
- Fasi di tipo produzione, definisce il “come” realizzare quanto ottenuto con le fasi di definizione. Quindi si progetta il software, si codifica, si integra e si rilascia al cliente

Il secondo stadio, ovvero quello di Manutenzione, manutenzione è a supporto del software realizzato e prevede fasi di definizione e/o produzione al suo interno.

Durante ogni fase si procede ad effettuare il testing di quanto prodotto, attraverso tecniche di verifica e validazione (V&V) applicate sia ai prodotti intermedi che al prodotto finale.

Esistono 5 tipologie di Manutenzione:

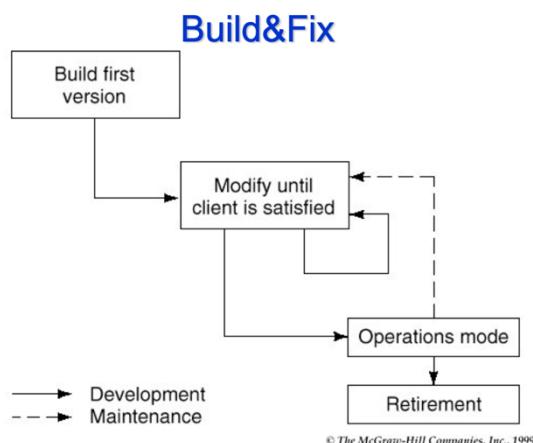
- Manutenzione Correttiva: che ha lo scopo di eliminare i difetti (fault) che producono guasti (failure) del software
- Manutenzione Adattativa: che ha lo scopo di adattare il software ad eventuali cambiamenti a cui è sottoposto l'ambiente operativo per cui è stato sviluppato
- Manutenzione Perfettiva: che ha lo scopo di estendere il software per accomodare funzionalità aggiuntive
- Manutenzione Preventiva (o software reengineering): che consiste nell'effettuare modifiche che rendano più semplici le correzioni, gli adattamenti e le migliorie

Definizione di Ciclo di Vita secondo lo IEEE Std 610-12:

- Intervallo di tempo che va tra l'istante in cui nasce l'esigenza di costruire un prodotto software e l'istante in cui il prodotto viene dismesso.

- Include le fasi di definizione dei requisiti, specifica, pianificazione, progetto preliminare, progetto dettagliato, codifica, integrazione, testing, uso, manutenzione e dismissione. Queste fasi possono essere eseguite in sequenza, sovrapposte o ripetute più volte

Il modello del ciclo di vita del software specifica la serie di fasi attraverso cui il prodotto software progredisce e l'ordine con cui vanno eseguite, dalla definizione dei requisiti alla dismissione. La scelta del modello dipende dalla natura dell'applicazione, dalla maturità dell'organizzazione, da metodi e tecnologie usate e da eventuali vincoli dettati dal cliente. L'assenza di un modello del ciclo di vita corrisponde ad una modalità di sviluppo detta “build & fix” (o “fix-it-later”), in cui il prodotto software viene sviluppato e successivamente rilavorato fino a soddisfare le necessità del cliente.



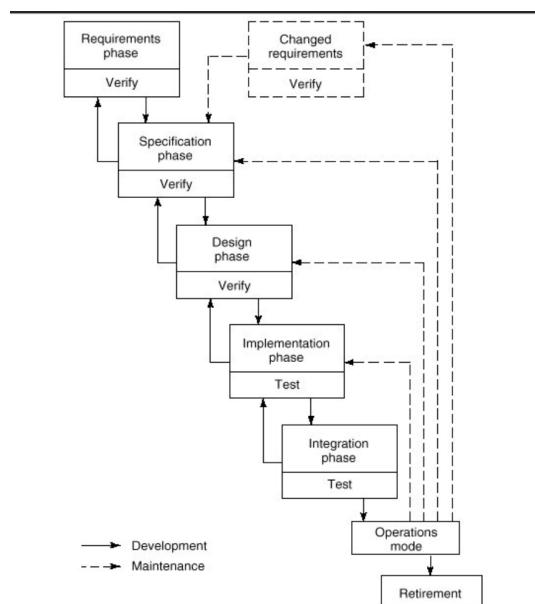
- **Lezione 4 – ( 21/10/2024 )**

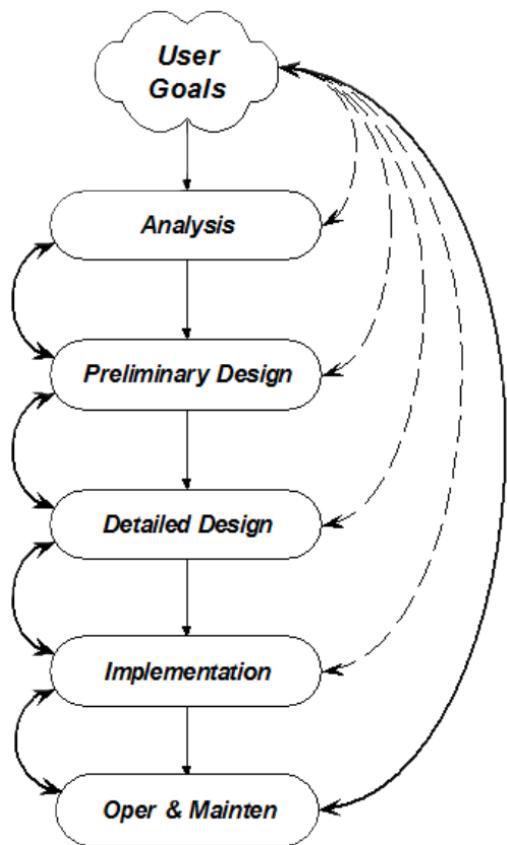
Il modello Modello Waterfall o Modello a Cascata, che si distingue nettamente dal modello Build & Fix. Viene chiamato "a cascata" poiché le fasi del processo sono presentate in sequenza, come in una cascata. Questo modello può essere considerato un punto di riferimento rispetto al quale vengono confrontati altri modelli di sviluppo software.

Nel Modello Waterfall, ogni fase non può essere considerata completata senza una verifica dei requisiti e della correttezza di quanto realizzato. In ogni rettangolo che rappresenta una fase specifica del ciclo di vita, è presente una riga con l'indicazione "Test" o "Verifica". Questo si deve al fatto che:

- Verify indica una verifica statica del lavoro svolto (controllo della correttezza, coerenza dei documenti e della progettazione),
- Test indica una verifica dinamica (esecuzione pratica del software per rilevare errori funzionali).

Un aspetto critico del Modello Waterfall è che i requisiti di un prodotto software tendono a cambiare continuamente durante il processo di sviluppo. Per questo motivo, il modello include meccanismi per il ritorno a fasi precedenti o l'integrazione di cambiamenti nei requisiti, come indicato dalle linee tratteggiate nel diagramma.





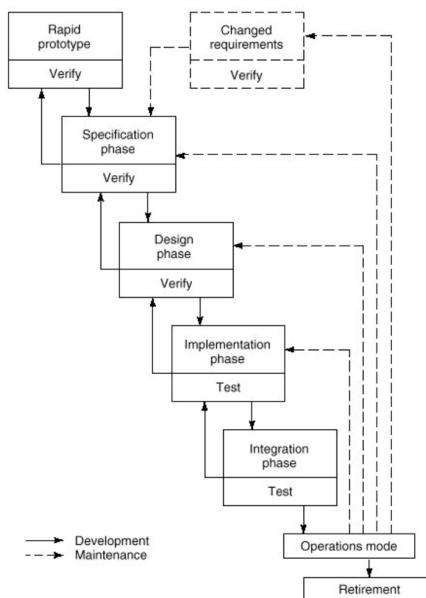
La figura illustra la differenza tra verifica e validazione nel processo di sviluppo del software.

- La verifica consiste nel controllare che un prodotto software sia stato sviluppato correttamente, ovvero che soddisfi i requisiti tecnici e di progetto.

Ad esempio: quando si passa un documento in input, si verifica che sia corretto rispetto alle specifiche, e solo in caso positivo si procede con la fase successiva.

- La validazione, invece, è un processo diverso, poiché ci si assicura che il prodotto finale soddisfi i bisogni e le aspettative dell'utente. Anche se un prodotto software può essere corretto dal punto di vista implementativo (cioè, ha superato la verifica), potrebbe non risultare utile o valido per l'utente che lo utilizzerà.

Nel corso degli anni, si è cercato di migliorare il modello a cascata (Waterfall). Uno dei primi miglioramenti fu l'introduzione del modello Rapid Prototyping.



In questo modello, nella fase di raccolta dei requisiti viene utilizzato un prototipo rapido, un prodotto che contiene principalmente l'interfaccia utente, ma che non esegue ancora tutte le funzionalità del sistema finale. Questo prototipo viene impiegato per aiutare l'ingegnere del

software a comprendere meglio i requisiti e a supportare le attività di sviluppo, consentendo di ottenere un feedback iniziale dagli utenti.

Il Prototyping Software ( o **Prototipazione del software** ) è una metodologia di sviluppo rapido che viene utilizzata principalmente per raccogliere e validare i requisiti di un software.

In pratica, si tratta di creare versioni preliminari del sistema, chiamate prototipi, che permettono a clienti e sviluppatori di avere un'idea concreta di come funzionerà il prodotto finale. I prototipi svolgono un ruolo fondamentale nel migliorare la comprensione reciproca tra chi utilizza il software e chi lo sviluppa.

