01 - Processi per lo Sviluppo Software

Qualità del Software

Il software non è solo un programma o un gruppo di programmi, ma include anche tutta la documentazione elettronica necessaria agli utenti, agli sviluppatori e ai responsabili della garanzia della qualità.

Le caratteristiche essenziali di un prodotto software di qualità sono:

- **Mantenibilità**: il software deve essere scritto in modo da potersi evolvere in base alle nuove richieste dei clienti.
- **Fidatezza**: un software affidabile non dovrebbe causare danni fisici o economici in caso di malfunzionamento.
- **Efficienza**: non dovrebbe fare un uso dispendioso delle risorse del sistema, come memoria o cicli di processore.
- **Accettabilità**: deve essere accettabile per il tipo di utenti per cui è progettato, ovvero **comprensibile**, **usabile** e **compatibile** con altri sistemi utilizzati dagli utenti.

Processo Software

In generale, un processo descrive chi fa che cosa, come e quando per raggiungere un obiettivo. Un processo per lo sviluppo del software (o processo software) descrive un **approccio disciplinato alla costruzione, al rilascio ed eventualmente alla manutenzione** del software.

Tutti i processi software condividono quattro attività fondamentali:

1. **Specifiche del software** (Ingegneria dei requisiti):
    - Clienti e sviluppatori definiscono le funzionalità e i vincoli operativi del software da produrre.
    - È lo stadio più critico del processo software, poiché errori in questa fase portano inevitabilmente a problemi successivi nella progettazione e implementazione.
    - Le fasi principali includono:
        - **Deduzione e analisi dei requisiti**: osservazione di sistemi esistenti, discussione con utenti, analisi dei task, ecc.
        - **Specifica dei requisiti**: traduzione delle informazioni raccolte in un documento che definisce un insieme di requisiti.
        - **Convalida dei requisiti**: controllo che i requisiti siano realistici, coerenti e completi.
2. **Sviluppo del software** (Progettazione e implementazione):

- Attività di conversione delle specifiche in un sistema eseguibile da consegnare al cliente.
- Nelle metodologie agili, *progettazione e implementazione sono intrecciate* e *non producono documenti formali*; il progetto è registrato informalmente (lavagne, portatili).
- Il **progetto** è una descrizione della struttura del software da implementare, dei modelli e delle strutture di dati usati, delle interfacce tra i componenti e degli algoritmi.
- Le **attività principali per la progettazione** di un sistema informativo includono:
  - **Progettazione dell'architettura**: identifica la struttura complessiva del sistema, dei componenti, delle loro relazioni e distribuzione.
  - **Progettazione del database**: progetta le strutture dati del sistema e la loro rappresentazione in un database.
  - **Progettazione dell'interfaccia**: definisce l'interfaccia dei componenti in modo che possano essere usati senza conoscerne l'implementazione.
  - **Progettazione e scelta dei componenti**: Ricerca di componenti riutilizzabili o progettazione di nuovi; il modello ottenuto può generare automaticamente l'implementazione.
- Lo sviluppo del programma segue la progettazione del sistema. Per sistemi di sicurezza critica, la progettazione è dettagliata in profondità prima dell'implementazione. È più comune che progettazione e sviluppo siano intrecciati.

3. **Convalida del software** (Verifica e convalida):
   - Attività intesa a dimostrare che un sistema è conforme alle sue specifiche e soddisfa le aspettative del cliente.
   - Il **test dei programmi**, eseguendo il sistema con dati di prova simulati, è la tecnica principale di convalida.
   - La convalida può richiedere attività di controllo, ispezione e revisione ad ogni stadio del processo di sviluppo.
   - Le **attività principali** sono:
     - **Test dei componenti** (o delle unità): componenti testati singolarmente dagli sviluppatori, con strumenti di automazione (es. JUnit).
     - **Test del sistema**: trova errori causati da interazioni impreviste tra componenti e problemi di interfaccia; verifica le proprietà significative del sistema.
     - **Test del cliente**: il sistema viene testato dal cliente con i suoi dati, funzionale all'approvazione per la messa in uso.
4. **Evoluzione del software** (Manutenzione del software):
   - Il software può essere modificato in qualsiasi momento, durante e dopo lo sviluppo; grandi modifiche sono più economiche di quelle hardware.
   - La distinzione storica tra sviluppo e manutenzione sta diventando irrilevante. L'ingegneria del software è vista come un unico processo evolutivo, in cui *il software*

viene modificato continuamente per adeguarsi ai cambiamenti dei requisiti e delle necessità dei clienti.

Gestione dei Cambiamenti

I cambiamenti sono inevitabili nella maggior parte dei progetti. È essenziale che il processo consenta di adeguare il software ai cambiamenti in corso. Per ridurre i costi di rilavorazione, si usano due approcci correlati:

- **Anticipazione dei cambiamenti**: il processo software include attività che possono anticipare o predire possibili variazioni.
- **Tolleranza ai cambiamenti**: il processo e il software sono progettati in modo che le modifiche possano essere facilmente apportate al sistema (spesso implica sviluppo incrementale).

Due metodi per far fronte ai cambiamenti dei requisiti di sistema sono:

- **Prototipazione del sistema**: Il sistema (o parte) viene sviluppato rapidamente per verificare i requisiti del cliente e la fattibilità delle scelte di progettazione, consentendo agli utenti di provarlo prima della consegna.
- **Consegna incrementale**: Vengono consegnati al cliente incrementi del sistema per commenti e prove, evitando l'approvazione prematura dei requisiti per l'intero sistema e consentendo modifiche a costi più bassi in successivi incrementi. Il **refactoring** (miglioramento della struttura e dell'organizzazione di un programma) è un altro importante meccanismo che supporta la tolleranza ai cambiamenti.

Modelli di Processi Software

Differenti tipi di sistemi richiedono differenti approcci di sviluppo. Ad esempio, il software real-time deve essere specificato completamente prima dello sviluppo, mentre nei sistemi di commercio elettronico le specifiche e il programma sono sviluppati assieme.

Esempi di processi software includono il **modello a cascata**, **Unified Process** (**UP**), **Scrum**, il **modello di sviluppo a spirale**, lo **sviluppo rapido di applicazione** (**RAD**) e l'**Extreme Programming** (**XP**). Le quattro attività fondamentali (specifica, sviluppo, convalida ed evoluzione) sono organizzate diversamente: in sequenza nel modello a cascata, o intrecciate nel modello di sviluppo incrementale.

Un **modello di processo software** o **paradigma di processo** è una rappresentazione semplificata di un processo software, una struttura di processo da estendere e adattare. I principali sono:

- **Modello a cascata**: Le attività di processo fondamentali (specifica, sviluppo e convalida) sono rappresentate come fasi distinte e sequenziali.

