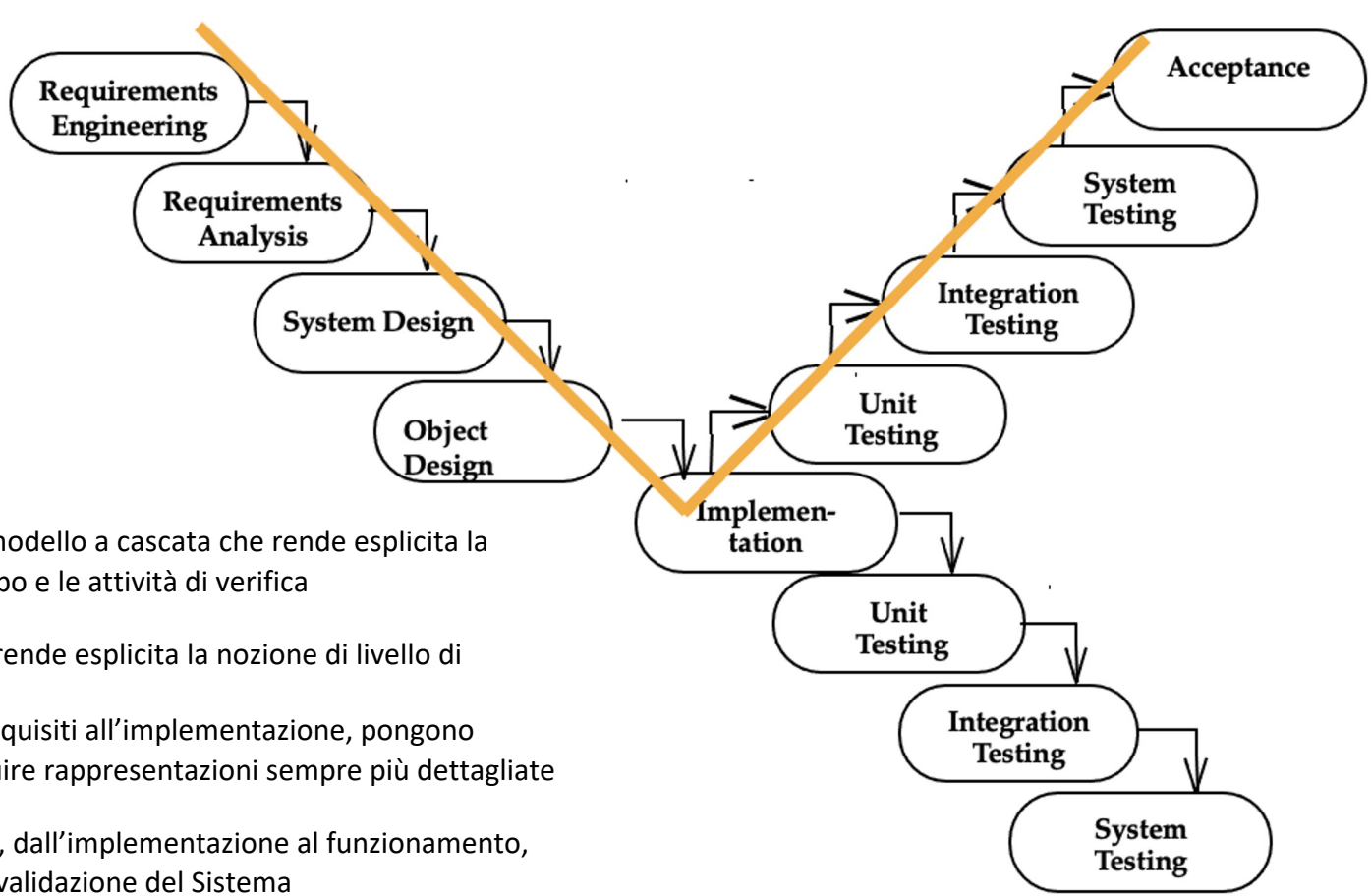
3. Modelli del ciclo di vita del software (2)

Paola Barra
a.a. 2023/2024

Il modello a V

Hughes Aircraft - 1982 (sequenziale)