Ci sono due usi principali dei prototipi nel processo di sviluppo software:

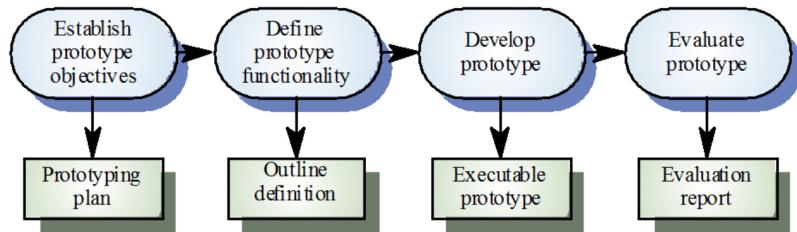
- *Elicitazione dei requisiti (o Requisiti dei rischi)*: questo è il processo in cui i requisiti vengono scoperti o definiti. Un prototipo permette agli utenti di sperimentare una versione preliminare del sistema, aiutandoli a capire se e come il software supporterà il loro lavoro. Questo è particolarmente utile per chiarire aspetti che potrebbero non essere subito evidenti.
- *Validazione dei requisiti (o Convalida dei Requisiti)*: una volta che i requisiti sono stati definiti, il prototipo viene utilizzato per verificare se il sistema risponde a quanto richiesto. In questa fase, si possono identificare errori o mancanze che potrebbero essere sfuggiti durante la definizione iniziale.

In sostanza, il *prototyping* serve a ridurre i rischi legati ai requisiti, rendendo il processo di sviluppo più sicuro e privo di fraintendimenti.

Il prototyping offre diversi vantaggi che lo rendono una pratica diffusa nello sviluppo software:

- Riduce fraintendimenti tra utenti e sviluppatori, migliorando la comunicazione e chiarendo meglio le aspettative.
- Individua servizi mancanti o confusi, ovvero funzionalità che magari non erano state previste inizialmente.
- Fornisce un sistema funzionante già nelle prime fasi dello sviluppo, il che aiuta a visualizzare concretamente come sarà il prodotto finale.
- Deriva la specifica software dal prototipo: il prototipo può essere usato come punto di partenza per scrivere la documentazione ufficiale del progetto
- Supporta la formazione degli utenti e i test del prodotto: gli utenti possono essere formati utilizzando il prototipo, e lo stesso può essere impiegato per eseguire test reali sul sistema (un processo chiamato "back-to-back").

Il processo di prototipazione software si articola in 4 fasi principali:

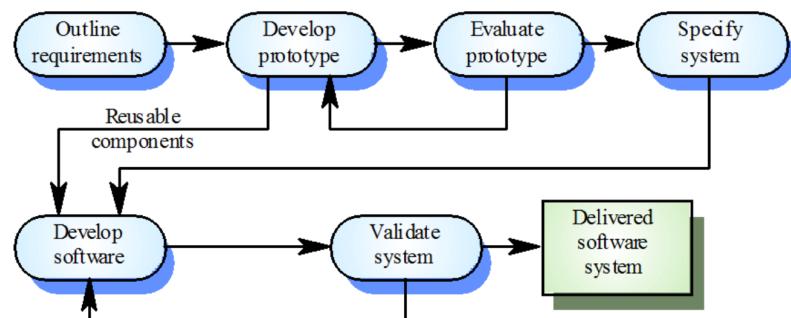


1. *Stabilire gli obiettivi del prototipo*: qui si definiscono le finalità del prototipo. Lo scopo potrebbe essere, ad esempio, verificare che una certa funzione risponda alle aspettative dell'utente. Questa fase culmina con la creazione di un piano di prototipazione.
2. *Definire la funzionalità del prototipo*: si stabiliscono le caratteristiche che il prototipo deve avere. Non tutte le funzionalità del sistema finale sono incluse, ma solo quelle più importanti o critiche per comprendere come funzionerà il software.
3. *Sviluppare il prototipo*: questa è la fase di costruzione vera e propria, in cui si realizza un prototipo eseguibile, basato sulle funzionalità precedentemente definite.
4. *Valutare il prototipo*: una volta sviluppato, il prototipo viene testato per verificarne l'efficacia. In questa fase si raccolgono feedback dagli utenti e si individua cosa funziona e cosa necessita di miglioramenti.

Prototipi come specifiche:

A volte i prototipi vengono utilizzati per scrivere parte della documentazione di progetto, ma ci sono alcune limitazioni. Ad esempio, le funzionalità critiche per la sicurezza non sono sempre facili da rappresentare in un prototipo, e i requisiti non funzionali (come prestazioni o sicurezza) non possono essere testati efficacemente con un prototipo.  
Un'implementazione non ha valore legale come contratto.

Il prototipo usa e getta (*Throw-away*)



Il prototipo "usa e getta" è una versione preliminare del sistema che viene creata per identificare problemi nei requisiti e poi scartata una volta ottenuti i feedback necessari. Questo tipo di prototipo non è pensato per essere mantenuto o utilizzato come prodotto finale, perché:

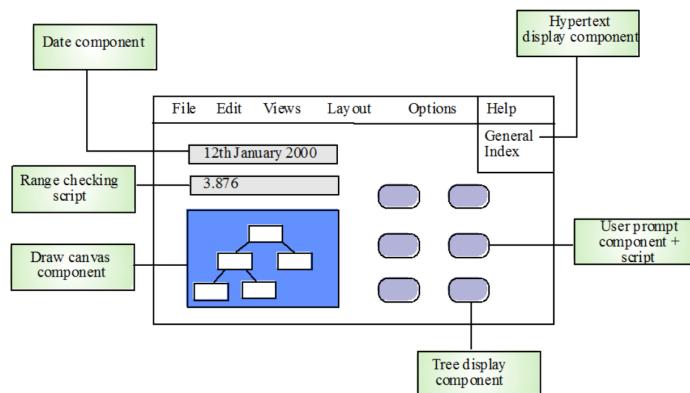
- Alcune caratteristiche potrebbero essere state omesse
- Non è pensato per la manutenzione a lungo termine, quindi non ha una struttura ottimizzata per essere ampliata o modificata nel tempo
- La struttura del prototipo è generalmente difficile da mantenere

Distribuire un prototipo come prodotto finale è sconsigliato perché potrebbe non rispettare i requisiti non-funzionali, mancare di documentazione, avere una struttura compromessa dai cambiamenti e non seguire gli standard di qualità previsti.

Alcuni punti fondamentali da ricordare sulla prototipazione sono:

- I prototipi danno agli utenti un'impressione concreta delle capacità del prodotto, aiutandoli a visualizzare come sarà.
- Il prototyping è sempre più utilizzato quando è necessario sviluppare un prodotto rapidamente.
- I prototipi usa e getta vengono creati per comprendere meglio i requisiti, ma non dovrebbero essere utilizzati come base per il prodotto finale.
- Lo sviluppo rapido dei prototipi è essenziale, anche se ciò significa dover omettere alcune funzionalità o allentare i vincoli non funzionali.
- La programmazione “visuale” è uno strumento comune nel prototyping, in cui l’interfaccia utente viene costruita utilizzando componenti standard, come avviene in linguaggi come VisualBasic.

#### → Problemi dello sviluppo visuale



Nonostante i vantaggi della programmazione visuale, essa presenta alcuni problemi, come:

- Difficoltà nel coordinare il lavoro di un team: quando più persone lavorano insieme, può essere difficile mantenere tutto sotto controllo.

- Mancanza di un'architettura software esplicita, che rende complesso gestire lo sviluppo a lungo termine.
- Dipendenze complesse tra le parti del programma che possono compromettere la manutenibilità del sistema, rendendo difficili le modifiche future.

Riassumendo possiamo dire che, il prototyping è uno strumento utile per sviluppare software in modo rapido ed efficiente, specialmente nelle fasi iniziali di raccolta e validazione dei requisiti, ma è importante essere consapevoli delle sue limitazioni e dei rischi associati, soprattutto quando si tratta di prototipi "usa e getta".

- Processo di iterazione:

L'iterazione nei processi di sviluppo è essenziale, poiché i requisiti di un progetto evolvono continuamente, specialmente nei prodotti complessi. Questa iterazione può essere applicata a qualsiasi modello di sviluppo e comprende approcci come lo *sviluppo incrementale* e lo *sviluppo a spirale*.

