

01. Processi per lo sviluppo software

Sviluppo di Applicazioni Software

Ferruccio Damiani

a.a. 2023/24

Università degli Studi di Torino - Dipartimento di Informatica

Attenzione!



©2024 Copyright for this slides by Ferruccio Damiani. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

<https://creativecommons.org/licenses/by/4.0/>

Si noti che

questi lucidi sono basati sul libro di testo del corso “C. Larman, *Applicare UML e i Pattern*, Pearson, 2016”; sul libro “I. Sommerville, *Ingegneria del Software*, Pearson, 2017”; e sul materiale fornito da Matteo Baldoni (docente del corso A).

Table of contents

1. Processi per lo sviluppo software
2. Modelli di processi software
3. Sviluppo iterativo ed evolutivo
4. Sviluppo agile del software

Processi per lo sviluppo software

Software di qualità

- Il software non è un semplice programma o un gruppo di programmi
- Il software include anche tutta la documentazione elettronica che serve agli utenti dei sistemi, agli sviluppatori e ai responsabili della garanzia della qualità
- Le caratteristiche essenziali di un prodotto software sono:
 - la **mantenibilità**: il software dovrebbe essere scritto in modo da evolversi in rapporto alle nuove richieste dei clienti
 - la **fidatezza**: un software fidato non dovrebbe causare danni fisici o economici nel caso di malfunzionamento del sistema
 - l'**efficienza**: il software non dovrebbe fare un uso dispendioso delle risorse del sistema, come la memoria o i cicli di processore
 - l'**accettabilità**: il software deve essere accettabile per il tipo di utenti per i quali è progettato, ossia comprensibile, usabile e compatibile con gli altri sistemi che essi usano

In generale, un processo

descrive **chi fa che cosa, come e quando** per raggiungere un obiettivo.

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.

Attività di processo

Quattro **attività (di processo)** fondamentali sono comuni a tutti i processi software:

1. **Specifiche del software**: clienti e sviluppatori definiscono le funzionalità e i vincoli operativi del software da produrre
2. **Sviluppo del software**: viene progettato e sviluppato il software
3. **Convalida del software**: il software viene convalidato per garantire ciò che il cliente richiede
4. **Evoluzione del software**: il software viene modificato per soddisfare eventuali cambiamenti dei requisiti del cliente e del mercato

Specifica del software

E' detta anche **ingegneria dei requisiti**, è l'attività per **capiere e definire** quali servizi sono richiesti dal sistema, e per **identificare i vincoli** all'operatività e allo sviluppo del sistema.

È uno stadio particolarmente critico del processo software, poiché gli errori in questa fase portano inevitabilmente a problemi successivi nella progettazione e implementazione del sistema.

Le fasi principali sono:

- **Deduzione e analisi dei requisiti**, osservazione di sistemi esistenti, discussione con i possibili utenti, analisi dei task, ecc.
- **Specificazione dei requisiti**, tradurre le informazioni raccolte in un documento che definisce un insieme di requisiti
- **Convalida dei requisiti**, controlla che i requisiti siano realistici, coerenti e completi

o **progettazione e implementazione del software**, è l'attività di **conversione** delle specifiche in un sistema eseguibile da consegnare al cliente.

Nelle metodologie agili (ne parliamo a breve) **la progettazione e l'implementazione sono intrecciate** e non producono documenti formali, il progetto viene registrato informalmente sulle lavagne e sui portatili dei programmati.

Il progetto è una descrizione della struttura del software che si deve implementare dei modelli e delle strutture di dati usati dal sistema, delle interfacce tra i componenti del sistema e degli algoritmi usati.

Le attività principali (per la progettazione di un sistema informativo) sono:

- **Progettazione dell'architettura**, identifica la struttura complessiva del sistema, dei componenti e delle loro relazioni e distribuzione
- **Progettazione del database**, progetta le strutture dati del sistema e come devono essere rappresentate in un database
- **Progettazione dell'interfaccia**, definisce l'interfaccia dei componenti del sistema in modo che un componente possa essere usato da altri componenti senza conoscerne la sua implementazione
- **Progettazione e scelta dei componenti**, si ricercano i componenti riutilizzabili o vengono progettati i nuovi componenti, il modello ottenuto può essere utilizzato per generare automaticamente una implementazione

Sviluppo del software

Lo sviluppo di un programma per implementare un sistema segue naturalmente la progettazione del sistema.

La progettazione di un sistema di sicurezza critica, di solito è dettagliata in modo approfondito prima di iniziare l'implementazione.

È più comune che la progettazione e lo sviluppo del programma siano intrecciati.