Dal modello a cascata al modello a V

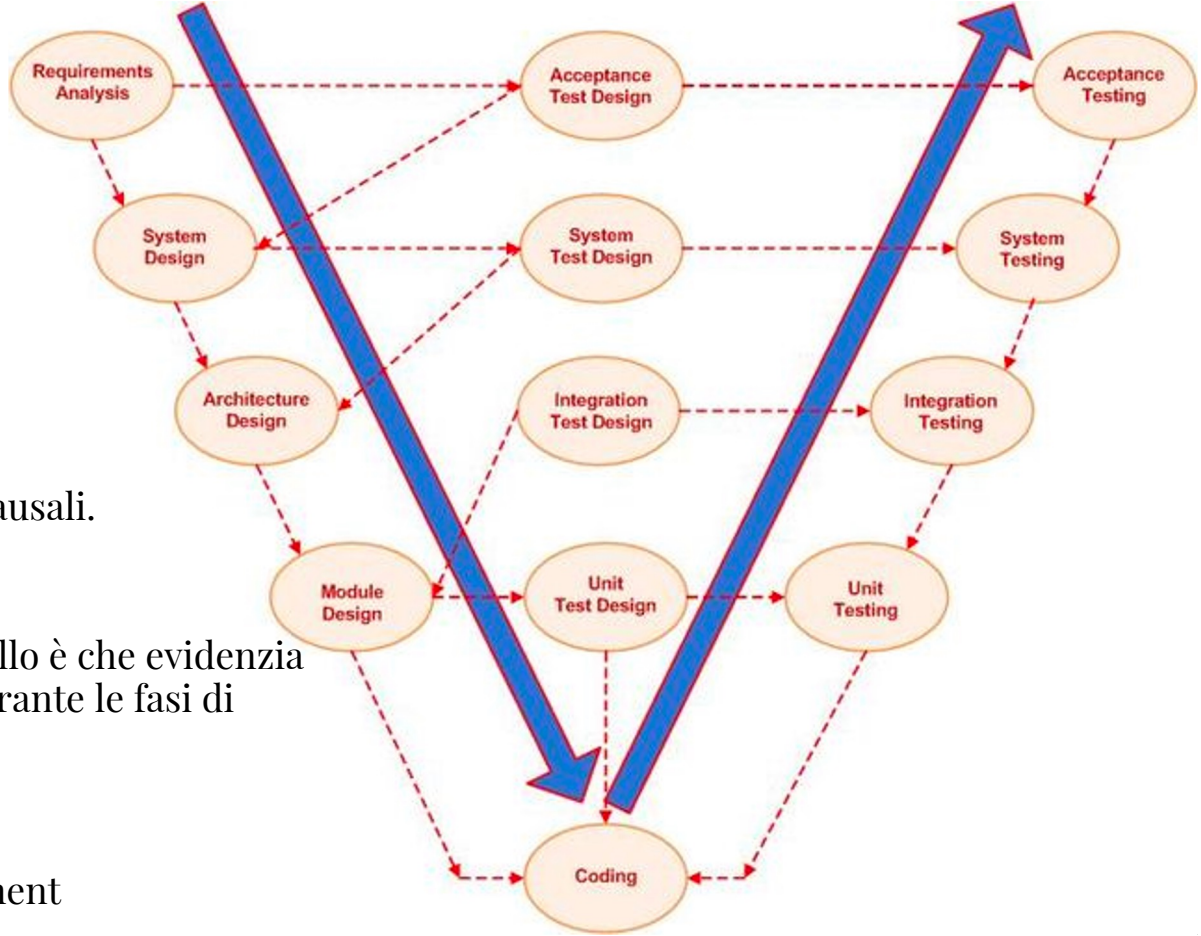


Il modello a V è una variante del modello a cascata che rende esplicita la dipendenza tra le attività di sviluppo e le attività di verifica

La differenza è che il modello a V rende esplicita la nozione di livello di astrazione

- Tutte le attività, dai requisiti all'implementazione, pongono l'attenzione nel costruire rappresentazioni sempre più dettagliate del sistema
- invece tutte le attività, dall'implementazione al funzionamento, sono focalizzate sulla validazione del Sistema

Il modello a V



Le frecce blu rappresentano il tempo.

Le frecce tratteggiate le dipendenze causali.

L'aspetto interessante di questo modello è che evidenzia come sia possibile progettare i test durante le fasi di sviluppo (prima della codifica)

Idea ripresa nel Test Driven Development

V-model

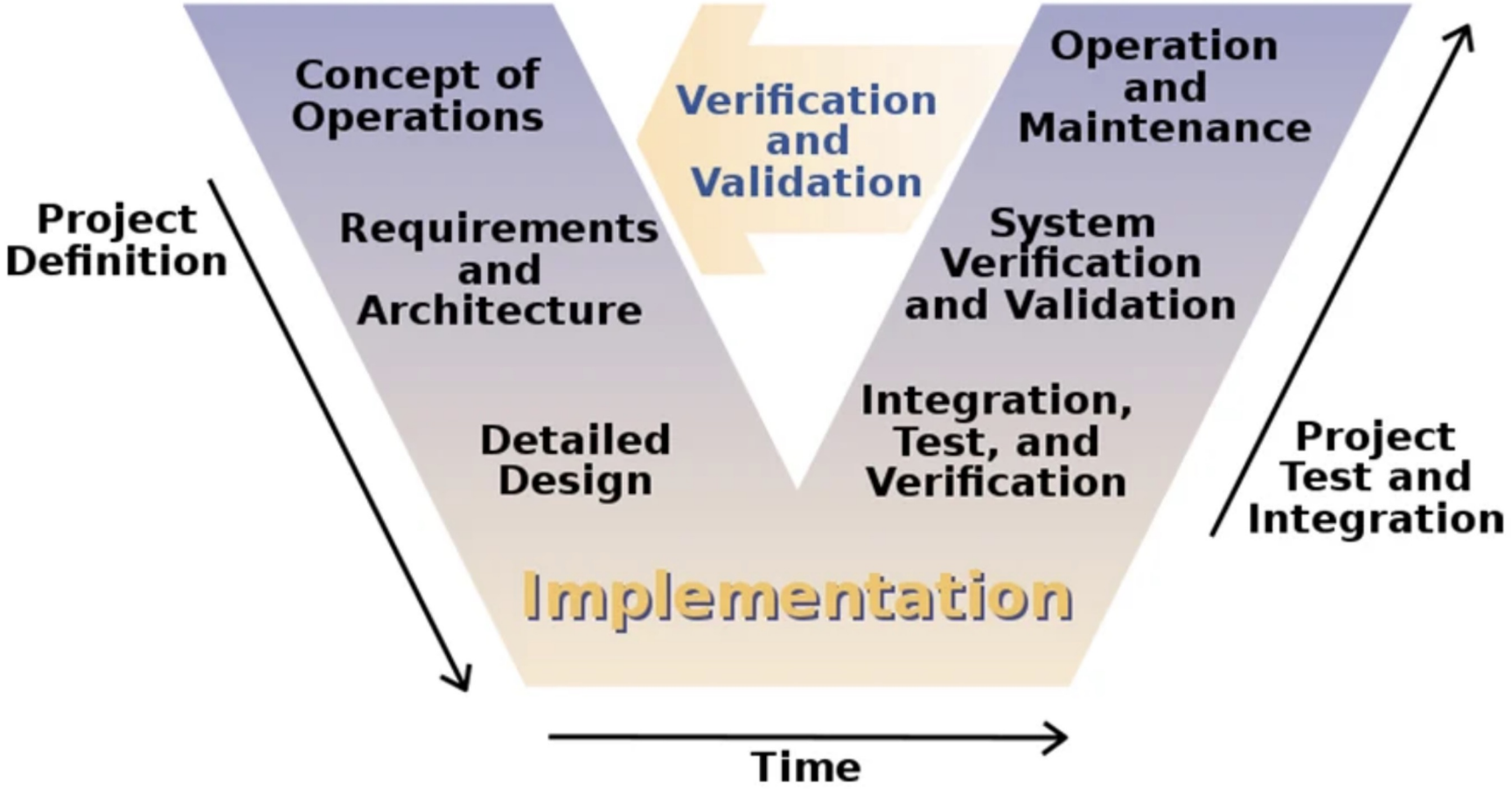
Il modello V è un modello di ingegneria del software utilizzato per facilitare lo sviluppo, la manutenzione e il miglioramento dei sistemi IT e per i requisiti di sicurezza del software.