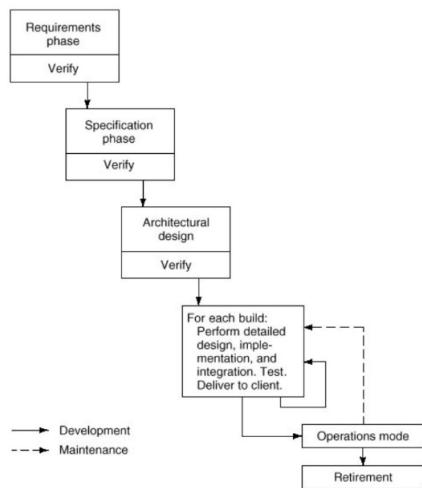
1. *Sviluppo incrementale*:

- Il prodotto è sviluppato e lanciato con incrementi dopo aver stabilito un'architettura generale;
- I requisiti e le specifiche per ogni aggiunta possono essere sviluppati;
- Gli utenti possono sperimentare con gli incrementi (le aggiunte) consegnati mentre gli altri vengono sviluppati. Pertanto questo serve come una forma di prototipo;
- Prevede di unire alcuni dei vantaggi della prototipazione ma con un processo maggiormente gestibile ed una struttura migliore;

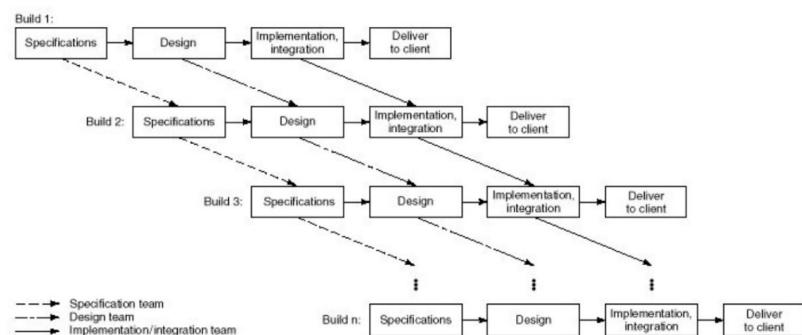
2. *Modello incrementale*:

- Il prodotto software viene sviluppato e rilasciato per incrementi (build) successivi;
- Include aspetti tipici del modello basato su rapid prototyping (l'utente può sperimentare l'utilizzo del prodotto contenente gli incrementi consegnati, mentre i restanti sono ancora in fase di sviluppo);
- Si rivela efficace quando il cliente vuole continuamente verificare i progressi nello sviluppo del prodotto e quando i requisiti subiscono modifiche;
- Può essere realizzato in due versioni alternative:

- Versione con overall architecture;



- Versione senza overall architecture (più rischiosa).



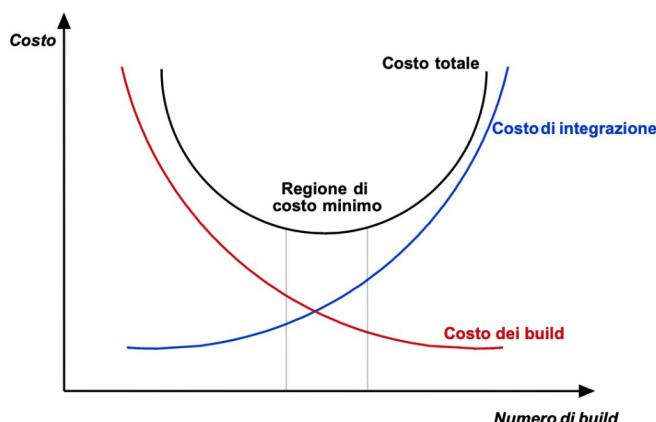
- Lezione 5 – ( 24/10/2024 )

Quando si fa una scelta per la produzione di un Prodotto Software possiamo valutare quanto impatto ha questa scelta sui costi.

Andremo a rappresentarlo attraverso un grafico dove inseriremo il costo di sviluppo (include tutte le risorse per realizzare il prodotto) dell'intero prodotto sull'asse delle ordinate, invece, rappresenteremo il numero di build sull'asse delle ascisse.

Se partiziono il prodotto in un piccolo numero di build il costo di integrazione sarà minimo. Invece, se lo partiziono in un grande numero di build il costo di integrazione crescerà. Quindi il costo di integrazione cresce al crescere del costo dei build.

Questi due costi saranno descritti dal Costo totale. Questo avrà una regione di minimo costo



Confronto con modello a cascata

#### *Modello a cascata*

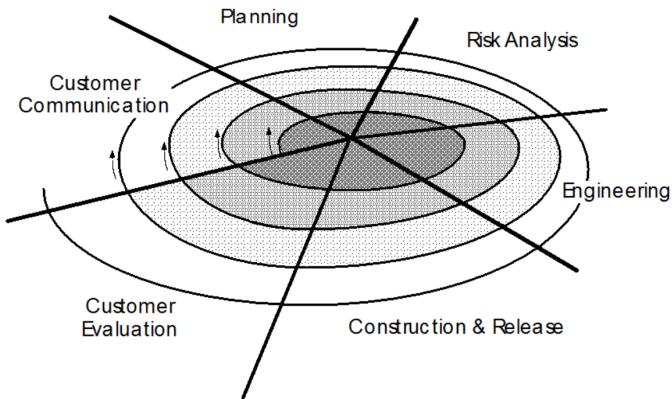
- Requisiti “congelati” al termine della fase di specifica
- Feedback del cliente solo una volta terminato lo sviluppo
- Fasi condotte in rigida sequenza (l’output di una costituisce input per la successiva)
- Prevede fasi di progetto dettagliato e codifica dell’intero prodotto
- Team di sviluppo costituito da un numero elevato di persone

#### *Modello incrementale*

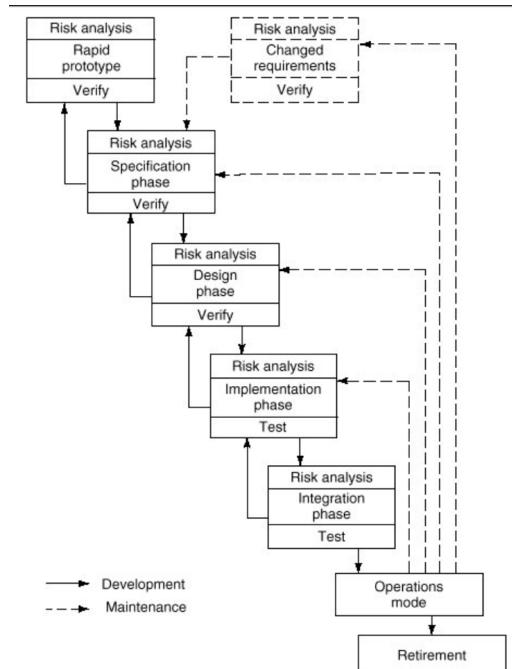
- Requisiti suddivisi in classi di priorità e facilmente modificabili
- Continuo feedback da parte del cliente durante lo sviluppo
- Fasi che possono essere condotte in parallelo
- Progetto dettagliato e codifica vengono effettuate sul singolo *build*
- Differenti team di sviluppo, ciascuno di piccole dimensioni

L'approccio incrementale è l'approccio conveniente fra i due modelli visti.

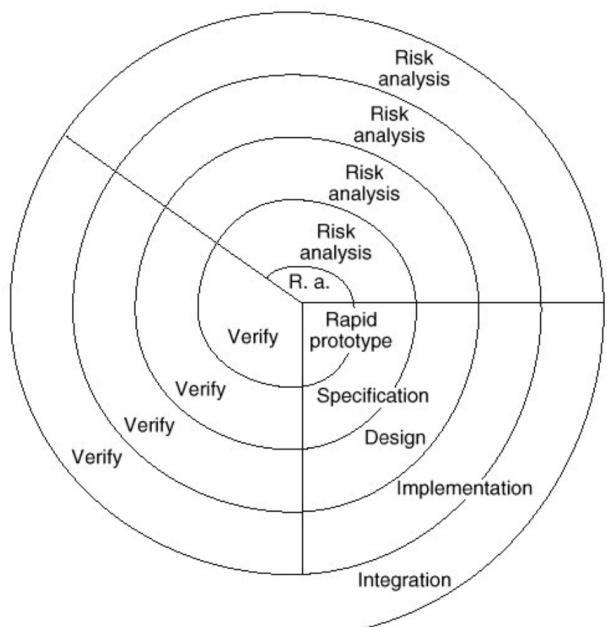
Un approccio sempre a carattere iterativo è il *Modello a Spirale*.



Si tratta di svolgere le attività percorrendo dalla parte più interna verso l'esterno. In questa spirale la dimensione radiale rappresenta l'implemento dei costi e la dimensione angolare rappresenta la variazione del tempo.



*Modello a spirale semplificato  
(versione lineare)*



*Modello a spirale semplificato*

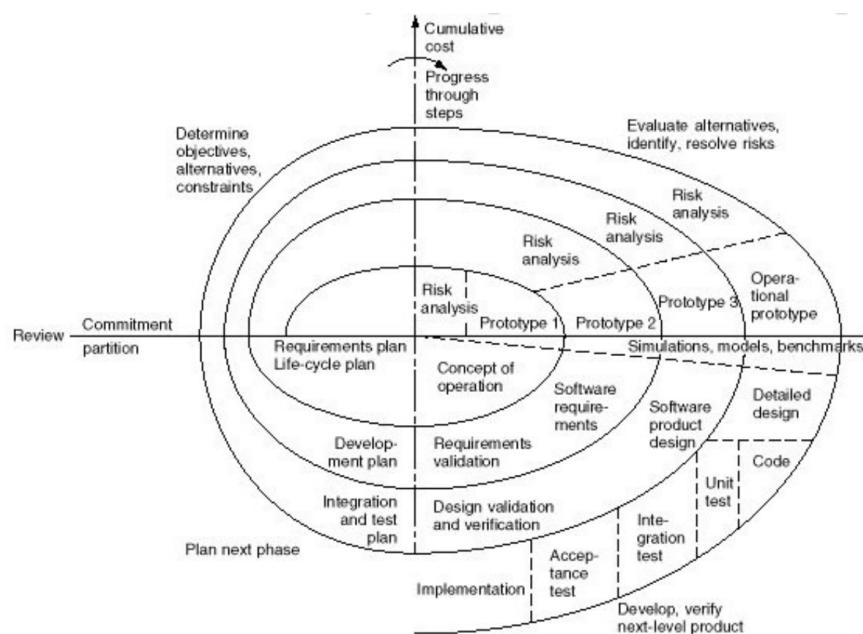
Il *Modello a spirale semplificato* (versione lineare) ci dice che quando entro in una fase dopo aver completato e verificato quella precedente, prima di entrare devo attuare un'attività di *Risk Analysis* (ovvero *Analisi del Rischio*).