Convalida del software

L'attività di **verifica e la convalida** è intesa a mostrare che un sistema è **conforme** alle sue specifiche e **soddisfa** le aspettative del cliente.

Il test dei programmi, dove il sistema viene eseguito utilizzando dati di prova simulati, è la tecnica principale di convalida del software.

La convalida può richiedere anche attività di controllo, ispezione, revisione ad ogni stadio del processo di sviluppo software (dalla definizione dei requisiti dell'utente allo sviluppo del programma).

Convalida del software

Le attività principali sono:

- **Test dei componenti (o delle unità)**, i componenti vengono testati dalle persone che sviluppano il sistema, ciascun componente separatamente e si usano strumenti di automazione come, ad esempio, *JUnit*
- **Test del sistema**, si occupa di trovare gli errori causati da interazioni impreviste tra componenti e i problemi con le relative interfacce, verifica le proprietà significative del sistema
- **Test del cliente**, il sistema viene testato dal cliente con i suoi dati, è funzionale all'approvazione per la messa in uso

Evoluzione del software

- o **manutenzione del software**, il software può essere modificato in qualsiasi momento, durante e dopo lo sviluppo del sistema, anche grandi modifiche sono sempre più economiche delle corrispondenti modifiche dell'hardware.

La distinzione (storica) tra sviluppo e manutenzione del software sta diventando sempre più **irrilevante**. Sono pochi i sistemi completamente nuovi ed è più sensato vedere lo sviluppo e la manutenzione come un tutt'uno.

L'ingegneria del software è un **unico processo evolutivo**, in cui il software viene modificato continuamente nel corso della sua vita per adeguarlo ai cambiamenti dei requisiti e delle necessità dei clienti.

Far fronte ai cambiamenti

Sono inevitabili nella maggior parte dei progetti, indipendentemente dal modello di processo utilizzato, è essenziale che esso consenta di adeguare il software che si sta sviluppando ai cambiamenti in corso.

Per ridurre i costi di **rilavorazione** del progetto, si usano due approcci correlati:

- **Anticipazione dei cambiamenti**, quando il processo software include attività che possono anticipare o predire possibili variazioni, prima che sia richiesta una significativa rilavorazione
- **Tolleranza ai cambiamenti**, quando il processo e il software sono progettati in modo che le modifiche possano essere facilmente apportate al sistema (di solito implica qualche forma di sviluppo incrementale)

Far fronte ai cambiamenti

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

- **Prototipazione del sistema**, il sistema (o una parte) viene sviluppato rapidamente per verificare i requisiti del cliente e la fattibilità delle scelte di progettazione. Consente agli utenti di provare il sistema prima della consegna e di perfezionare i loro requisiti
- **Consegna incrementale**, vengono consegnati al cliente gli incrementi del sistema per essere commentati e provati. Evita l'approvazione prematura dei requisiti per l'intero sistema e consente alle modifiche di essere incluse in successivi incrementi a costi relativamente più bassi

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

Processo software

Differenti tipi di sistemi richiedono differenti approcci di sviluppo: il software real-time deve essere specificato in maniera completa prima che inizi lo sviluppo, mentre nei sistemi di commercio elettronico le specifiche e il programma sono solitamente sviluppati assieme.

Esempi di processi software sono

il processo 'a cascata', Unified Process (UP), Scrum, il modello di sviluppo a spirale, lo sviluppo rapido di applicazione (RAD), Extreme Programming (XP).

Le quattro attività fondamentali (specific, sviluppo, convalida ed evoluzione) sono organizzate i modo diverso in processi di sviluppo diversi: in sequenza (nel modello a cascata) o intrecciate (nel modello di sviluppo incrementale)

- Il **modello di processo software** o **paradigma di processo** è una rappresentazione semplificata di un processo software: sono strutture di processo da estendere e adattare per creare processi software più specifici
- **Modello a cascata**: le attività di processo fondamentali, specifica, sviluppo e convalida sono rappresentate come fasi distinte del processo
- **Sviluppo incrementale**: intreccia le attività di specifica, sviluppo e convalida. Il sistema viene sviluppato come una serie di versioni (incrementi), ciascuna delle quali aggiunge nuove funzionalità alla versione precedente
- **Integrazione e configurazione**: si basa sulla disponibilità di un gran numero di componenti o sistemi riutilizzabili. Il processo di sviluppo si basa sulla configurazione di questi componenti per utilizzarli in una nuova disposizione e integrarli in un sistema