Grazie ai vantaggi che porta in termini di **riduzione dei rischi** e di **aumento dell'efficienza**, V-model è un modello di sviluppo ampiamente utilizzato nelle organizzazioni industriali. Può infatti essere utilizzato per progetti software di qualsiasi dimensione e nei settori più disparati.

La rappresentazione logica del modello invece di discendere lungo una linea retta, dopo la fase di programmazione risale con la tipica forma della lettera V. **Il modello dimostra la relazione tra ogni fase del ciclo di vita dello sviluppo del software e la sua fase di testing (collaudo del software).**

Il V-model realizza un metodo ben strutturato, in cui ogni fase è implementabile tramite la documentazione dettagliata della fase precedente. **Le attività di testing hanno luogo già all'inizio del progetto prima della codifica, il che consente abbattere i tempi di progetto.**

Oltre alle rispettive fasi di sviluppo di un progetto, il V-model definisce in parallelo le procedure di garanzia della qualità e descrive come le singole fasi possono interagire tra loro.


Concept of Operations
Requirements and Architecture
Detailed Design
Implementation
Integration, Test, and Verification
System Verification and Validation
Operation and Maintenance
Verification and Validation
Project Definition
Project Test and Integration
Time

Il modello standard

All’inizio del progetto, il modello prevede un’analisi dei requisiti generali del sistema pianificato. In seguito, si arricchisce di requisiti per l’architettura di sistema. Segue quindi la progettazione del sistema, in cui sono pianificati i componenti e le interfacce del sistema. Una volta completate queste fasi, può essere progettata nel dettaglio l’architettura del software

Dopodiché segue l’effettivo sviluppo del software secondo gli schemi definiti e infine le fasi di garanzia della qualità, riferite alle varie fasi di sviluppo.

Il modello a V prevede la possibilità che ci siano dei feedback tra le fasi di testing / integrazione e le fasi di definizione poste allo stesso livello. Tali feedback costituiscono le correzioni e le modifiche da applicare quale esito negativo delle fasi di testing e di integrazione.

La corretta implementazione dell’architettura software pianificata viene verificata mediante unit test, i quali consentono di verificare nel dettaglio se i singoli moduli del software soddisfano esattamente le funzioni richieste e forniscono realmente i risultati attesi.

Alla fine del progetto, l’analisi dei requisiti dell’intero sistema viene messa in relazione con il collaudo del prodotto finito. Al momento del collaudo finale, il cliente verifica se le specifiche sono rispettate durante il funzionamento. Di norma, viene testato solo il comportamento del software a livello di interfaccia (test di accettazione).

Vantaggi e svantaggi del V-model

Garantisce un elevato grado di trasparenza e processi chiaramente definiti e comprensibili. Tra i vantaggi offerti vi sono l'**ottimizzazione della comunicazione tra le parti coinvolte**; la **minimizzazione dei rischi** e una **migliore pianificazione**; il **miglioramento della qualità del prodotto**; il **risparmio sui costi** grazie ad attività trasparenti dell'intero ciclo di vita del prodotto.

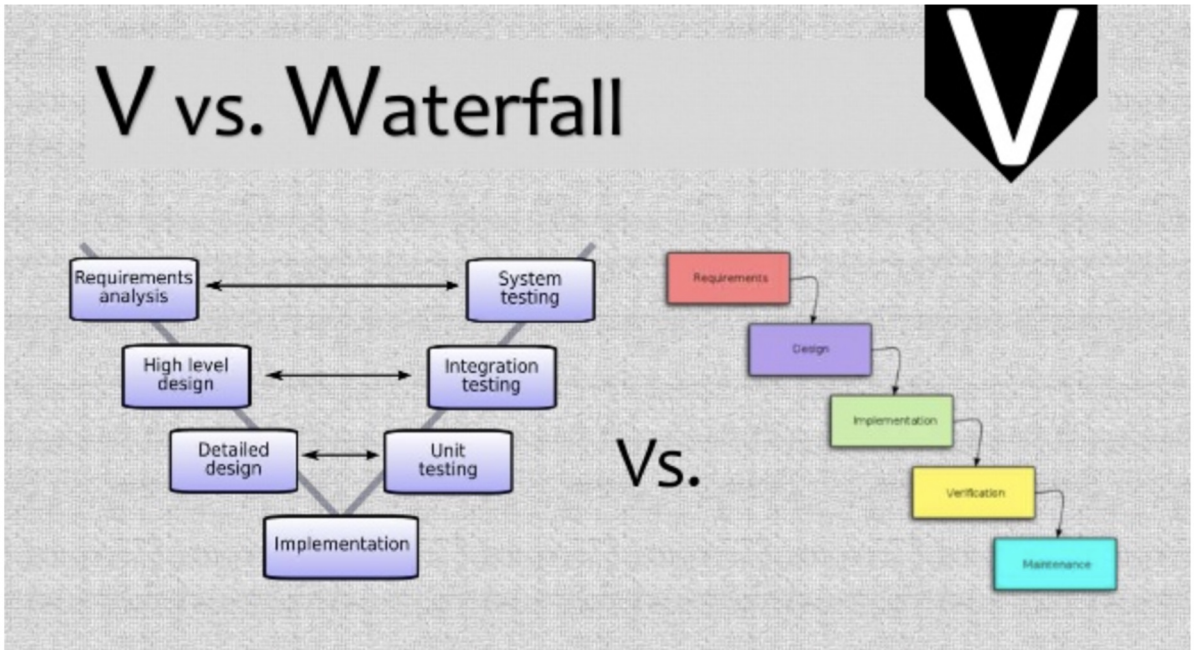
Nel complesso il modello può aiutare ad evitare lavori superflui e assicurare che tutte le attività siano completate al momento giusto e nel giusto ordine, riducendo al minimo il tempo di inattività.

In alcuni casi **però** il modello di sviluppo è inadeguato per mappare completamente il processo di sviluppo dal punto di vista degli sviluppatori e la struttura relativamente rigida non consente quasi mai una risposta flessibile ai cambiamenti.

In questi casi sono disponibili modelli di sviluppo alternativi che possono essere utilizzati a seconda del progetto e della struttura del team. Ad esempio il modello a cascata è particolarmente adatto per piccoli progetti lineari, mentre il modello a spirale può essere utilizzato per progetti iterativi.

Differenze tra modello a cascata e V-model

La differenza principale tra il modello a cascata e il modello a V è che nel modello a cascata il test del software viene eseguito dopo il completamento della fase di sviluppo, mentre nel modello a V, ogni fase del ciclo di sviluppo ha una fase di test direttamente associata.