Quindi nel mio team di sviluppo devo avere delle persone che mi verificano l'*Analisi del Rischio* e mi devono dire se il rischio è affrontabile o se sono talmente elevati che non posso effettuare la produzione del progetto software. Questo modello sembra rischioso, invece, si è dimostrato altamente efficace.

Un approccio di questo tipo, che si è rivelato altamente produttivo, si può applicare a tutti i tipi di software?

Immaginiamo la situazione di un software a contratto, dove il cliente paga lo sviluppo del software e ad un mese dallo sviluppo del software lo sviluppatore dice che a causa dei rischi il progetto cessa di nascere. Ovviamente ci sta un contratto che lega queste due entità, però in caso contrario ci sta una problematica. Infatti questo approccio è più efficace nel caso di software interni.

Successivamente nel 1988 Boehm, pubblica un nuovo modello chiamato *Modello full – spiral*, con le relative sperimentazioni che l'hanno portato a ciò.



#### ▪ Gestione del rischi

La gestione del rischio si occupa di identificare i rischi e ideare piani per minimizzare il loro effetto su un progetto. Un rischio è la probabilità che alcune circostanze avverse si verifichino. Esistono varie categorie di rischio :

- I rischi connessi al progetto influenzano il programma (schedule) o le risorse
- I rischi connessi al prodotto influenzano la qualità o la performance del software che si sta sviluppando

- I rischi aziendali influenzano lo sviluppo dell'organizzazione o il reperimento del software

Esempio di Rischi per CATEGORIA:

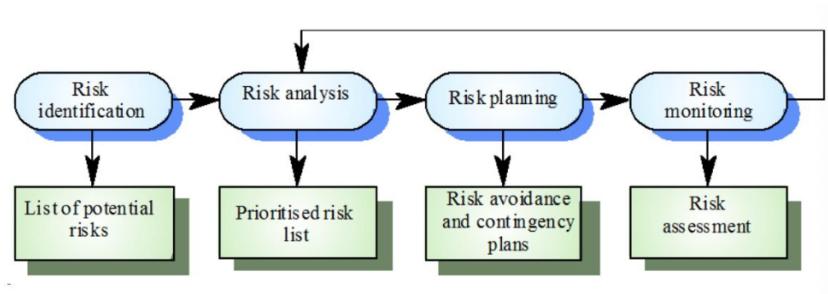
### Risks by category

Risk	Risk type	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware which is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

Rischio	Tipo di rischio	Descrizione
Turnover del personale	Progetto	Il personale esperto lascerà il progetto prima che sia completato.
Cambio di gestione	Progetto	Ci sarà un cambiamento nella gestione organizzativa con priorità differenti.
Indisponibilità hardware	Progetto	L'hardware essenziale per il progetto non sarà consegnato nei tempi previsti.
Cambio dei requisiti	Progetto e prodotto	Ci sarà un numero maggiore di cambiamenti ai requisiti rispetto al previsto.
Ritardi nelle specifiche	Progetto e prodotto	Le specifiche delle interfacce essenziali non saranno disponibili in tempo.
Sottostima delle dimensioni	Progetto e prodotto	La dimensione del sistema è stata sottostimata.
Sottoperformance del CASE	Prodotto	Gli strumenti CASE che supportano il progetto non funzionano come previsto.
Cambiamento tecnologico	Business	La tecnologia su cui si basa il sistema è superata da una nuova tecnologia.
Competizione del prodotto	Business	Un prodotto concorrente viene commercializzato prima che il sistema sia completato.

Processo di gestione del rischio:

- Identificazione del rischio:* Identificazione dei rischi connessi al progetto, al prodotto e dei rischi aziendali
- Analisi del rischio:* Stima della probabilità e delle conseguenze di questi rischi
- Pianificazione del rischio:* Redigere piani per evitare o minimizzare gli effetti del rischio
- Monitoraggio del rischio:* Controllo del rischio per tutto il progetto



Identificazione del Rischio (o tipo di Rischio):

- Rischi tecnologici
- Rischi per le persone
- Rischi organizzativi

- Rischi degli strumenti
- Rischi per i requisiti
- Rischi di stima

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components which should be reused contain defects which limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements which require major design rework are proposed. Customers fail to understand the impact of requirements changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

Tipo di rischio	Rischi possibili
Tecnologia	Il database utilizzato nel sistema non può processare tante transazioni al secondo come previsto.
	I componenti software che dovrebbero essere riutilizzati contengono difetti che ne limitano la funzionalità.
Persone	È impossibile reclutare personale con le competenze richieste.
	Il personale chiave è malato e non disponibile in momenti critici.
	La formazione richiesta per il personale non è disponibile.
Organizzativo	L'organizzazione viene ristrutturata in modo che una gestione diversa sia responsabile del progetto.
	I problemi finanziari organizzativi forzano riduzioni nel budget del progetto.
Strumenti	Il codice generato dagli strumenti CASE è inefficiente.
	Gli strumenti CASE non possono essere integrati.
Requisiti	Vengono proposti cambiamenti ai requisiti che richiedono una revisione importante del design.
	I clienti non comprendono l'impatto dei cambiamenti nei requisiti.
Stime	Il tempo richiesto per sviluppare il software è sottostimato.
	Il tasso di riparazione dei difetti è sottostimato.
	La dimensione del software è ostimata.

#### ■ Analisi del rischio

L'analisi del rischio consiste nella valutazione di due aspetti principali: la probabilità che il rischio si verifichi e la gravità dei suoi effetti sul progetto o prodotto. La probabilità di un rischio può essere classificata in cinque livelli:

- Molto bassa: meno del 10%
- Bassa: tra il 10% e il 25%
- Moderata: tra il 25% e il 50%
- Alta: tra il 50% e il 75%
- Molto alta: oltre il 75%

Gli effetti di un rischio, invece, possono essere:

- Catastrofici
- Gravi
- Tollerabili
- Insignificanti

Dopo aver valutato la probabilità e la gravità, bisogna identificare i "top-ten" rischi, cioè i dieci rischi principali. Questi includono tutti i rischi con effetti catastrofici e quelli con effetti gravi che hanno più di una probabilità moderata di verificarsi. Questi rischi vengono poi classificati per ordine di importanza, in base alla combinazione di probabilità e gravità.

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget.	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components which should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements which require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious
The database used in the system cannot process as many transactions per second as expected.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
CASE tools cannot be integrated.	High	Tolerable
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
Required training for staff is not available.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable
The size of the software is underestimated.	High	Tolerable
The code generated by CASE tools is inefficient.	Moderate	Insignificant

Rischio	Probabilità	Effetti
Problemi finanziari organizzativi forzano riduzioni nel budget del progetto.	Bassa	Catastrofico
È impossibile reclutare personale con le competenze richieste per il progetto.	Alta	Catastrofico
Il personale chiave è malato in momenti critici del progetto.	Moderata	Grave
I componenti software che dovrebbero essere riutilizzati contengono difetti che ne limitano la funzionalità.	Moderata	Grave
Vengono proposti cambiamenti ai requisiti che richiedono una revisione importante del design.	Moderata	Grave
L'organizzazione viene ristrutturata in modo che una gestione diversa sia responsabile del progetto.	Alta	Grave
Il database utilizzato nel sistema non può processare tante transazioni al secondo come previsto.	Moderata	Grave
Il tempo richiesto per sviluppare il software è sottostimato.	Alta	Grave
Gli strumenti CASE non possono essere integrati.	Alta	Tollerabile
I clienti non comprendono l'impatto dei cambiamenti nei requisiti.	Moderata	Tollerabile
La formazione richiesta per il personale non è disponibile.	Moderata	Tollerabile
Il tasso di riparazione dei difetti è sottostimato.	Moderata	Tollerabile
La dimensione del software è sottostimata.	Alta	Tollerabile
Il codice generato dagli strumenti CASE è inefficiente.	Moderata	Irrilevante

## ■ Pianificazione del rischio

Una volta individuati i rischi, è necessario sviluppare strategie per gestirli. Ci sono tre tipi di strategie principali:

- Strategie di elusione: riducono la probabilità che il rischio si presenti.
- Strategie di minimizzazione: riducono l'impatto del rischio, se si dovesse verificare.
- Piani di emergenza: sono soluzioni alternative da attuare nel caso in cui il rischio si materializzi, per limitare i danni.

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.
Requirements changes	Derive traceability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying in components, investigate use of a program generator.