- Non esiste un modello di processo universale che si possa applicare appropriatamente a tutti i tipi di sviluppo del software
- Il processo appropriato **dipende dai requisiti dei clienti** e delle politiche normative, dall'ambiente in cui il software sarà utilizzato e dal tipo di software che si intende sviluppare, ad esempio:
 - Il software a sicurezza critica di solito è sviluppato utilizzando un processo a cascata in quanto sono richieste molte analisi e documentazione prima di iniziare l'implementazione
 - I prodotti software adesso sono sempre sviluppati utilizzando un modello di processo incrementale
 - I sistemi aziendali vengono sempre più sviluppati configurando e integrando i sistemi esistenti per creare un nuovo sistema con le funzionalità richieste

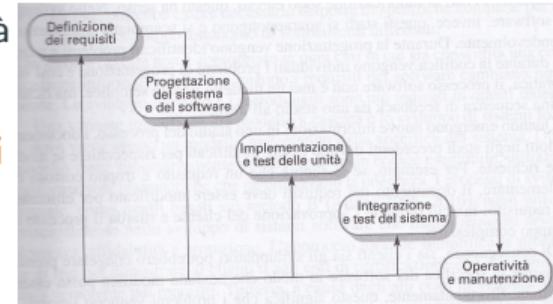
Modelli di processi software

- Gran parte dello sviluppo pratico del software si basa su un modello generale, ma spesso incorpora caratteristiche di altri modelli
- Ingegnerizzazione di grandi sistemi: ha senso combinare alcune delle migliori caratteristiche di tutti i processi generali: è necessario avere informazioni sui requisiti essenziali dei sistemi per progettare un'architettura software a supporto di tali requisiti, non è possibile procedere solo in modo incrementale
- I sottosistemi di un sistema più grande possono essere sviluppati utilizzando approcci differenti: le parti del sistema che sono ben chiare possono essere specificate e sviluppate utilizzando il modello a cascata, altre parti del sistema, che sono difficili da specificare preventivamente, dovrebbero essere sempre sviluppate utilizzando un approccio incrementale

Modello a cascata (o sequenziale)

È basato su uno svolgimento sequenziale delle diverse attività dello sviluppo del software:

- all'inizio del progetto vengono definiti in dettaglio tutti i requisiti

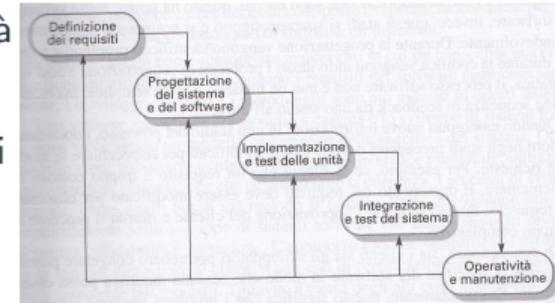


© I. Sommerville. Ingegneria del software. Pearson, 2017.

Modello a cascata (o sequenziale)

È basato su uno svolgimento sequenziale delle diverse attività dello sviluppo del software:

- all'inizio del progetto vengono definiti in dettaglio tutti i requisiti
- all'inizio del progetto viene definito un piano temporale dettagliato delle attività da svolgere

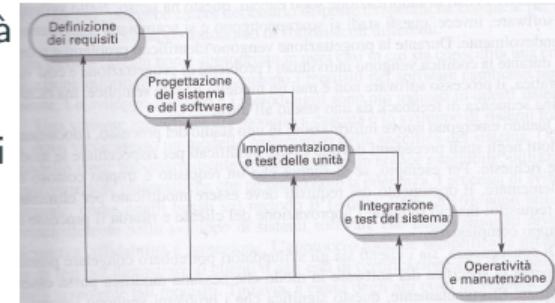


© I. Sommerville. Ingegneria del software. Pearson, 2017.

Modello a cascata (o sequenziale)

È basato su uno svolgimento sequenziale delle diverse attività dello sviluppo del software:

- all'inizio del progetto vengono definiti in dettaglio tutti i requisiti
- all'inizio del progetto viene definito un piano temporale dettagliato delle attività da svolgere
- si prosegue con la modellazione (analisi e progettazione)

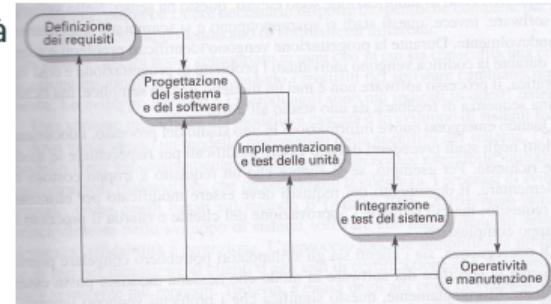


©I. Sommerville. Ingegneria del software. Pearson, 2017.