- **Sviluppo incrementale**: Intreccia le attività di specifica, sviluppo e convalida. Il sistema è sviluppato come una serie di versioni (incrementi), ciascuna aggiungendo nuove funzionalità.
- **Integrazione e configurazione**: Si basa sulla disponibilità di componenti o sistemi riutilizzabili. Il processo di sviluppo configura e integra questi componenti in una nuova disposizione.

Non esiste un modello di processo universale. Il processo appropriato dipende dai requisiti dei clienti, dalle politiche normative, dall'ambiente e dal tipo di software.

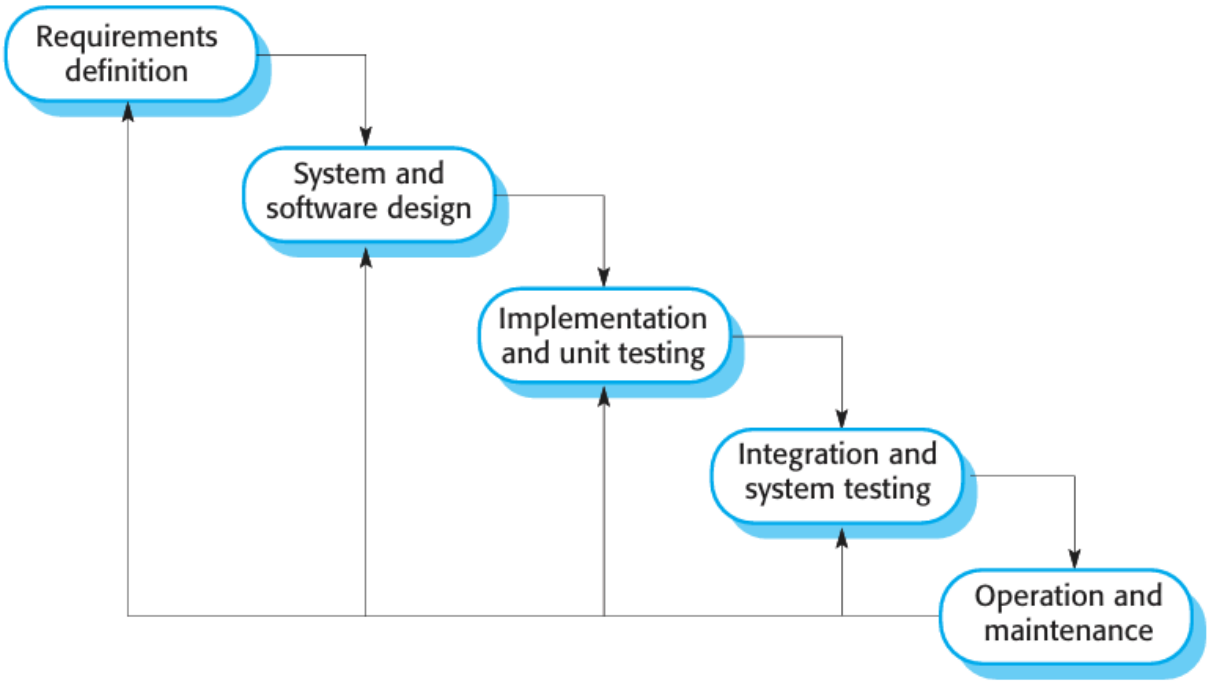
- Il **software a sicurezza critica** di solito usa il modello a cascata, richiedendo molte analisi e documentazione.
- I **prodotti software** sono sempre più sviluppati con un modello di processo incrementale.
- I **sistemi aziendali** sono sempre più sviluppati configurando e integrando sistemi esistenti.

Gran parte dello sviluppo pratico combina caratteristiche di diversi modelli. Per l'ingegneria di grandi sistemi, ha senso combinare i migliori approcci, dato che è necessario avere requisiti essenziali per progettare un'architettura software. I sottosistemi di un sistema più grande possono usare approcci diversi: parti chiare con il modello a cascata, parti difficili da specificare preventivamente con un approccio incrementale.

Modello a Cascata (o Sequenziale)

È basato su uno svolgimento sequenziale delle diverse attività di sviluppo del software.

- All'inizio del progetto vengono definiti in dettaglio tutti i **requisiti** e un **piano temporale dettagliato** delle attività.
- Si prosegue con la **modellazione** (**analisi e progettazione**), quindi viene creato un **progetto completo del software**.
- A questo punto inizia la **programmazione del sistema software**, seguita da **verifica** e **rilascio** (e successiva manutenzione).



Questo modello fu derivato dai modelli usati nell'ingegnerizzazione di grandi sistemi militari ed era comune fino a pochi anni fa. È un esempio di processo guidato da un piano, dove si pianificano tutte le attività prima di iniziare lo sviluppo.

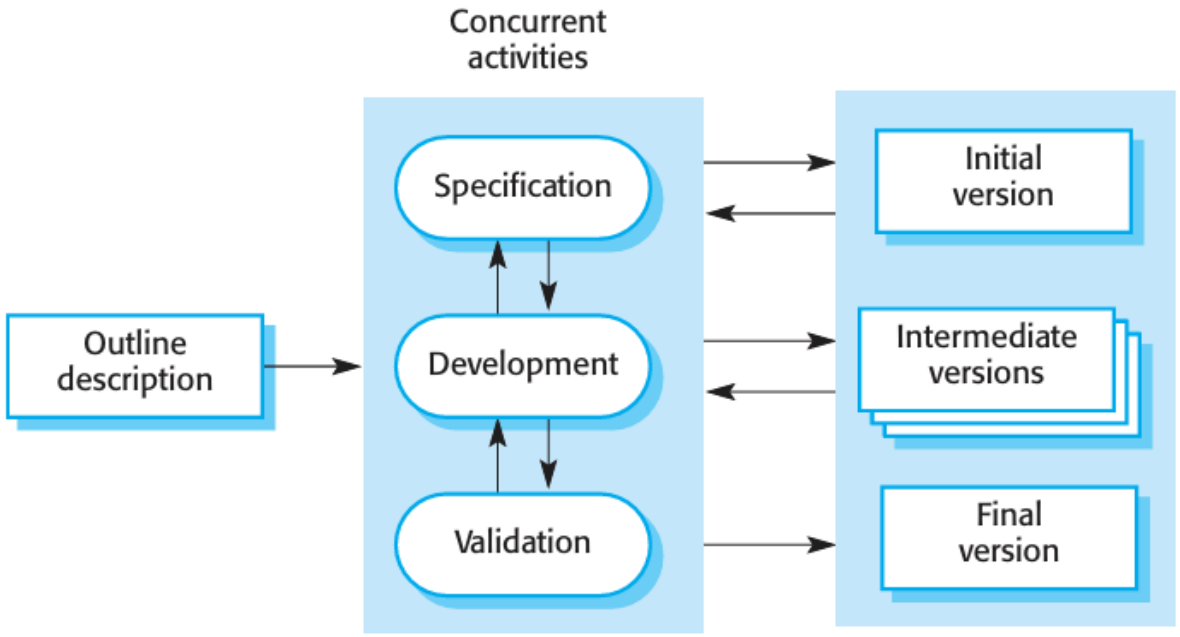
È appropriato per:

- Sistemi integrati dove il software deve interfacciarsi con sistemi hardware.
- Sistemi critici dove è richiesta un'analisi approfondita della sicurezza e della protezione.
- Grandi sistemi software parte di sistemi più complessi sviluppati da più società.