## ■ Monitoraggio del rischio

Il monitoraggio dei rischi è un processo continuo. Ogni rischio identificato deve essere valutato regolarmente per verificare se la sua probabilità o i suoi effetti sono cambiati nel tempo. Per fare ciò, si osservano i fattori di rischio, ovvero gli indicatori che possono segnalare se un rischio sta diventando più o meno probabile.

Se si nota un cambiamento significativo nei fattori di rischio o nei possibili effetti, bisogna tornare all'analisi del rischio per aggiornare la strategia. Infine, i principali rischi dovrebbero essere discussi regolarmente nei meeting di avanzamento della gestione del progetto, per assicurarsi che siano gestiti correttamente (*management progress*).

<b>Risk type</b>	<b>Potential indicators</b>
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team member, job availability
Organisational	organisational gossip, lack of action by senior management
Tools	reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	many requirements change requests, customer complaints
Estimation	failure to meet agreed schedule, failure to clear reported defects

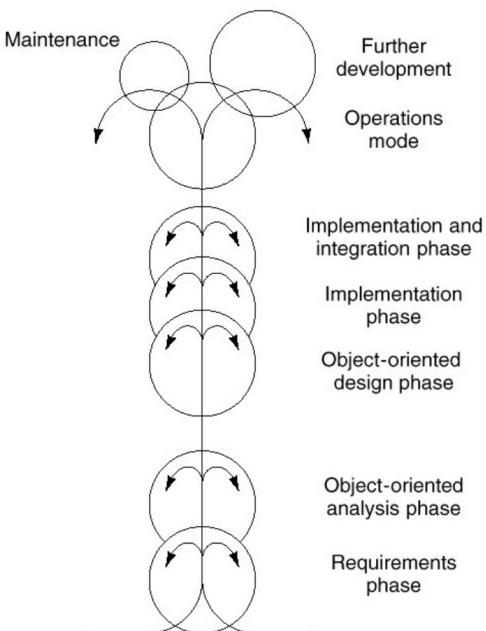
- Lezione 6 – ( 28/10/2024 )

Altri modelli che vengono usati sono:

- Modello object – oriented (o “Modello a Fontana”)

Questo modello è noto anche come "modello a fontana" a causa delle frecce che ricordano zampilli d'acqua. Si basa sull'idea di applicare i principi della programmazione orientata agli oggetti anche nelle fasi di analisi dei requisiti e progettazione del software. Si tratta di un modello altamente iterativo, in cui ogni cerchio rappresenta una fase specifica dello sviluppo. In particolare, il modello evidenzia che le fasi di specifica e progettazione devono essere affrontate con un approccio orientato agli oggetti. Tuttavia, una volta completata la fase di progettazione, non è strettamente necessario adottare una codifica orientata agli oggetti. Il modello prevede inoltre un'interazione sia all'interno di ciascuna fase (intrafase) che tra le fasi (interfase), con alcune forme di sovrapposizione tra le attività (cerchi sovrapposti nella figura).

Grazie a questo approccio, l'attività di manutenzione risulta ridotta rispetto al modello a cascata (Waterfall).



- Modello di ingegneria simultanea (o concorrente)

Ha come obiettivo la riduzione di tempi e costi di sviluppo, mediante un approccio sistematico al progetto integrato e concorrente di un prodotto software e del processo ad esso associato. Le fasi di sviluppo coesistono invece di essere eseguite in sequenza. Questo approccio è stato usato di più dalle case automobilistiche.

[ Ha accennato ricordi del suo lavoro e ha introdotto un approccio chiamato M.o.n.è. (non so come è scritto) ]

- Modello basato su metodi formali

Comprende una serie di attività che conducono alla specifica formale matematica del software, al fine di eliminare ambiguità, incompletezze ed inconsistenze e facilitare la verifica dei programmi mediante l'applicazione di tecniche matematiche.

La Cleanroom Software Engineering (1987) ne rappresenta un esempio di realizzazione, in cui viene enfatizzata la possibilità di rilevare i difetti del software in modo più tempestivo rispetto ai modelli tradizionali.

- Il Modello Microsoft

La Microsoft è un'azienda che sin dai tempi degli anni '80 ha affrontato, durante lo sviluppo di software commerciali, problemi di incremento della qualità dei prodotti software e di riduzione di tempi e costi.

Per risolvere ciò si è adottato un processo che è sia iterativo, sia incrementale sia concorrente, permettendo di esaltare doti di creatività delle persone che si occupano di sviluppare il Prodotto Software.

Per quanto riguarda le informazioni sull'approccio adottato da Microsoft per lo sviluppo dei propri software, non disponiamo di molte conoscenze, poiché l'azienda non ne ha mai parlato apertamente. Le uniche fonti che abbiamo a disposizione provengono da *M. A. Cusumano e R. W. Selby*, i quali hanno pubblicato un articolo intitolato "*How Microsoft Builds Software*" sulla rivista "*Communications of the ACM*", vol. 40, n. 6, nel giugno del 1997.

Attualmente la Microsoft usa l'approccio denominato "*Synchronize-and-Stabilize*". Questo approccio è basato su:

- sincronizzazione → si sincronizzano quotidianamente le attività svolte da persone che lavorano sia individualmente che all'interno di piccoli team (da 3 a 8 persone). I componenti software, anche se solo parzialmente sviluppati, vengono assemblati in un prodotto (daily build), che viene poi testato e corretto.
- stabilizzazione → si stabilisce periodicamente il prodotto in incrementi (milestone) successivi durante l'avanzamento del progetto, piuttosto che un'unica volta alla fine

Questo approccio ha un ciclo di sviluppo a 3 fasi:

- 1° FASE → *Fase progettuale/di pianificazione*: definisce la visione, la descrizione dettagliata e la tabella di marcia del prodotto;
- 2° FASE → *Fase di sviluppo*: presenta lo sviluppo in 3/4 sottoprogetti sequenziali ognuno derivante da una pubblicazione importante (milestone release);
- 3° FASE → *Fase di stabilizzazione*: test onnicomprensivi interni ed esterni, prodotto finale, consolidamento e invio;

### Fase di Pianificazione

Definisci la visione del prodotto, le specifiche e il programma

#### Planning Phase

Define product vision, specification, and schedule

- **Vision Statement** Product and program management use extensive customer input to identify and priority-order product features.
- **Specification Document** Based on vision statement, program management and development group define feature functionality, architectural issues, and component interdependencies.
- **Schedule and Feature Team Formation** Based on specification document, program management coordinates schedule and arranges feature teams that each contain approximately 1 program manager, 3–8 developers, and 3–8 testers (who work in parallel 1:1 with developers).

- **Dichiarazione di Visione:** Il management del prodotto e del programma utilizza un ampio input dei clienti per identificare e ordinare in base alla priorità le caratteristiche del prodotto.
- **Documento delle Specifiche:** Sulla base della dichiarazione di visione, il management del programma e il gruppo di sviluppo definiscono la funzionalità delle caratteristiche, le problematiche architettoniche e le interdipendenze dei componenti.
- **Pianificazione e Formazione dei Team di Funzionalità:** Basandosi sul documento delle specifiche, il management del programma coordina il programma e organizza i team di funzionalità, ognuno dei quali include circa 1 responsabile di programma, 3-8 sviluppatori e 3-8 tester (che lavorano in parallelo 1:1 con gli sviluppatori).

#### Development Phase

Feature development in 3 or 4 sequential subprojects that each results in a milestone release

Program managers coordinate evolution of specification. Developers design, code, and debug. Testers pair with developers for continuous testing.

- **Subproject I** First 1/3 of features (Most critical features and shared components)
- **Subproject II** Second 1/3 of features
- **Subproject III** Final 1/3 of features (Least critical features)

### Fase di Sviluppo

Sviluppo delle funzionalità in 3 o 4 sotto-progetti sequenziali, ognuno dei quali si conclude con una release milestone

I responsabili di programma coordinano l'evoluzione delle specifiche. Gli sviluppatori progettano, codificano e risolvono i bug. I tester collaborano con gli sviluppatori per eseguire test continui.