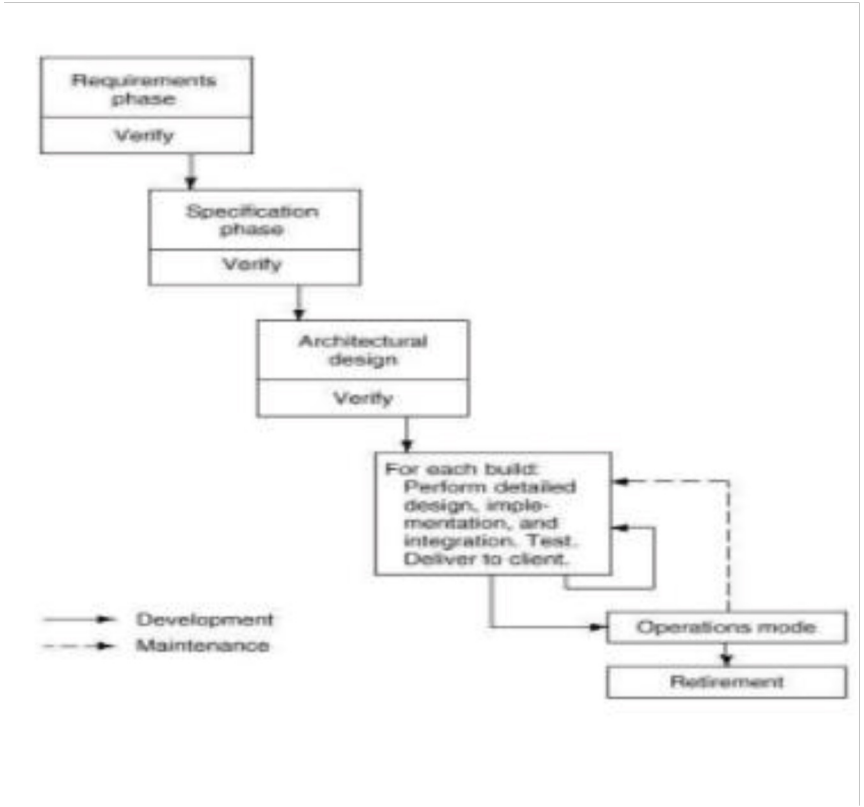


Modello incrementale

Modello incrementale

Il sistema è costruito aggiungendo qualcosa a quello che già era stato fatto

- I requisiti e il progetto sono definiti inizialmente, poi
- Il sistema è implementato, integrato e testato con una serie di passaggi incrementali

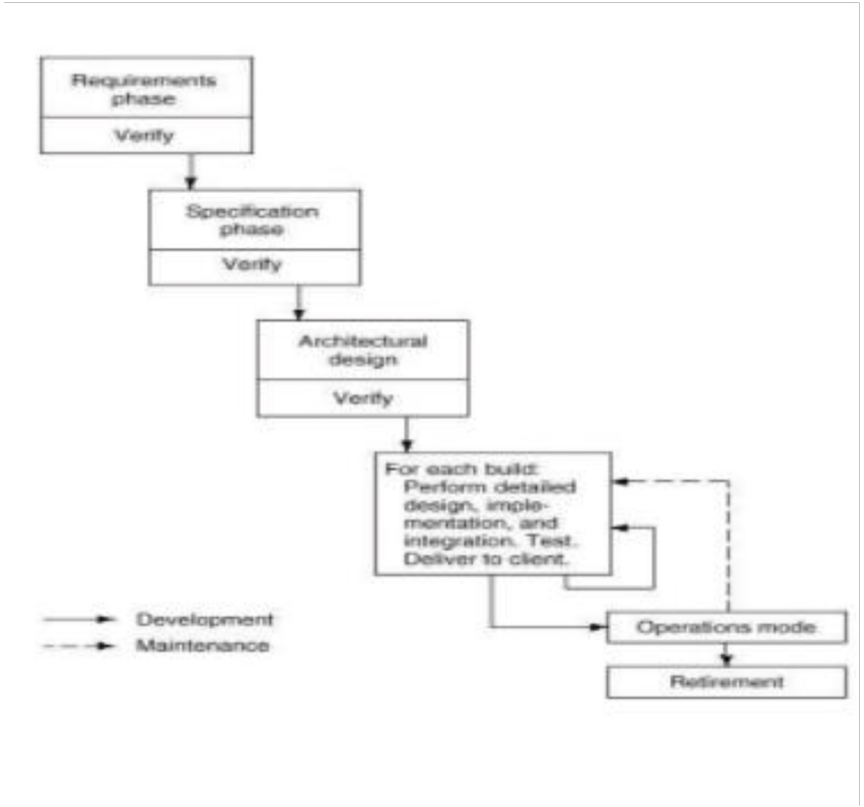


Modello incrementale

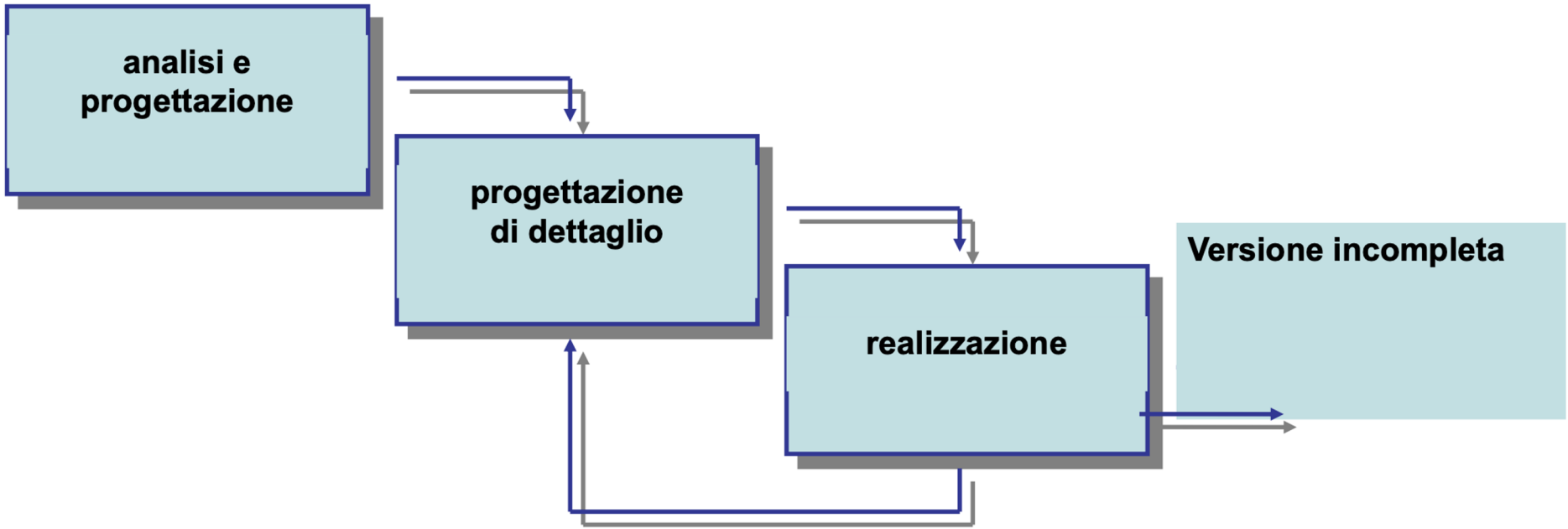
Si applica in caso di requisiti stabili, serve a :