Non è appropriato quando è consentita una comunicazione informale nei team o quando i requisiti cambiano rapidamente. Una variante è lo sviluppo di sistemi formali, con un modello matematico raffinato in codice eseguibile, molto costoso e per sistemi con severi requisiti di sicurezza/affidabilità/protezione.

Sviluppo Incrementale

Si basa sull'idea di sviluppare un'implementazione iniziale, esporla agli utenti e perfezionarla attraverso molte versioni. Le attività di specifica, sviluppo e convalida sono **intrecciate** anziché separate, con feedback veloci.



Lo sviluppo incrementale è l'approccio più comune per sistemi applicativi e prodotti software. Può essere **plan-driven** (incrementi stabiliti in anticipo) o **agile** (incrementi iniziali identificati, i successivi dipendono da avanzamento e priorità del cliente).

È **migliore del modello a cascata** per sistemi i cui requisiti possono cambiare durante lo sviluppo (es. molti sistemi aziendali e prodotti software). Riflette il modo in cui risolviamo i problemi, arrivando alla soluzione per passaggi successivi e correggendo errori. È più economico e semplice apportare modifiche durante lo sviluppo. Ogni incremento o versione incorpora funzionalità richieste dal cliente, e gli incrementi iniziali includono le funzionalità più importanti e urgenti per una valutazione precoce.

Vantaggi:

- Costo di implementazione delle modifiche ai requisiti ridotto.
- Più facile ottenere feedback del cliente.
- Possibile consegnare in anticipo una versione utilizzabile del software.

Svantaggi:

- Il processo non è visibile; i manager necessitano consegne regolari per misurare i progressi.
- La struttura dei sistemi tende a degradarsi con nuovi incrementi (i metodi agili suggeriscono il refactoring costante per ridurre il degrado strutturale e la complicazione del codice).
- Nei sistemi grandi e complessi con team diversi, la pianificazione anticipata è necessaria per definire chiaramente le responsabilità.

Sviluppo Incrementale vs a Cascata

- **Sviluppo incrementale**: Passi di sviluppo brevi e feedback per chiarire i requisiti.
- Modello a cascata: Grande lavoro speculativo iniziale.

Secondo studi (Larman, Basili, 2003), i metodi iterativi sono associati a percentuali di successo e produttività più elevate e a minori difetti. Il metodo a cascata è una pratica mediocre per la maggior parte dei progetti software, caratterizzato da minore produttività, maggiori difetti e stime iniziali di tempi e costi notevolmente diverse dai valori finali. Questo perché il modello a cascata presuppone che le specifiche siano prevedibili e stabili, definibili correttamente fin dall'inizio, con un basso tasso di cambiamenti.

Integrazione e Configurazione

È comune riutilizzare il software esistente, modificandolo e integrandolo nel nuovo sistema. Dal 2000, i processi di sviluppo che sfruttano il riutilizzo si stanno diffondendo. Tipi di software riutilizzabile includono:

- Sistemi applicativi indipendenti configurati per ambienti specifici (sistemi generici adattati).
- Collezioni di oggetti sviluppate come componenti o pacchetti da integrare tramite framework (es. JavaSpring).
- Servizi web conformi agli standard, disponibili per siti remoti su Internet.

Le **fasi principali** di questo modello sono:

- Specifica dei requisiti iniziali del sistema.
- Ricerca e valutazione del software che può fornire le funzionalità richieste.
- Perfezionamento dei requisiti utilizzando le informazioni sul software trovato.
- Configurazione del sistema delle applicazioni per creare il nuovo sistema.
- Adattamento e integrazione dei componenti se un sistema pronto all'uso non è disponibile.

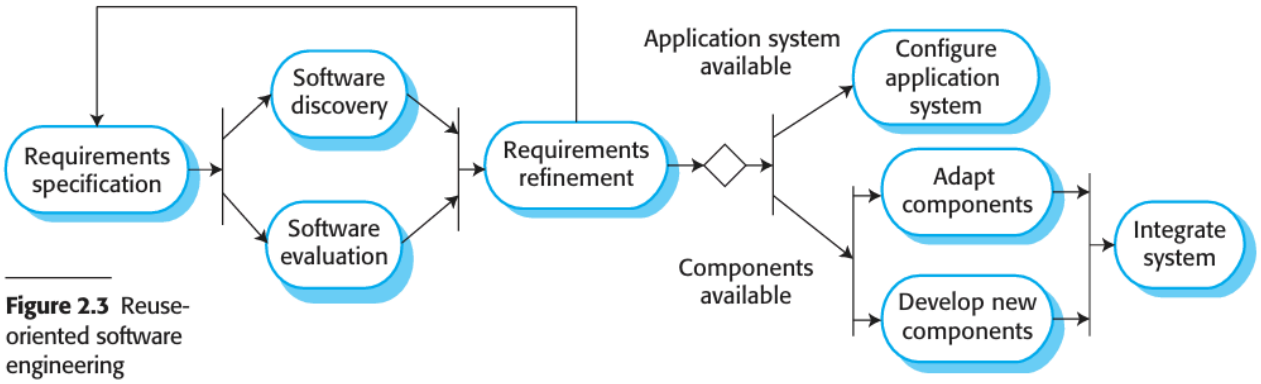


Figure 2.3 Reuse-oriented software engineering

Vantaggi:

- Riduce la quantità di software da sviluppare, diminuendo costi e rischi e portando a consegne più veloci.