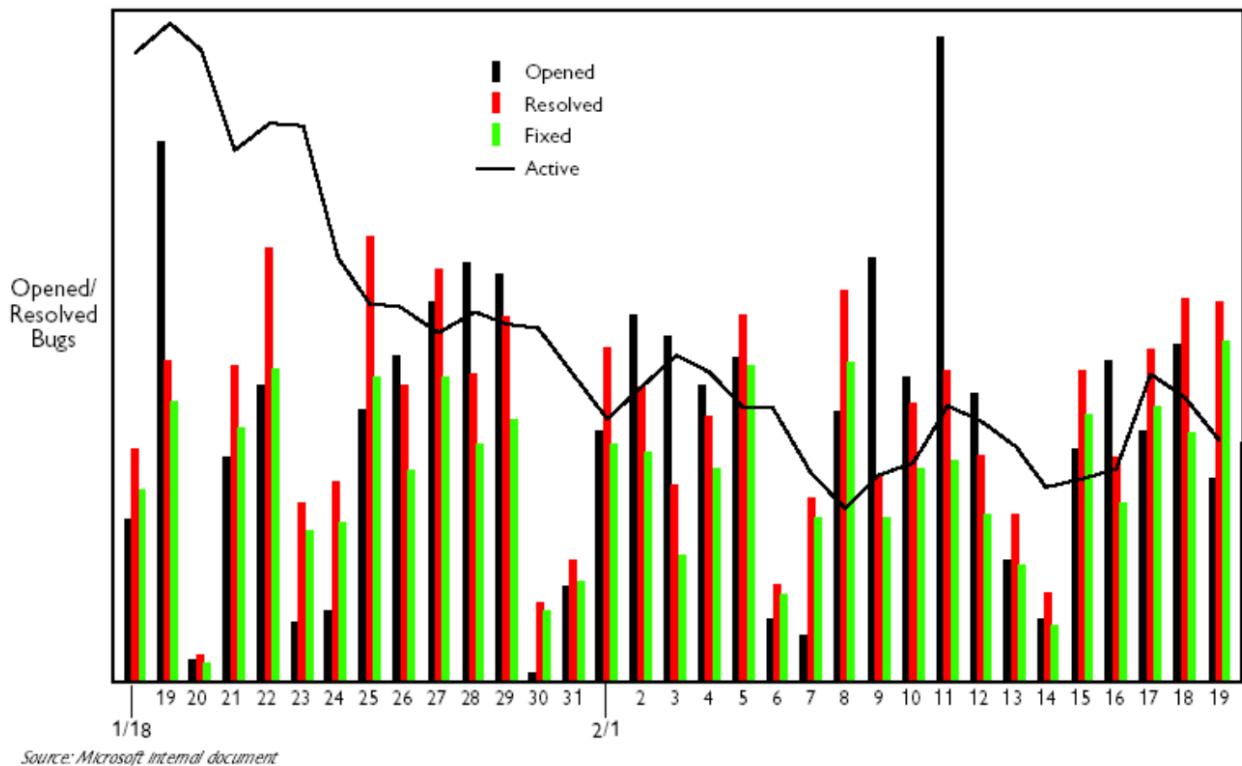
- **Sotto-progetto I:** Primo terzo delle funzionalità (le più critiche e i componenti condivisi)
- **Sotto-progetto II:** Secondo terzo delle funzionalità
- **Sotto-progetto III:** Ultimo terzo delle funzionalità (le meno critiche)

<b>Fase di Stabilizzazione</b> Test approfonditi interni ed esterni, stabilizzazione finale del prodotto e rilascio
<p><b>Stabilization Phase</b> Comprehensive internal and external testing, final product stabilization, and ship</p> <p>Program managers coordinate OEMs and ISVs and monitor customer feedback. Developers perform final debugging and code stabilization. Testers recreate and isolate errors.</p> <ul style="list-style-type: none"> <li>• <b>Internal Testing</b> Thorough testing of complete product within the company</li> <li>• <b>External Testing</b> Thorough testing of complete product outside the company by "beta" sites, such as OEMs, ISVs, and end users</li> <li>• <b>Release preparation</b> Prepare final release of "golden master" disks and documentation for manufacturing</li> </ul>

M. A. Cusumano e R. W. Selby, nel loro articolo definiscono pure le strategie che stanno alla base dello sviluppo software in Microsoft. Queste strategie presentate nell'articolo sono due:

1. Strategia per definire prodotto e processo → "considerare la creatività come elemento essenziale". Principi di realizzazione:
  - a. Dividere il progetto in *milestone* (da 3 a 4)
  - b. Definire una "product vision" e produrre una specifica funzionale che evolverà durante il progetto
  - c. Selezionare le funzionalità e le relative priorità in base alle necessità utente
  - d. Definire un'architettura modulare per replicare nel progetto la struttura del prodotto
  - e. Assegnare task elementari e limitare le risorse
  
2. Strategia per lo sviluppo e la consegna dei prodotti → "lavorare in parallelo con frequenti sincronizzazioni". Principi di realizzazione:
  - a. Definire team paralleli ed utilizzare daily build per la sincronizzazione
  - b. Avere sempre un prodotto da consegnare, con versioni per ogni piattaforma e mercato
  - c. Usare lo stesso linguaggio di programmazione all'interno dello stesso sito di sviluppo
  - d. Testare continuamente il prodotto durante il suo sviluppo
  - e. Usare metriche per il supporto alle decisioni

L'unica metrica che viene collezionata è quella riportata in foto



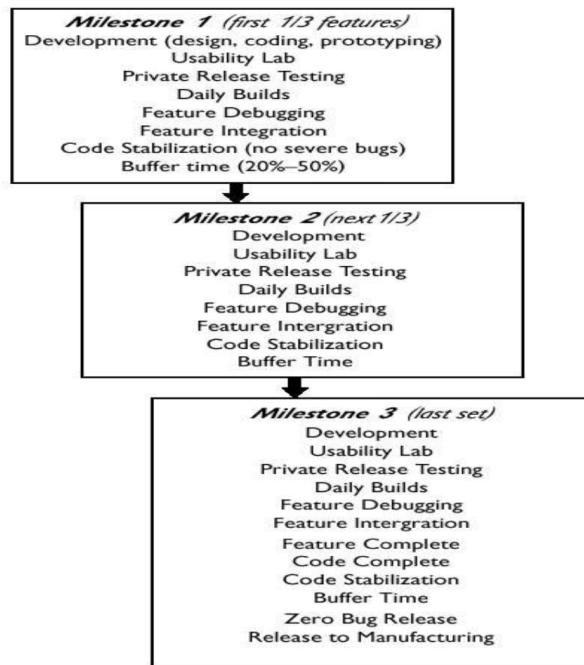
Ad ogni giorno sono associate delle metriche che definiscono i difetti. Nel grafico ogni giorno vedremo 3 colori:

- La barra nera indica i “difetti aperti”, ovvero quando si riscontra un malfunzionamento bisogna capire perché il software non funziona correttamente
- La barra rossa indica i difetti già localizzati, ovvero quando io ho capito dove si trova già il difetto
- La barra verde indica i difetti localizzati e che ho rimosso

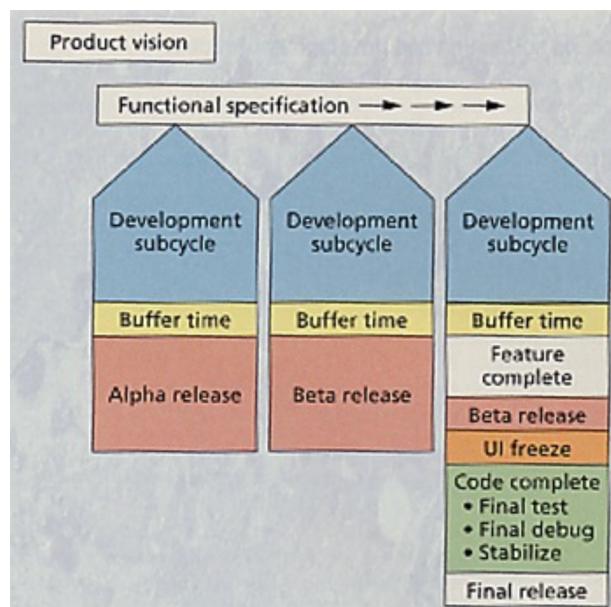
La linea tratteggiata rappresenta il numero di difetti attivi, ovvero quelli che sono stati identificati, ma non ancora risolti.

La qualità del prodotto è determinata dalle aspettative del cliente; in questo caso, Microsoft funge sia da cliente che da produttore, stabilendo i propri standard di qualità. L'affidabilità del prodotto è legata all'andamento di questa linea tratteggiata: se rimane al di sotto di una soglia prestabilita, possiamo considerare il prodotto pronto per essere immesso sul mercato, come promesso.

Un esempio di sviluppi di *Milestones*



Modello del ciclo di sviluppo *synch-and-stabilize*



Possiamo vedere che partiamo da una *Product Vision*, che è quella che viene messa sul mercato, poi ci sta una *Specific Funzionale*, che è seguita da frecce per specificare che va avanti fino alla fine. Poi ci stanno le varie fasi di *milestones*.

La figura mostra il ciclo di sviluppo del prodotto, che parte dalla *Product Vision*, cioè la visione complessiva del prodotto che si vuole immettere sul mercato.

Successivamente si passa alla *Specificazione Funzionale*, che continua lungo tutto il ciclo di sviluppo (indicata dalle frecce).

Il processo è suddiviso in tre *sottocicli di sviluppo* (*Development Subcycle*), ognuno dei quali include un *Buffer Time*, cioè un tempo aggiuntivo rispetto a quello stimato per gestire eventuali imprevisti e mantenere il progetto nei tempi previsti.

Ogni sottociclo porta a diverse fasi di rilascio:

- Il primo sottociclo culmina nella *Alpha Release*, una versione iniziale del software.
- Il secondo sottociclo rilascia la *Beta Release*, una versione più avanzata ma ancora soggetta a test.
- Il terzo sottociclo prevede la fase di *UI Freeze*, in cui l'interfaccia utente viene congelata, seguita dal completamento del codice e delle funzionalità principali, per poi procedere con i test finali e le ultime correzioni.

Una volta superate tutte queste fasi, si arriva alla *Final Release*, la versione stabile e pronta per il mercato.