- ritardare le realizzazioni delle componenti che dipendono criticamente da fattori esterni (tecnologie, hardware sperimentale, ecc)

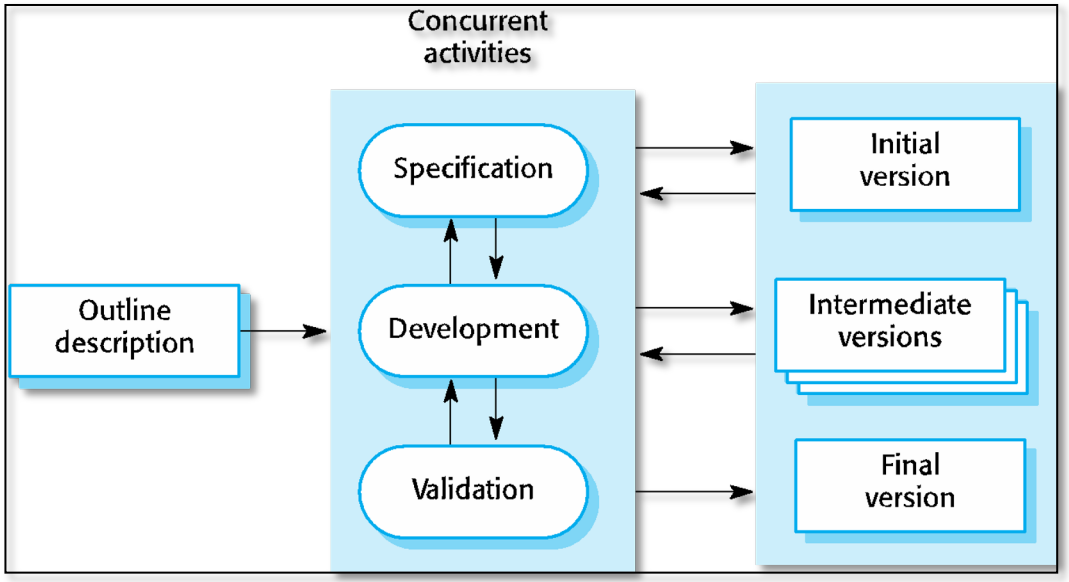
Se non è progettato bene diventa un Build and Fix.



Modello incrementale



Sviluppo incrementale



Si basa sull'idea di sviluppare un'implementazione iniziale, esporla agli utenti e perfezionarla attraverso molte versioni, finchè non si ottiene il sistema richiesto.

Può essere agile.

Gli incrementi del sistema sono stabiliti in anticipo.

Vengono definiti gli incrementi iniziali, ma lo sviluppo dei successivi incrementi dipende dall'avanzamento del lavoro e dalle priorità del cliente.

Modello incrementale

Prevede rilasci multipli e successivi

- Ciascuno realizza un incremento di funzionalità

I requisiti sono classificati e trattati in base alla loro importanza strategica

- I primi incrementi puntano a soddisfare i requisiti più importanti sul piano strategico
- Così i requisiti importanti diventano presto chiari e stabili, quindi più facilmente soddisfacibili
- Quelli meno importanti hanno invece più tempo per stabilizzarsi e armonizzarsi con lo stato del sistema

Modello incrementale

Analisi dei requisiti e progettazione architetturale vengono svolte una sola volta

- Per stabilizzare presto i requisiti principali
- Per stabilizzare presto l'architettura complessiva del sistema
- Per decidere preventivamente il numero di incrementi e i loro specifici obiettivi

La realizzazione è incrementale

- Raffinando l'analisi dei requisiti e la progettazione di dettaglio, strettamente entro l'architettura adottata
- Il completamento dei primi incrementi serve a rendere disponibili le principali funzionalità

Benefici dello sviluppo incrementale

Il costo per soddisfare le richieste di modifiche dei requisiti del cliente è ridotto

- La quantità di analisi e di documentazione che deve essere rifatto è molto inferiore di quanto richiesto dal modello a cascata

E’ più facile considerare il feedback del cliente sul lavoro di sviluppo che è stato fatto

- I clienti possono commentare le dimostrazioni del software e vedere quanto è stato implementato

E’ possibile una consegna più rapida e la distribuzione di software utile al cliente

- I clienti sono in grado di usare e riceverne guadagno dal software molto prima di quanto possibile con un processo a cascata

Problemi dello sviluppo incrementale

Il processo non è visibile

- I responsabili hanno bisogno di deliverable consegnati con regolarità per misurare il progresso
- Se i sistemi sono sviluppati velocemente, non è efficiente con i costi di produzione di documenti associati ad ogni versione del sistema

La struttura del sistema tende a degradare quando sono aggiunti nuovi incrementi

- A meno che non sono investiti tempo e denaro di refactoring per migliorare il software, le modifiche regolari tendono a corromperne la struttura
- Incorporare ulteriori modifiche nel software diventa sempre più difficile e costoso

Sviluppo basato su componenti

Basato sul riuso del software dove i sistemi sono integrati da componenti o sistemi esistenti

Gli elementi riusati possono essere configurati per adattare il loro comportamento e funzionamento ai requisiti dell'utente