Svantaggi:

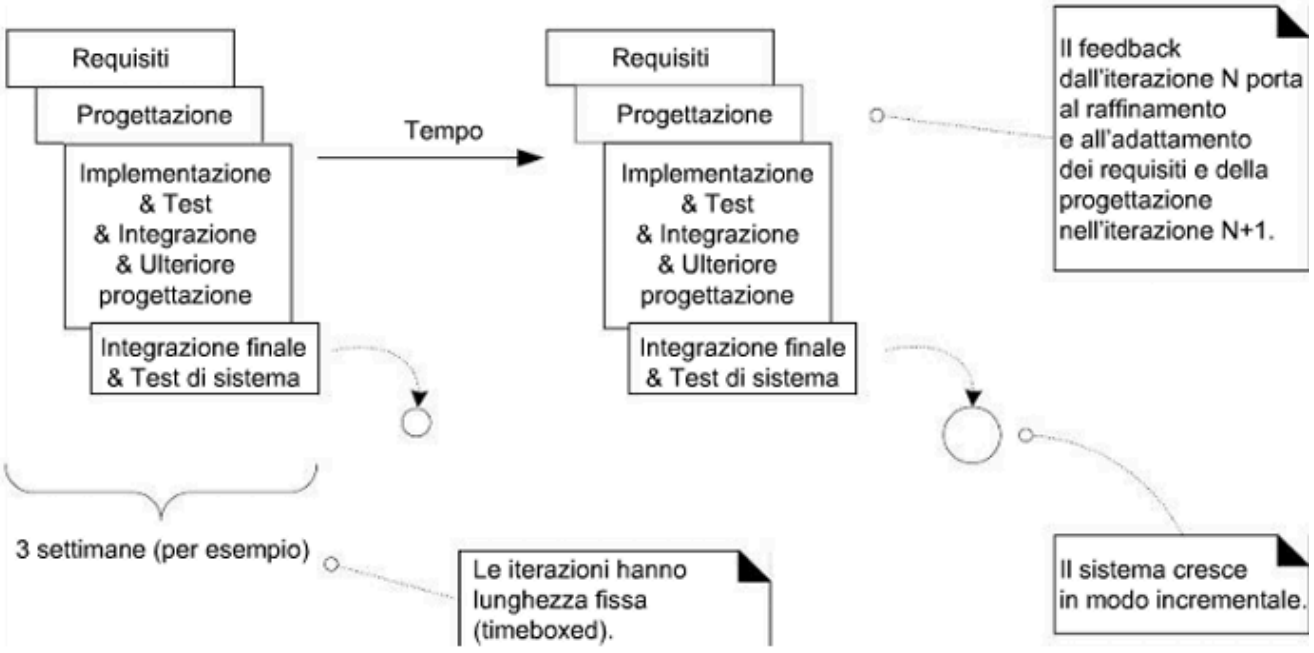
- Compromessi inevitabili nei requisiti; perdita di controllo sull'evoluzione del sistema, poiché nuove versioni dei componenti riutilizzati non sono sotto il controllo dell'organizzazione che li usa.

Sviluppo Iterativo ed Evolutivo

Lo sviluppo incrementale, iterativo ed evolutivo comporta fin dall'inizio la programmazione e il test di un sistema software. Lo sviluppo inizia prima che tutti i requisiti siano stati definiti in modo dettagliato, e il feedback è utilizzato per chiarire e migliorare le specifiche in evoluzione.

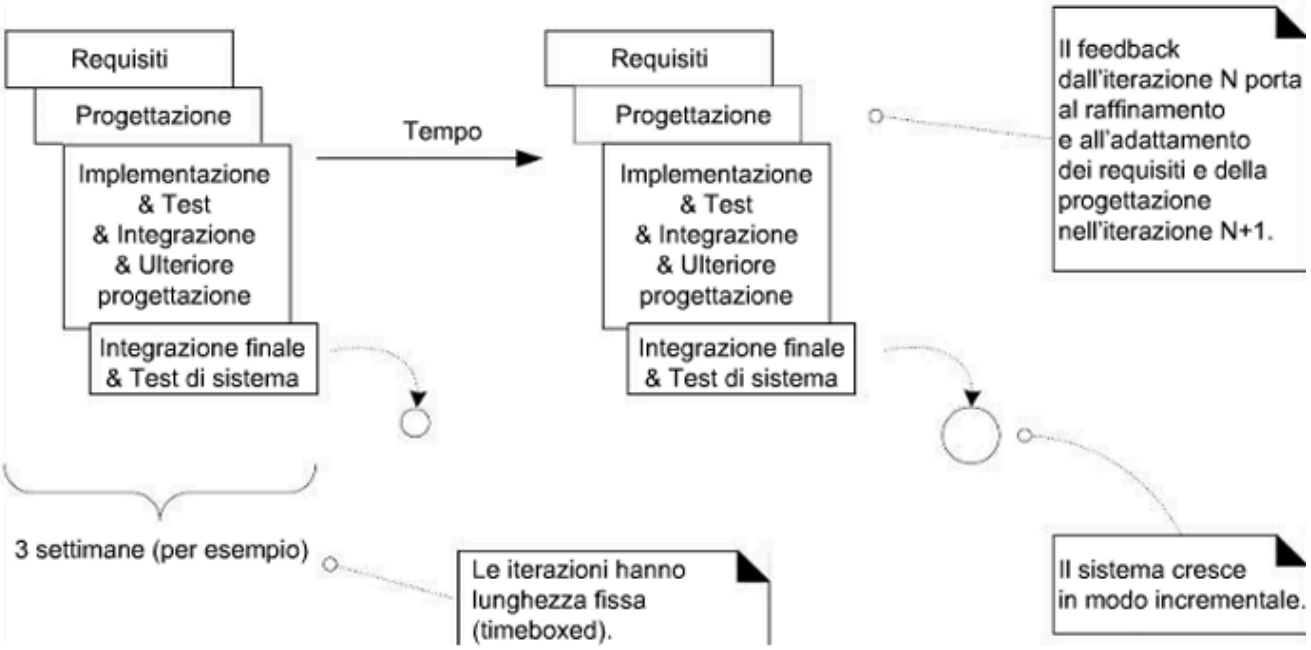
Nell'approccio iterativo:

- Lo sviluppo è organizzato in una serie di mini-progetti brevi, di lunghezza fissa, chiamati **iterazioni**.
- Il risultato di ciascuna iterazione è un **sistema eseguibile**, testato e integrato, anche se **parziale**.
- Ciascuna iterazione comprende le proprie attività di analisi dei requisiti, progettazione, implementazione e test.