### Confronto tra modelli *synch - and - stabilize* e *waterfall*

Synch-and-Stabilize	Sequential Development
Product development and testing done in parallel	Separate phases done in sequence
Vision statement and evolving specification	Complete "frozen" specification and detailed design before building the product
Features prioritized and built in 3 or 4 milestone subprojects	Trying to build all pieces of a product simultaneously
Frequent synchronizations (daily builds) and intermediate stabilizations (milestones)	One late and large integration and system test phase at the project's end
"Fixed" release and ship dates and multiple release cycles	Aiming for feature and product "perfection" in each project cycle
Customer feedback continuous in the development process	Feedback primarily after development as inputs for future projects
Product and process design so large teams work like small teams	Working primarily as a large group of individuals in a separate functional department

- Lezione 7 – (31/10/2024)

- Il Modello Netscape

*M. A. Cusumano e D. B. Yoffie* hanno fatto un’esperienza simile, a quella che *M. A. Cusumano e R. W. Selby* hanno fatto alla Microsoft, in un’azienda più piccola dal nome Netscape, la quale lavorava a progetti adattabili allo sviluppo di applicazioni Internet, browser e prodotti server. Anch’essi, nell’Ottobre del 1999, pubblicarono un articolo sulla rivista *IEEE Computer*, l’articolo *Software Development on Internet Time*.

La Netscape adottava lo stesso modello *synchronize-and-stabilize* che adottava la Microsoft.

Dal punto di vista della produttività i risultati sono simili tra le due aziende, nonostante la Netscape aveva una dimensione dello staff differente. Infatti aveva in media 1 tester ogni 3 sviluppatori.

Loro hanno osservato nell’azienda i seguenti difetti:

- Scarso effort di pianificazione ( tranne che su prodotti server )
- Documentazione incompleta
- Scarso controllo sullo stato di avanzamento del progetto (lasciato all’esperienza e all’influenza dei project manager)
- Scarso controllo su attività di ispezione del codice (codereview)
- Pochi dati storici per il supporto alle decisioni

Nell’articolo era presente la seguente tabella dove si vede che l’azienda era veramente piccola

**Table 2. Allocation of staff in Netscape’s client and server development divisions, mid-1998.**

	<b>Client products</b>	<b>Server products</b>	<b>Total</b>
Software engineering	110	200	310
Testing (QA)	50	80	130
Product management	50	42	92
Subtotal	210	322	533
Other*	30	98	128
Total	240	420	660

\*Persons in activities such as documentation, user interface design, OEM porting, internationalization and localization, and special product support.

Il processo di sviluppo Netscape:

- **Step 1: Requisiti del prodotto e proposta del progetto**

- Pianificazione anticipata del meeting (APM) tenuta per fare brainstorming di idee (marketing, sviluppo, esecutivo).
- Si sviluppa la visione del prodotto, inizialmente dagli ingegneri più esperti, adesso principalmente dai responsabili di prodotto (product managers).
- Della progettazione e scrittura di codice dagli ingegneri per esplorare tecnologie alternative o idee tecniche.
- Documento dei requisiti del prodotto compilato dai responsabili del prodotto con l'aiuto degli sviluppatori.
- Revisione non ufficiale di queste specifiche preliminari dagli ingegneri.
- La specifica funzionale inizia dagli ingegneri, talvolta con l'aiuto dai responsabili del prodotto.
- Programma e piano di bilancio compilati dal settore commerciale ed ingegneristico e discusso in modo non ufficiale con i dirigenti.

- **Step 2: Prima revisione esecutiva**

- Documento di revisione esecutiva dei requisiti del prodotto, programma e proposta di bilancio;
- Piano modificato secondo le necessità;

- **Step 3: Inizio della fase di sviluppo**

- Caratteristiche di design e scrittura in codice, lavoro di progettazione secondo le necessità;
- Integrazione giornaliera dei componenti come sono creati e costruiti;
- Lista dei problemi generati e inizio delle correzioni;

- **Step 4: Revisione esecutiva provvisoria (se necessario)**

- Le specifiche funzionali dovrebbero essere complete a questo punto;
- Correzione di metà strada nelle specifiche o risorse del progetto dove necessario;
- Problemi di coordinamento con altri prodotti o progetti discussi dove necessario;
- Continuo dello sviluppo;

- **Step 5: Primo rilascio interno (alpha) (impiega approssimativamente sei settimane)**
  - Lo sviluppo si arresta temporaneamente;
  - Intensivo debugging e testing del codice esistente;
  - Rilascio dell'Alpha per un feedback interno (o possibilmente un rilascio dello sviluppatore);
  - Continua lo sviluppo;
  - Inserimento del feedback dell'utente;
  - Obiettivo completamento delle funzionalità (raramente conseguito, nonostante i server in particolare cercano di essere il più completi possibile);
  - Una settimana per stabilizzare il rilascio beta;
- **Step 6: Pubblicazione di beta 1 o test sul campo 1 (impiega approssimativamente sei settimane)**
  - Ripetizione dei passi dello sviluppo e dei test nello step 5;
  - Gruppi server passano a test sul campo con clientela limitata piuttosto che beta pubbliche;
- **Step 7: Pubblicazione beta 2 e 3 (ogni beta impiega all'incirca sei settimane)**
  - Ripetizione dei passi dello sviluppo come nello Step 5;
  - Congelamento dell'interfaccia utente milestone (UI freeze milestone);
  - Completamento della funzione dello status “mandatorio”, anche se sono ancora concessi piccoli cambiamenti;
- **Step 8: Completamento del codice**
  - Nessuna aggiunta di codice eccetto quello per correggere i bug; le caratteristiche sono funzionalmente complete;
- **Step 9: Prova finale e rilascio**
  - Ricerca degli errori (debugging) finale e stabilizzazione della versione di rilascio;
  - Meetings di abilitazione con i dirigenti più esperti (senior) per la decisione del lancio;
  - Rilascio per la produzione (RTM Release to manufacturing) e rilascio commerciale;

All'inizio degli anni 2000, è emerso un approccio critico verso i processi software tradizionali, spesso percepiti come troppo rigidi e limitanti per i team di sviluppo. Questo ha portato all'introduzione dei *metodi agili*, un insieme di pratiche che estendevano il concetto di sviluppo iterativo e incrementale, privilegiando aspetti come la comunicazione continua, il feedback rapido e la flessibilità nel modo di lavorare.

Il *Manifesto Agile*, documento fondamentale per questo movimento, raccoglie i valori e i principi condivisi dai metodi agili. I valori principali del *Manifesto Agile* sono:

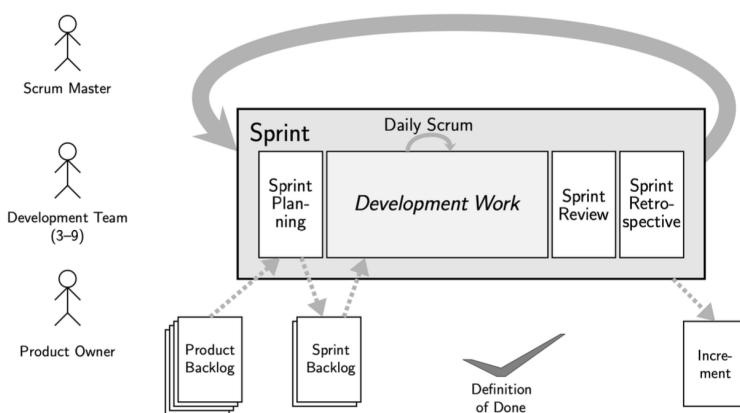
- Valorizzare le persone e le interazioni più dei processi e degli strumenti.
- Preferire un software funzionante rispetto alla documentazione esaustiva.
- Favorire la collaborazione con il cliente invece di una rigida negoziazione contrattuale.
- Scegliere di adattarsi ai cambiamenti piuttosto che seguire un piano fisso.

Questi valori non eliminano del tutto gli elementi più tradizionali, ma stabiliscono che il team darà priorità agli aspetti dinamici e flessibili del lavoro.

In aggiunta ai valori, il *Manifesto Agile* descrive dodici principi che offrono ulteriori indicazioni per applicare correttamente i metodi agili, promuovendo un approccio collaborativo e adattabile nello sviluppo del software. Insieme, questi valori e principi formano la base di ciò che oggi chiamiamo *sviluppo agile*.

Quello che noto oggi giorno da aziende che sviluppano software è l'approccio *Scrum* (tradotto "mischia"). Questo approccio non ne stato introdotto per lo scopo dell'ingegneria del software, è stato introdotto da due giapponesi, ben prima del *Manifesto Agile*.

La metodologia è formalizzata nella *Scrum Guide*, che descrive ruoli, eventi e artefatti principali.



La metodologia prevede tre ruoli principali:

1. *Scrum Master* → Facilita l'implementazione di Scrum, assicurando che il team e il proprietario del prodotto rispettino le regole e aiutando gli stakeholder a interagire efficacemente con il team.
2. *Proprietario del Prodotto* → Gestisce il backlog del prodotto, dando priorità ai requisiti da sviluppare per garantire che le funzionalità più importanti siano affrontate prima.
3. *Team di Sviluppo* → Responsabile dello sviluppo del prodotto, svolge attività che includono progettazione, codifica e test, collaborando per realizzare incrementi di prodotto funzionanti.