Il riuso è ora un approccio standard per costruire molti sistemi commerciali

Tipi di software riusabile

Sistemi applicativi stand-alone (talvolta chiamati COTS) che sono configurati per essere usati in particolari ambienti

Collezione di oggetti che sono sviluppati come un package da integrare con un framework di componenti come .NET or J2EE

Web service che sono sviluppati in accordo agli standard di servizio e disponibili per invocazioni da remoto

Vantaggi e svantaggi del riuso del software

Vantaggi

- Costi e rischi ridotti poiché è sviluppato meno software da zero
- Consegna e distribuzione del sistema più veloce

Svantaggi

- Compromessi sui requisiti inevitabili
  - il sistema potrebbe non soddisfare le reali esigenze degli utenti
- Perdita di controllo sull'evoluzione degli elementi del sistema riusati

Modelli sempre più iterativi

- Modello a cascata
- Modello a V
- Rapid prototyping
- Incrementale
- Modello a spirale

Metodi agili

Metodi agili

Nascono alla fine degli ’90 in reazione all’eccessiva rigidità dei modelli allora prevalenti

- http://agilemanifesto.org/

Si basano su quattro principi fondanti

- Individuals and interactions over processes and tools
  L’eccessiva rigidità ostacola l’emergere del valore
- Working sofware over comprehensive documentation
  La documentazione non sempre corrisponde a SW funzionante
- Customer collaboration over contract negotiation
  L’interazione con gli stakeholder va incentivata e non ingessata
- Responding to change over following a plan
  La capacità di adattamento al cambiare delle situazioni è importante

Processi agili

Per metodologia agile (o leggera) o metodo agile si intende un particolare metodo per lo sviluppo del software che coinvolge quanto più possibile il committente.

Adatti a progetti con meno di 50 sviluppatori

Una metodologia agile si basa sui principi del Manifesto di Snowbird, feb 2021.

- http://agilemanifesto.org/

Agile manifesto (aka Snowbird)

Comunicazione:

- tutte le persone e le interazioni sono più importanti di processi e strumenti
- tutti possono parlare con tutti, e.g. l'ultimo dei programmatori con il cliente
- la comunicazione tra gli attori di un progetto sw è la miglior risorsa del progetto;
- collaborare con i clienti al di là del contratto (la collaborazione diretta offre risultati migliori dei rapporti contrattuali; gli analisti mantengaono la descrizione formale il più semplice e chiara possibile)

Semplicità:

- è più importante avere sw funzionante che documentazione
- bisogna mantenere il codice semplice e avanzato tecnicamente, riducendo la documentazione al minimo indispensabile

Feedback:

- rilasciare nuove versioni del software ad intervalli frequenti
- sin dal primo giorno si testa il codice

Coraggio:

- dare in uso il sistema il prima possibile e implementare i cambiamenti richiesti man mano
- rispondere al cambiamento più che aderire al progetto

Gestione dei cambiamenti

Affrontare le modifiche

- Le modifiche sono inevitabili in tutti i grandi progetti software
  - I cambiamenti aziendali portano a nuovi requisiti o modifiche ai requisiti del sistema
  - Le nuove tecnologie aprono nuove possibilità per il miglioramento dell'implementazione
  - Cambiare le piattaforme comporta modifiche all'applicazione
- Le modifiche comportano una rilavorazione e quindi i costi delle modifiche sommano la rilavorazione (es., rianalizzare i requisiti) e i costi di implementazione della nuova funzionalità

Ridurre i costi di rilavorazione

- E’ necessario **prevedere le modifiche**
  - un’attività del processo sw che prevede i possibili cambiamenti prima che sia necessaria una rilavorazione significativa
    - Ad esempio, potrebbe essere sviluppato un **prototipo** per mostrare ai clienti le caratteristiche chiave del sistema
- **Tolleranza alle modifiche**
  - il processo è progettato in modo che le modifiche possono essere accomodate a costi relativamente bassi
    - di solito, comporta qualche forma di sviluppo incrementale
      - le modifiche proposte possono essere implementate in incrementi che non sono stati ancora sviluppati

Le contro indicazioni

SW privo di documentazione produce costo, non valore

- Commentare il codice non basta serve spiegare e motivare le scelte realizzative

Senza un piano, non si possono valutare rischi e avanzamenti

- La sola misurazione di consuntivo non può bastare

Cambiare si può, ma con consapevolezza del rapporto costo/benefici

Le contro indicazioni

L’idea base è il concetto di “user story”

- Una funzionalità significativa che l’utente vuole realizzare con il SW richiesto
- Cioè uno scenario d’uso

Ogni “user story” è definita da

- Un documento di descrizione del problema
- Il verbale delle conversazioni con gli stakeholder effettuate per discutere e comprendere il problema
- La strategia da usare per confermare che il SW realizzato soddisfi gli obiettivi di quel problema

Le contro indicazioni

Assunti base

- Suddividere il lavoro in piccoli incrementi a valore aggiunto, magari anche sviluppabili indipendentemente
- Sviluppare ciascun incremento in sequenza continua dall'analisi all'integrazione

Obiettivi strategici

- Poter sempre dimostrare al cliente quanto è stato fatto
- Verificare l'avanzamento tramite progresso reale
- Dare agli sviluppatori la soddisfazione del risultato

Buoni esempi

- Scrum, Kanban (just-in-time), Scrumban

METODI AGILI

L’idea base è il concetto di “user story”

- Una funzionalità significativa che l’utente vuole
- realizzare con il SW richiesto Cioè uno scenario d’uso

Ogni “user story” è definita da

- Un documento di descrizione del problema
- Il verbale delle conversazioni con gli stakeholder effettuate per discutere e comprendere il problema
- La strategia da usare per confermare che il SW realizzato soddisfi gli obiettivi di quel problema

Gestire i cambiamenti dei requisiti

Prototipizzazione del sistema

- E’ sviluppata velocemente una versione del sistema o una sua parte per verificare i requisiti del cliente e la fattibilità delle decisioni
- Supporta l’anticipazione delle modifiche

Consegna incrementale

- Sono consegnati al cliente degli incrementi del sistema con i quali è possibile fare commenti o sperimentare
- Supporta ambo le possibilità di evitare i cambiamenti e la loro tolleranza