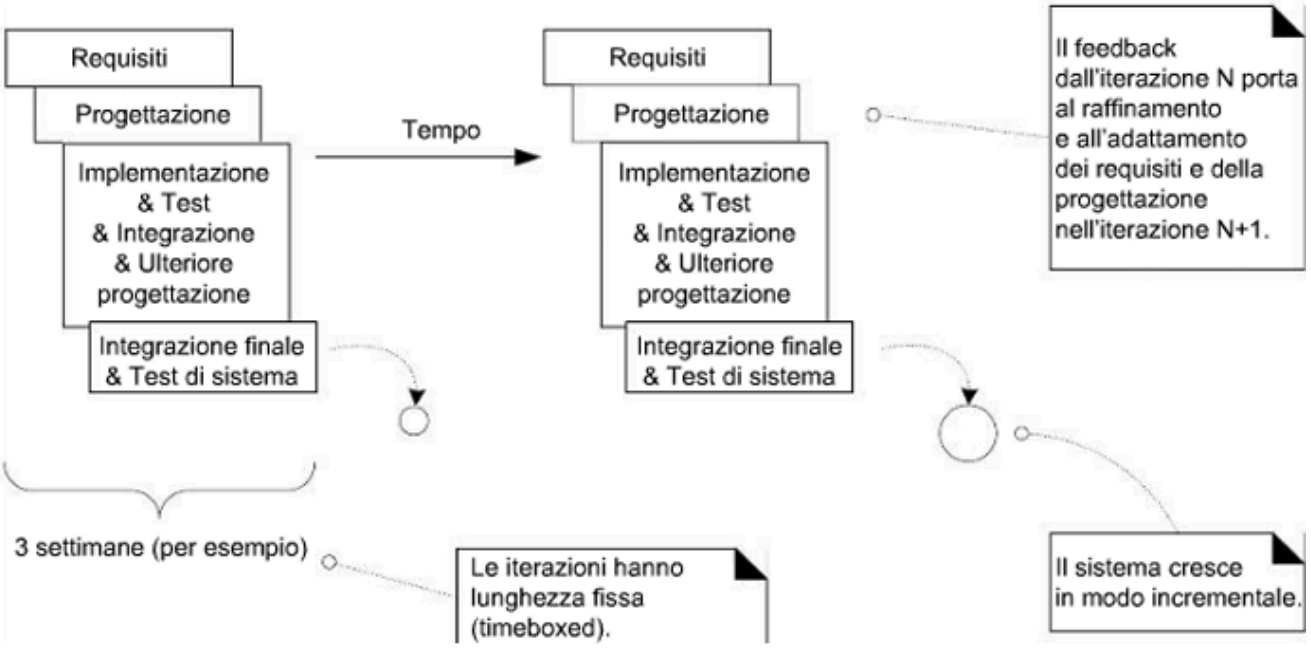


Questo sviluppo è:

- **Incrementale**: Il sistema cresce incrementalmente nel tempo, iterazione dopo iterazione.
- **Evolutivo**: Il feedback e l'adattamento fanno evolvere le specifiche e il progetto.



Nel processo iterativo, non c'è fretta di iniziare la codifica, né una fase di progettazione lunga che perfeziona tutti i dettagli prima della programmazione. Il risultato di un'iterazione non è un prototipo ma un **sottoinsieme del sistema finale**. Ogni iterazione comporta la scelta di un piccolo sottoinsieme di requisiti, una rapida progettazione, implementazione e test, consentendo feedback rapidi da parte di utenti, sviluppatori e tester, e l'opportunità di modificare o adattare i requisiti e il progetto. Attraverso il feedback iterativo e l'adattamento, il sistema evolve e converge verso i requisiti corretti e il progetto più appropriato.



Esempi di processi di sviluppo iterativo ed evolutivo sono **Unified Process** (UP), **Extreme Programming** (XP) e **Scrum**.

Vantaggi:

- Riduzione precoce dei rischi maggiori (tecnici, requisiti, obiettivi, usabilità).
- Progresso visibile fin dall'inizio.

- Feedback precoce e coinvolgimento dell’utente e adattamento.
- Gestione della complessità (evita la "paralisi da analisi").
  È fondamentale un'iterazione da due a sei settimane, con passi piccoli, feedback rapido e adattamento. Un'iterazione di lunghezza fissa è detta **timeboxed**.

Storia dello Sviluppo Iterativo ed Evolutivo

- Alla fine degli anni Cinquanta, al *progetto spaziale Mercury*, fu applicato lo **sviluppo incrementale, iterativo ed evolutivo** (**IDD**) piuttosto che il metodo a cascata. Il programma Mercury fu il primo programma spaziale statunitense con equipaggio.
- Questo approccio fu usato anche nel *Trident nuclear programme* (missili balistici su sottomarini nucleari).
- Negli anni Settanta, il software di controllo di volo dello *Space Shuttle* fu costruito con *17 iterazioni di circa quattro settimane l'una*.
- IBM, nel 1968, produsse il primo documento che promuoveva lo sviluppo iterativo.

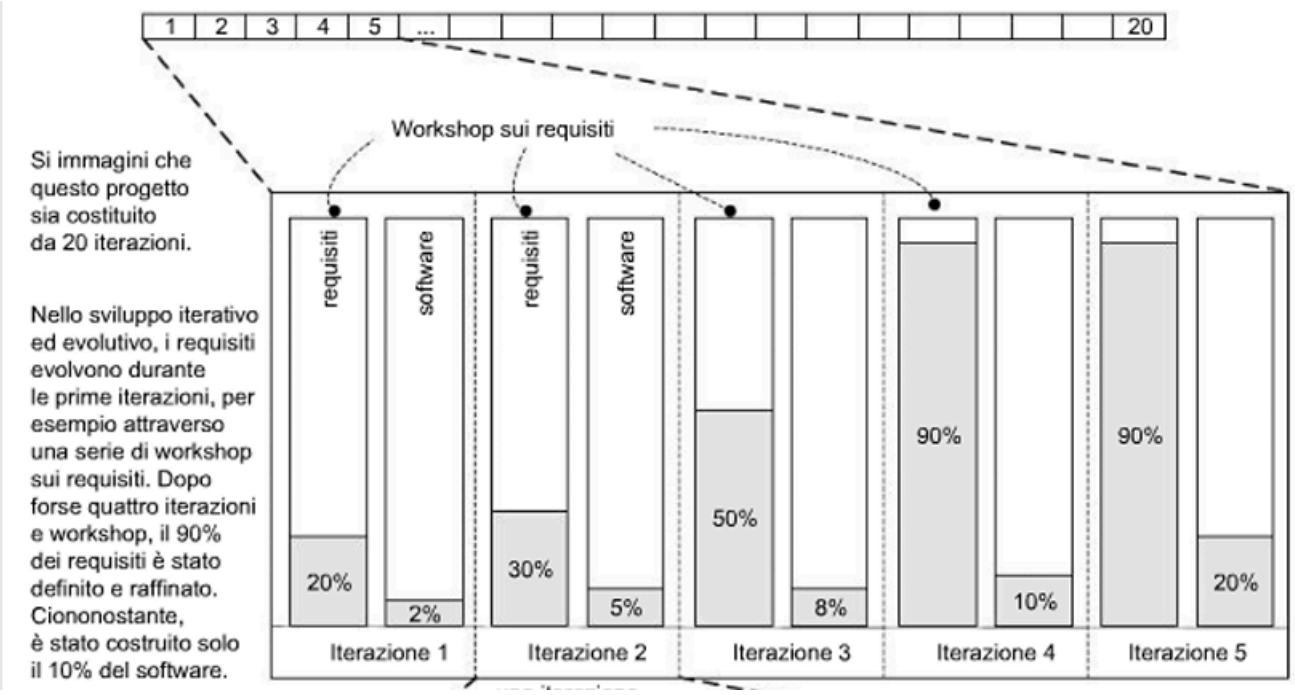
Un esempio di progetto iterativo (UP), composto da 20 iterazioni prima del rilascio al cliente:

1. Prima dell'iterazione 1, si tiene il primo workshop dei requisiti timeboxed, per esempio di due giorni esatti. Sono presenti i responsabili dell'organizzazione e dello sviluppo (compreso il chief architect, il capo architetto).
   - La mattina del primo giorno si esegue un'analisi dei requisiti di alto livello, per esempio identificando solo i nomi dei casi d'uso e le caratteristiche, oltre ai requisiti non funzionali più importanti. L'analisi non sarà perfetta.
   - Si chiede al chief architect e ai responsabili dell'organizzazione di scegliere da questo elenco di alto livello il 10% (per esempio il 10% dei 30 nomi dei casi d'uso identificati) che possegga una miscela delle seguenti tre qualità: 1) significatività dal punto di vista dell'architettura (per implementarlo, richiede di progettare, costruire e testare il nucleo dell'architettura); 2) elevato valore di business (si tratta di caratteristiche importanti per l'organizzazione); 3) rischio elevato (per esempio che "sia in grado di gestire 500 transazioni concorrenti"). Per esempio, procedendo in questo modo vengono identificati tre casi d'uso: UC2, UC11 e UC14.
   - Per il rimanente giorno e mezzo, si esegue un'analisi intensa e dettagliata dei requisiti funzionali e non funzionali per questi tre casi d'uso. Al termine, il 10% dei requisiti è stato analizzato in profondità, e il rimanente 90% solo ad alto livello.
2. Prima dell'iterazione 1, si tiene una riunione di pianificazione dell'iterazione in cui si sceglie un sottoinsieme di requisiti da UC2, UC11 e UC14 su cui fare progettazione, implementazione e test entro un tempo specificato (per esempio, un'iterazione timeboxed di tre settimane). Si noti che potrebbe non essere possibile sviluppare interamente i tre casi d'uso scelti, perché questo potrebbe richiedere troppo lavoro. Dopo aver scelto un sottoinsieme specifico di obiettivi, questi vanno suddivisi in un insieme di compiti più dettagliati da svolgere nell'iterazione, anche sulla base delle indicazioni del team di sviluppo.
3. Si esegue l'iterazione 1 in tre settimane (occorre attenersi al timebox scelto).
   - I primi due giorni, gli sviluppatori e gli altri fanno modellazione e progettazione a coppie, abbozzando diagrammi UML su più lavagne (abbozzando eventualmente anche altri tipi di modelli) nella stanza comune del progetto, guidati e istruiti dal chief architect.

- Gli sviluppatori si levano il "cappello da modellatore" e si mettono il "cappello da programmatore", passando dunque dalla modellazione alla programmazione. Iniziano a programmare, fare test e integrazione in modo continuo nelle settimane rimanenti dell'iterazione, utilizzando i modelli abbozzati come punto di partenza e di ispirazione, sapendo però che i modelli sono parziali e spesso solo approssimativi.
- Viene eseguito un grosso lavoro di test: unitari, di accettazione, di carico, di usabilità e così via.
- Una settimana prima della fine, si chiede al team se gli obiettivi originari dell'iterazione possono essere raggiunti; in caso contrario, si riduce la portata dell'iterazione, rimettendo gli obiettivi secondari nell'elenco delle cose da fare nelle iterazioni successive.
- Il martedì dell'ultima settimana il codice viene congelato; tutto il codice deve essere caricato, integrato e testato per creare la baseline dell'iterazione.
- Il mercoledì mattina viene fatta una dimostrazione del sistema parziale alle parti interessate esterne, per rendere visibili i progressi iniziali. Alle parti interessate viene richiesto un feedback.

4. Si esegue il secondo workshop sui requisiti verso la fine dell'iterazione 1, per esempio l'ultimo mercoledì e giovedì. Tutto il materiale dell'ultimo workshop viene rivisto e raffinato. Viene quindi scelto un altro 10% o 15% dei casi d'uso significativi dal punto di vista dell'architettura e di elevato valore di business, lo si analizza in dettaglio per uno o due giorni. Al termine, probabilmente il 25% dei casi d'uso e dei requisiti non funzionali sarà stato scritto in modo dettagliato. Ma non sarà perfetto.
5. Il venerdì mattina si tiene un'altra riunione di pianificazione dell'iterazione, per l'iterazione successiva.
6. Si esegue l'iterazione 2, con gli stessi passi.
7. Si ripete il tutto, per quattro iterazioni e cinque workshop dei requisiti, di modo che alla fine dell'iterazione 4 probabilmente l'80% o 90% dei requisiti sia stato scritto in modo dettagliato; solo il 10% del sistema sarà stato implementato.
    - Si noti che questo grande e dettagliato insieme di requisiti è basato su feedback ed evoluzione, ed è pertanto di qualità molto superiore rispetto a delle specifiche a cascata puramente speculative.

8. Si è probabilmente solo al 20% della durata dell'intero progetto. In termini di UP, questa è la fine della **fase di elaborazione**. A questo punto, vengono fatte delle stime più dettagliate dei tempi e dei costi, con riferimento ai requisiti di qualità elevata in possesso. Grazie all'indagine significativa e realistica fatta, basata su programmazione, test e feedback iniziali, le stime di ciò che si può fare e di quanto tempo occorrerà sono molto più affidabili.
9. Da questo punto in poi, è poco probabile che si tengano altri workshop sui requisiti; i requisiti sono stabili, anche se non vanno mai considerati completamente congelati. Si continua con una serie di iterazioni di tre settimane, scegliendo gli obiettivi di lavoro successivi in modo adattivo in ciascuna riunione di pianifica-



Pianificazione Guidata dal Rischio e dal Cliente

Nei processi iterativi, il piano di lavoro è stabilito per una sola iterazione alla volta, una **pianificazione iterativa** o **adattativa**:

- **UP**: Alla fine di ciascuna iterazione per la successiva.
- **Scrum**: All'inizio di ciascuna iterazione per il piano della corrente.