Il lavoro viene organizzato in cicli chiamati *Sprint*, della durata di 2-4 settimane. Durante ciascun *Sprint*, il team lavora su un insieme predefinito di elementi scelti dal backlog del prodotto. Ogni Sprint segue un ciclo strutturato:

1. *Sprint Planning* - Inizio dello Sprint, in cui il team definisce gli obiettivi selezionando gli elementi da sviluppare e trasferendoli nel backlog dello Sprint.
2. *Daily Scrum* - Riunione giornaliera breve (stand-up) in cui il team si allinea sulle attività e affronta eventuali ostacoli.
3. *Sprint Review* - Alla fine dello Sprint, l'incremento sviluppato viene presentato al proprietario del prodotto e agli stakeholder per ottenere feedback.
4. *Sprint Retrospective* - Riunione finale per analizzare le performance e individuare possibili miglioramenti per i successivi Sprint.

Ogni *Sprint* produce un incremento di software funzionante e migliorato rispetto alla versione precedente, seguendo un processo che incoraggia la collaborazione, la trasparenza e l'adattabilità del team.

Un altro elemento fondamentale di *Scrum* è la *Definition of Done*, che rappresenta il criterio di completezza stabilito dal team di sviluppo per considerare un'attività realmente conclusa. La *Definition of Done* prevede che il lavoro sia sufficientemente testato e integrato, senza introdurre errori nel codice principale.

Un'altra pratica comune, spesso usata in combinazione con *Scrum* (ma non formalmente definita nella *Scrum Guide*), è quella delle *User Stories*. Le *User Stories* sono brevi descrizioni dei requisiti dell'utente, formulate come "storie" dal suo punto di vista. Usano un formato standard:

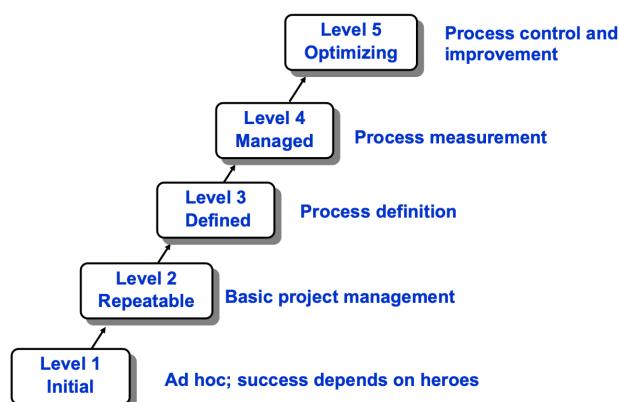
*As a <role>, I want <goal> so that <benefit>*

Le *User Stories* aiutano il team a comprendere le esigenze degli utenti. Quando una *User Story* è troppo grande viene suddivisa in *User Stories* più piccole per facilitarne la gestione e viene chiamata *Epics*.

Questi strumenti aiutano il team a mantenere chiarezza sugli obiettivi e sulla qualità del lavoro completato, migliorando la gestione dei requisiti e l'efficienza dei cicli di sviluppo.

- Modello di Maturità delle Capacità (CMM Capability Maturity Model):

Il *Capability Maturity Model (CMM)*, introdotto nel 1993 dal *SEI (Software Engineering Institute)*, è un modello ideato per valutare il livello di maturità del processo software all'interno di un'organizzazione, misurando l'efficacia complessiva nell'applicazione delle tecniche di ingegneria del software. Questo modello utilizza un questionario e uno schema di valutazione organizzato su cinque livelli gerarchici; ogni livello include tutte le caratteristiche definite nei livelli precedenti, rendendolo un modello di tipo "additivo". Ciò significa che, per raggiungere un determinato livello di maturità, è necessario aver soddisfatto i requisiti di tutti i livelli precedenti.



Noi illustreremo il modello *CMM originale*, senza considerare le evoluzioni successive, come il *CMMI (Capability Maturity Model Integration)*, in cui la "I" finale indica appunto l'integrazione (*Integrated*).

Ecco i cinque livelli di maturità del modello CMM:

- Il *livello Iniziale* è un livello dove non esiste ancora un processo software strutturato; il processo viene sviluppato in modo ad hoc per ciascun progetto. Il successo è quindi fortemente legato all'impegno e alle competenze delle singole persone coinvolte nello sviluppo del software, piuttosto che a processi stabiliti.
- Il secondo livello, chiamato *Repeatable*, qui sono presenti processi di base per gestire efficacemente i progetti, permettendo di monitorare costi, pianificazioni e attività. L'obiettivo è quello di replicare i successi ottenuti nei progetti

passati. Vengono applicate tecniche di project management di base, e il livello prende il nome proprio dalla capacità di ripetere pratiche efficaci.

- Il terzo livello, chiamato *Defined*, qui il progetto è definito sia da un punto di vista gestionale che ingegneristico. È qui che la maggior parte delle aziende riesce a ottenere la certificazione, poiché il livello 3 è quello con il maggior numero di certificazioni rilasciate. I processi sono standardizzati e documentati, fornendo una solida base per lo sviluppo software.
- Il quarto livello, chiamato *Managed*, ovvero livello “*gestito*”, qui dopo aver definito il progetto, l'organizzazione è in grado di misurare e analizzare l'efficacia del proprio processo di sviluppo, basandosi su dati quantitativi. Ciò permette di avere un controllo più accurato delle performance e della qualità dei processi software.
- Il quinto livello, chiamato *Optimizing*, qui dopo aver definito il processo ed essere in grado di misurarne sono in grado di migliorarlo continuamente.

Per definire e valutare il livello di maturità di un'organizzazione secondo il modello CMM, il SEI ha individuato delle aree chiave, denominate *Key Process Areas (KPA)*. Ogni *KPA* rappresenta una funzione specifica che l'organizzazione deve implementare per conseguire la certificazione del livello desiderato. Complessivamente, esistono 18 *KPA* da soddisfare per poter raggiungere il quinto livello del modello CMM.

Ogni *KPA* è descritta rispetto a:

- Obiettivi
- impegni e responsabilità da assumere
- capacità e risorse necessarie per la realizzazione – attività da realizzare
- metodi di "monitoring" della realizzazione
- metodi di verifica della realizzazione

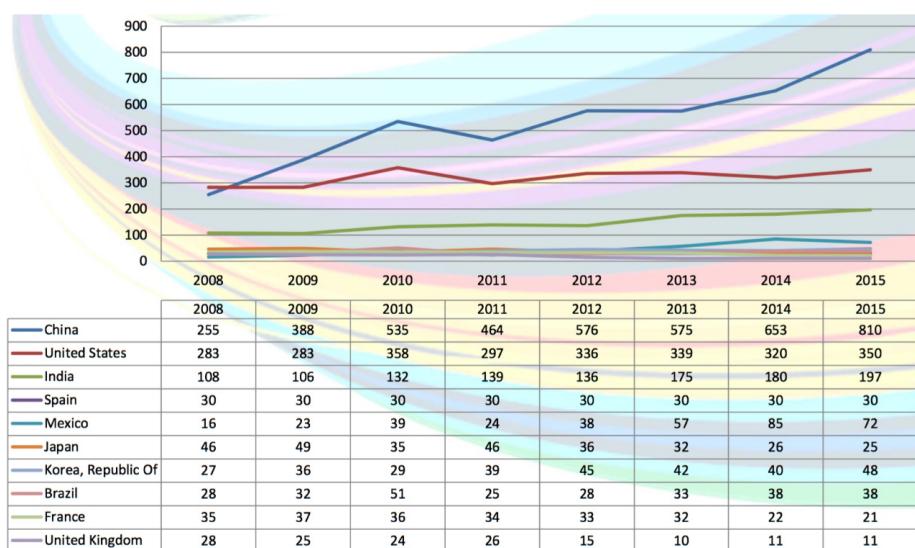
Ecco allocate le 18 *KPA* per ciascun livello

Level	Characteristic	Key Process Areas	Result
<i>Optimizing (5)</i>	<i>Continuous process capability improvement</i>	<i>Process change management</i> <i>Technology change management</i> <i>Defect prevention</i>	<b>Productivity &amp; Quality</b>
<i>Managed (4)</i>	<i>Product quality planning; tracking of measured software process</i>	<i>Software quality management</i> <i>Quantitative process management</i>	
<i>Defined (3)</i>	<i>Software process defined and institutionalized to provide product quality control</i>	<i>Peer reviews</i> <i>Intergroup coordination</i> <i>Software product engineering</i> <i>Integrated software management</i> <i>Training program</i> <i>Organization process definition</i> <i>Organization process focus</i>	
<i>Repeatable (2)</i>	<i>Management oversight and tracking project; stable planning and product baselines</i>	<i>Software configuration management</i> <i>Software quality assurance</i> <i>Software subcontract management</i> <i>Software project tracking &amp; oversight</i> <i>Software project planning</i> <i>Requirements management</i>	
<i>Initial (1)</i>	<i>Ad hoc (success depends on heroes)</i>	<i>"People"</i>	<b>Risk</b>

Una Statistica fatta a febbraio 2000, che presenta una lista delle organizzazioni a livello 4 e 5 (maturità elevata) include:

- 71 organizzazioni negli USA, 44 a livello 4 (tra cui Oracle, NCR, Siemens Info Systems, IBM Global Services) e 27 organizzazioni a Livello 5 (tra cui Motorola, Lockheed-Martin, Boeing, Honeywell)
- 25 organizzazioni al di fuori degli USA, 1 a livello 4 in Australia, 14 a livello 4 in India, 10 a livello 5 in India

*Number of appraisals by country (06/15)*



*Trends (as of June 2015)*