Prototipizzazione del sistema

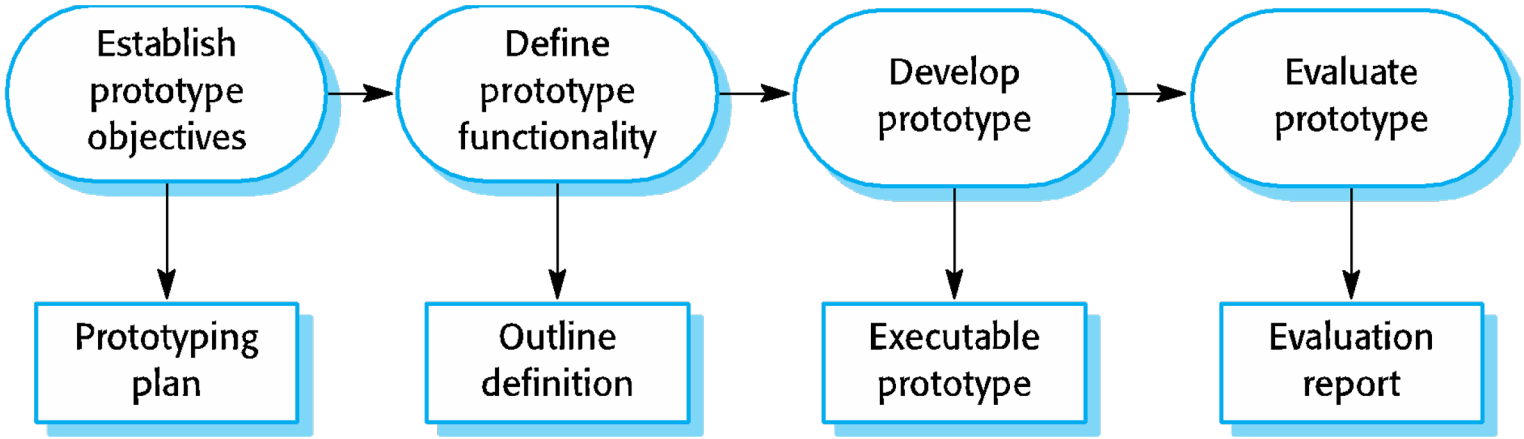
Prototipazione del software

- Un prototipo è una versione iniziale del sistema usato per dimostrare i concetti e determinare opzioni di progettazione
- Un prototipo può essere usato:
  - Nel processo di ingegneria dei requisiti per aiutare la scoperta e la validazione dei requisiti
  - Nei processi di progettazione per esplorare le opzioni e sviluppare un progetto di interfaccia utente
  - Nel processo di testing per eseguire test back-to-back

Benefici della prototipazione

- Migliore usabilità del sistema
- Maggiore corrispondenza con le necessità del cliente
- Migliore qualità di progettazione
- Migliore manutenibilità
- Sforzo di sviluppo inferiore

Processi di sviluppo di un prototipo



Sviluppo di un prototipo

- Può essere basato su linguaggi o strumenti di prototipazione rapida
- Può trascurare alcune funzionalità
  - Il prototipo dovrebbe focalizzarsi su parti del prodotto che non sono ben comprese
  - Il controllo di errori e il loro recupero può non essere incluso nel prototipo
  - Il focus è su requisiti funzionali e non su quelli non-funzionali quali attendibilità e sicurezza

Prototipi throw-away

- I prototipi dovrebbero essere scartati dopo lo sviluppo poiché non rappresentano una buona base per un sistema di produzione:
  - Può essere impossibile raffinare il sistema per soddisfare i requisiti non-funzionali
  - Normalmente, i prototipi non sono documentati
  - La struttura del prototipo è generalmente degradata dopo i rapidi cambiamenti
  - E’ molto probabile che il prototipo non soddisferà gli standard qualitativi organizzativi

Consegna incrementale

Consegna incrementale

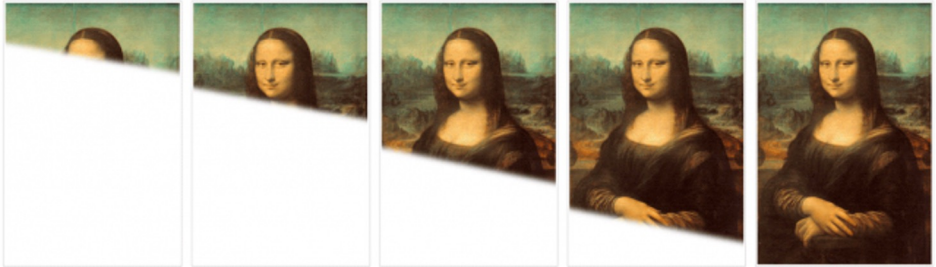
- Piuttosto che consegnare il sistema in un unica volta, lo sviluppo e la consegna è suddivisa in incrementi, attraverso i quali, ciascuno, consegna parte della funzionalità richiesta
- I requisiti utente sono resi prioritari e sono inclusi nei primi incrementi i requisiti di priorità più elevata
- Una volta che lo sviluppo di un incremento è avviato, i requisiti sono congelati sebbene i requisiti per gli incrementi successivi possano continuare ad evolvere

Sviluppo incrementale VS consegna incrementale

- Sviluppo incrementale
  - Sviluppa il sistema con incrementi e valuta ogni incremento prima di procedere allo sviluppo dell'incremento successivo
  - Approccio tipico usato nei metodi agili
- Consegna incrementale
  - Distribuisce un incremento usato dagli utenti finali
  - Valutazione più realistica sull'uso pratico del software
  - Difficile da implementare per sostituire il sistema poiché gli incrementi hanno minori funzionalità del sistema da sostituire