Gli obiettivi dell'iterazione non vengono cambiati, perché le iterazioni sono brevi e il feedback frequente, consentendo al team di concentrarsi sul lavoro stabilito.

La pianificazione è **guidata dal rischio** e **guidata dal cliente**. Le iterazioni iniziali sono scelte per:

- identificare e attenuare i rischi maggiori.
- costruire e rendere visibili le caratteristiche a cui il cliente tiene di più.
- stabilizzare il nucleo dell'architettura del software (che è un rischio molto alto).

Sviluppo Agile del Software

Contesto

Le aziende moderne operano in un ambiente globale in rapido cambiamento, devono cogliere nuove opportunità e rispondere a nuovi mercati, condizioni economiche variabili e concorrenza.
Il software è essenziale per le operazioni aziendali e deve trarre vantaggio da queste opportunità.
La rapidità dello sviluppo e della consegna è il requisito più critico per la maggior parte dei sistemi software aziendali.

Spesso è praticamente impossibile ottenere un insieme completo di requisiti stabili.
I requisiti cambiano perché i clienti non possono prevedere l'impatto di un sistema sulle pratiche operative, le interazioni con altri sistemi o quali operazioni utente automatizzare. I requisiti reali diventano chiari solo dopo la consegna e l'uso del sistema, e fattori esterni possono indurre ulteriori modifiche.

Negli anni '80 e '90, si puntava su un'attenta pianificazione, garanzia di qualità formale, metodi di analisi e progettazione rigorosi (per grandi progetti aerospaziali e governativi). Tuttavia, per i sistemi di piccole e medie dimensioni, gli overhead erano troppo alti.

Negli anni '90, sono emersi i **metodi agili**, che **si concentrano sul software stesso** piuttosto che sulla progettazione e documentazione.
I metodi agili sono **particolarmente indicati per sviluppare applicazioni con requisiti che cambiano rapidamente**, e sono **ideati per consentire una consegna rapida ai clienti**, che possono proporre requisiti nuovi o modificati da includere in successive iterazioni.

Principi e Pratiche

I metodi per lo sviluppo agile applicano di solito lo **sviluppo iterativo ed evolutivo**. L'enfasi è su una **risposta rapida e flessibile ai cambiamenti** (agilità), con **iterazioni brevi** e raffinamento evolutivo di piani, requisiti e progetto.

- **Agile Modeling**: Lo scopo della modellazione (es. abbozzare UML) è principalmente la **comprensione** e la **comunicazione**, non la documentazione. Si esplorano rapidamente le alternative e il percorso verso un buon progetto orientato agli oggetti.

I metodi agili suggeriscono che il software sia **sviluppato e consegnato in modo incrementale**.

I principi agili sono utili per due tipi di sviluppo di sistemi:

- Sviluppo di prodotti di piccole e medie dimensioni.
- Sviluppo personalizzato di sistemi all’interno di un’organizzazione con un chiaro impegno del cliente nel processo di sviluppo.

Principi Agili

Adottare un metodo agile non significa evitare del tutto la modellazione, ma usarla per facilitare comprensione e comunicazione.

Non si deve modellare per eseguire l'intera progettazione del software; i problemi di progettazione semplici sono rimandati alla fase di programmazione e risolti durante la programmazione e i test.

Va utilizzato lo strumento più semplice possibile che aiuti la creatività con il minimo dispendio di energie (es. abbozzi UML alla lavagna).

La modellazione non va fatta da soli, ma a coppie (o in tre), per scoprire, capire e condividere.

Solo il codice verificato dimostra il vero progetto; i diagrammi precedenti sono suggerimenti incompleti, da considerare come esplorazioni "usa e getta".

La modellazione per la progettazione OO dovrebbe essere eseguita dagli stessi sviluppatori che si occuperanno della programmazione; creare modelli da passare ad altri programmatori segue pratiche a cascata, opposte alla filosofia agile.

Extreme Programming (XP)

È uno dei metodi agili più noti.

- Lo sviluppo incrementale è supportato tramite **piccole e frequenti release** del sistema. I requisiti si basano su scenari semplici utilizzati per decidere le funzionalità da includere in un incremento.
- Il coinvolgimento dell’utente è garantito dalla **costante presenza del cliente** nel team di sviluppo.
- Le persone sono supportate dalla **programmazione in coppia**, dal processo collettivo del codice del sistema e da uno sviluppo sostenibile che non richiede orari di lavoro eccessivamente lunghi.
- Le modifiche sono supportate da **release regolari**, **sviluppo preceduto da test**, **refactoring** per evitare la degenerazione del codice e integrazione continua di nuove funzionalità.
- Il mantenimento della semplicità è supportato dal **refactoring costante** (migliorando la qualità del codice) e dall'uso di **progetti semplici** che non prevedono necessariamente modifiche future.

Pratiche Innovative dei Metodi Agili

- **Storie utente**: Scenari d’uso in cui potrebbe trovarsi un utente del sistema, discussi a stretto contatto con il cliente.
- **Refactoring**: Non ha senso progettare per il cambiamento, le modifiche previste spesso non si avverano. Le modifiche vengono sempre apportate al codice in sviluppo, che è

costantemente rifattorizzato. Lo sviluppo incrementale deteriora la struttura del software, e il refactoring la migliora.
- Sviluppo con **test iniziali** (**Test-Driven Development - TDD**): Lo sviluppo non può procedere finché tutti i test non sono stati superati; è uno sviluppo guidato dai test.
- **Programmazione a coppie**: Due programmatori siedono alla stessa postazione per sviluppare il software, supportando la proprietà e responsabilità comune del sistema, la revisione informale e incentivando il refactoring.