WATERFALL

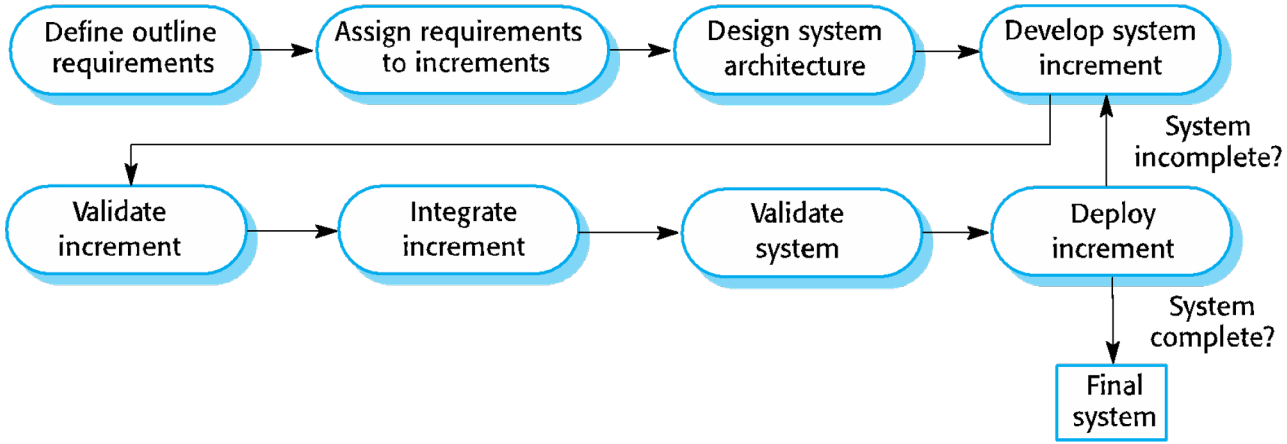
AGILE



Iterative

Incremental

Consegna incrementale



Vantaggi della consegna incrementale

- I primi incrementi fungono come un prototipo per aiutare nella scoperta dei requisiti per gli incrementi successivi
- Minor rischio di fallimento dell'intero progetto
- I servizi di sistema a maggiore priorità tendono ad essere maggiormente testati

Problemi della consegna incrementale

La maggior parte dei sistemi richiede una serie di servizi di base utilizzati da diverse parti del sistema

- Siccome i requisiti non sono definiti nel dettaglio fino a che è implementato un incremento, può essere difficile identificare i servizi comuni necessari a tutti gli incrementi

L'essenza dei processi iterativi è che la specifica è sviluppata congiuntamente al software

- Tuttavia, ciò confligge con il modello di acquisto delle organizzazioni, nel quale la specifica dell'intero sistema è parte del contratto dello per lo sviluppo del sistema

SCRUM

Cos’è lo scrum

Lo Scrum è un framework di sviluppo agile utilizzato principalmente nel campo del software, ma applicabile anche in altri settori, per gestire progetti complessi. È stato creato per affrontare problemi di sviluppo software che richiedono flessibilità, adattabilità e una stretta collaborazione tra membri del team.

Scrum è la metodologia Agile più diffusa ed è per questo che viene spesso confusa con Agile.

Il termine scrum viene utilizzato per la prima volta in un articolo di Nonaka e Takeuchi, The New New Product Development Game, pubblicato in Harvard Business Review nel 1986 in cui **descrivono una nuova modalità di lavoro dei team di sviluppo prodotto che ha forte somiglianze con la “mischia del rugby”**, scrum appunto, durante la quale non vi è nessun capo, nessun leader che prende le decisioni, non vi sono schemi è il team che in modo autonomo e auto organizzato prende le decisioni sul momento.

Scrum consiste in una serie di pratiche che rendono il lavoro dei team altamente produttivo e funzionante nella consegna di valore per il cliente finale.

Lo Scrum si basa su tre pilastri

Trasparenza: significa che tutti coloro che partecipano ad un progetto sanno quale è lo scopo (trasparenza verticale) e sanno che cosa fanno gli altri (trasparenza orizzontale). Trasparenza implica anche che il lavoro e le misure delle loro performance siano visibili a tutti a qualsiasi livello organizzativo.

Ispezione: basato sul concetto di controllo empirico di processo significa che ogni iterazione ed incremento vengano verificati in base alle metriche di misurazione decise per apportare modifiche alle iterazioni successive rendendo estremamente adattabile l'andamento del processo. Naturalmente occorre non superare la soglia di tolleranza del processo all'ispezione per non vanificarne la validità.

Adattamento: è la conseguenza dell'ispezione e significa che al posto di seguire un piano preordinato il team di sviluppo ripianifica in base ai risultati dell'ispezione per apportare il maggior valore al cliente finale orientato al miglioramento continuo delle proprie performance.

Ruoli chiave

- **Product Owner**: Responsabile della definizione delle funzionalità e delle priorità del prodotto.
- **Scrum Master**: Responsabile di facilitare il processo Scrum e rimuovere gli ostacoli che impediscono al team di lavorare efficacemente.
- **Team di sviluppo**: Il gruppo di professionisti che lavora insieme per consegnare il lavoro.

Tutti insieme formano lo Scrum Team, che ha la caratteristica di essere autogestito e cross-funzionale, vale a dire che ha al suo interno tutte le competenze per rilasciare un incremento di software senza dipendere da team esterni o da persone esterne al team.

Eventi Scrum

- **Sprint**: Un periodo di tempo fisso (solitamente da 2 a 4 settimane) durante il quale il team lavora per completare una quantità definita di lavoro.
- **Daily Scrum**: Una breve riunione quotidiana in cui i membri del team si aggiornano sugli avanzamenti e discutono dei problemi.
- **Sprint Review**: Una riunione alla fine di ogni sprint in cui il team presenta il lavoro completato e ottiene feedback dagli stakeholder.
- **Sprint Retrospective**: Una riunione alla fine di ogni sprint in cui il team riflette sul proprio processo e identifica miglioramenti.

Artifacts (Artefatti):

Per artefatti si intende le modalità con cui Scrum visualizza il lavoro e il “valore”. Questi artefatti sono:

- **Product Backlog**: Una lista priorizzata di tutte le funzionalità, gli elementi o le attività che devono essere sviluppate nel progetto.
- **Sprint Backlog**: Una selezione di elementi dal Product Backlog che il team si impegna a completare durante uno sprint.
- **Incremento**: Il risultato del lavoro del team alla fine di uno sprint, che dovrebbe essere potenzialmente consegnabile e di valore.

Ogni artefatto include un “**commitment**”, con il fine di migliorare la trasparenza e attraverso il quale è possibile misurare i progressi del team. Questi commitment sono:

- **Product Goal**, che fa parte del Product Backlog e descrive uno stato futuro del prodotto. Il Product Goal serve come obiettivo a lungo termine
- **Sprint Goal**, che è parte dello Sprint Backlog; descrive un obiettivo più a breve termine e il motivo per cui un’iterazione aggiunge valore al prodotto
- **Definiton of Done**, che descrive quando un Increment (ovvero un incremento di software) raggiunge uno standard qualitativo adeguato a consentire il rilascio all’utente finale