Scrum: gestione Agile della Progettazione

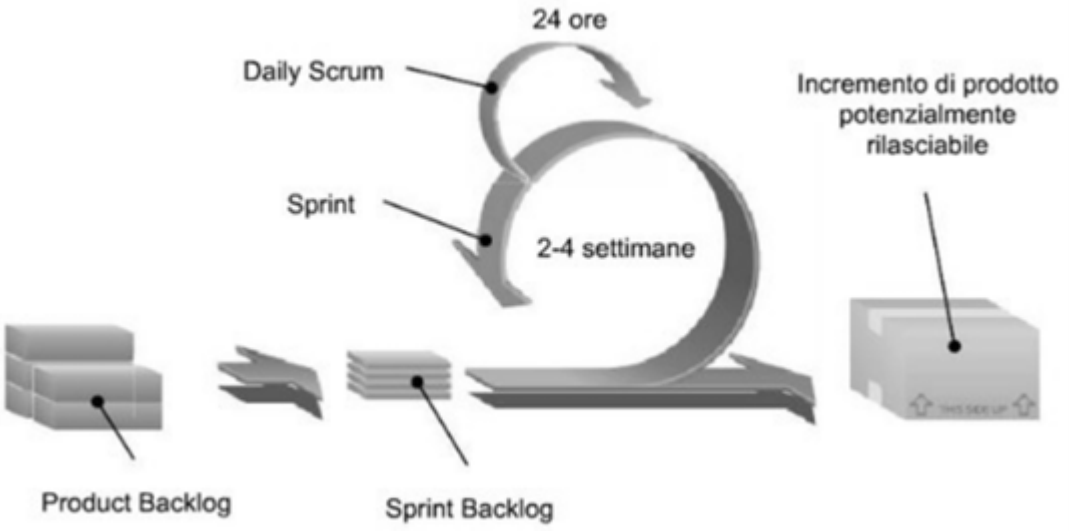
Scrum è un framework per organizzare progetti agili e fornire visibilità esterna su ciò che sta accadendo, occupandosi dell'organizzazione del lavoro e della gestione dei progetti. È un **approccio iterativo e incrementale**, dove **ogni iterazione ha una durata fissa** chiamata **Sprint**, e non viene mai estesa.

Sono presenti tre ruoli:

- **Product Owner**: Definisce le caratteristiche del prodotto software e le priorità tramite il **Product Backlog** (un elenco di voci, funzionalità e requisiti).
- **Development Team**: Possiede le competenze necessarie per sviluppare il software.
- **Scrum Master**: Aiuta il gruppo ad apprendere e applicare Scrum, non è il manager del Development Team, ma un istruttore e una guida.

All'inizio del progetto, il Product Backlog descrive tutte le caratteristiche del prodotto e viene aggiornato iterazione dopo iterazione.
Il Development Team seleziona dal Product Backlog un insieme di voci da sviluppare durante uno **Sprint** (***Sprint Goal***), compilando lo **Sprint Backlog** (compiti dettagliati).
Il risultato di ciascuno Sprint *deve essere un prodotto software funzionante*, chiamato "*incremento di prodotto potenzialmente rilasciabile*", integrato, verificato e documentato per l'utente finale.

Nella **Sprint Review**, il Product Owner e il Development Team presentano l'incremento di prodotto agli stakeholder, lo dimostrano, ottengono feedback e decidono cosa fare nel successivo Sprint.

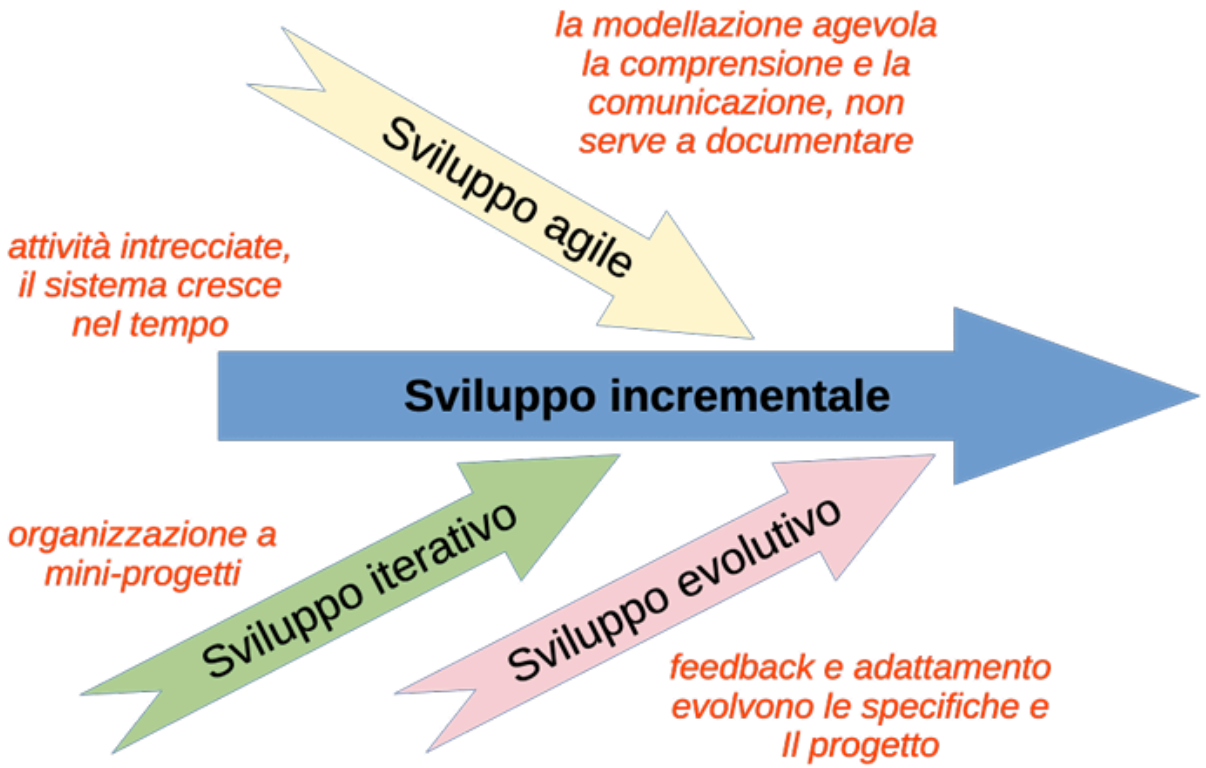


> **Nota**
>
> La caratteristica distintiva di Scrum è l'enfasi sull’adozione di team auto-organizzati e auto-gestiti.
>
> Scrum è basato su un insieme di elaborati ed eventi che rendono visibili gli obiettivi e il progetto delle iterazioni e favoriscono un adattamento evolutivo del processo.

Problemi Pratici con i Metodi Agili

- L'informalità dello sviluppo agile è incompatibile con l'approccio legale alla definizione dei contratti tipico delle grandi società.
- Sono più indicati per lo sviluppo di nuovo software, non per la manutenzione (che costituisce la maggior parte dei costi nelle grandi società).
- Sono ideati per piccoli team fisicamente vicini, mentre la maggior parte dello sviluppo software oggi coinvolge team distribuiti a livello globale.

> La scalabilità dei metodi agili richiede l'integrazione di pratiche basate sui piani, come più documentazione, più rappresentanti del cliente, strumenti comuni per i team e allineamento delle release.


la modellazione agevola
la comprensione e la
comunicazione, non
serve a documentare
Sviluppo agile
attività intrecciate,
il sistema cresce
nel tempo
Sviluppo incrementale
organizzazione a
mini-progetti
Sviluppo iterativo
Sviluppo evolutivo
feedback e adattamento
evolvono le specifiche e
Il progetto