Modello a cascata (o sequenziale)

È basato su uno svolgimento sequenziale delle diverse attività dello sviluppo del software:

- all'inizio del progetto vengono definiti in dettaglio tutti i requisiti
- all'inizio del progetto viene definito un piano temporale dettagliato delle attività da svolgere
- si prosegue con la modellazione (analisi e progettazione)
- quindi viene creato un progetto completo del software

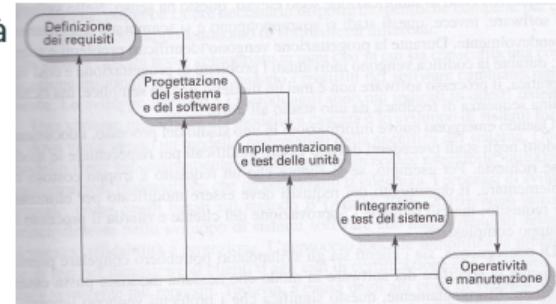


© I. Sommerville. Ingegneria del software. Pearson, 2017.

Modello a cascata (o sequenziale)

È basato su uno svolgimento sequenziale delle diverse attività dello sviluppo del software:

- all'inizio del progetto vengono definiti in dettaglio tutti i requisiti
- all'inizio del progetto viene definito un piano temporale dettagliato delle attività da svolgere
- 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

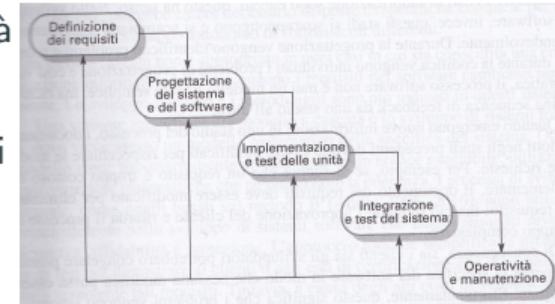


© I. Sommerville. Ingegneria del software. Pearson, 2017.

Modello a cascata (o sequenziale)

È basato su uno svolgimento sequenziale delle diverse attività dello sviluppo del software:

- all'inizio del progetto vengono definiti in dettaglio tutti i requisiti
- all'inizio del progetto viene definito un piano temporale dettagliato delle attività da svolgere
- 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
- seguono verifica e rilascio (e successiva manutenzione) del prodotto realizzato



© I. Sommerville. Ingegneria del software. Pearson, 2017.

Modello a cascata (o sequenziale)

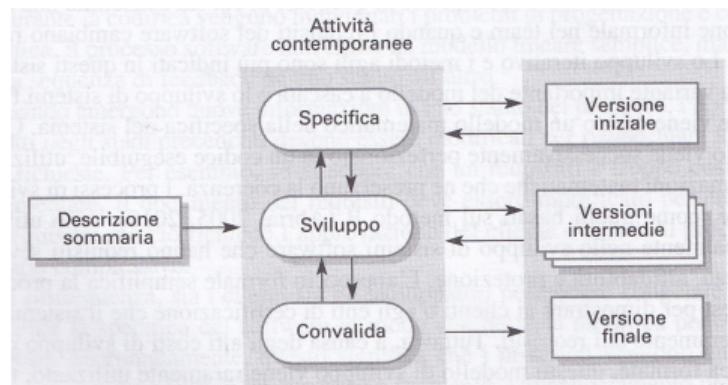
- Il modello di processo di sviluppo a cascata fu derivato dai modelli utilizzati nell'ingegnerizzazione di grandi sistemi militari
- Il modello a cascata era piuttosto comune fino a pochi anni fa
- Il modello a cascata è un esempio di processo guidato da un piano, almeno in principio, occorre pianificare tutte le attività di processo prima di iniziare lo sviluppo del software, è appropriato:
 - nei sistemi integrati, dove il software deve interfacciarsi con i sistemi hardware
 - i sistemi critici, dove occorre un'analisi approfondita della sicurezza e della protezione del software
 - grandi sistemi software che fanno parte di sistemi più complessi sviluppati da più società
- Il modello a cascata non è appropriato in quei casi in cui sia consentita una comunicazione informale nei team e quando i requisiti del software cambiano rapidamente
- Una variante è lo sviluppo di sistemi formali, dove viene creato un modello matematico della specifica del sistema successivamente perfezionato in un codice eseguibile (sistemi software con severi requisiti di sicurezza, affidabilità, protezione), è molto costoso

Sviluppo incrementale

Lo sviluppo incrementale si basa sull'idea di sviluppare un'implementazione iniziale, esporla agli utenti e perfezionarla attraverso molte versioni, finché non si ottiene il sistema richiesto

Le attività di

specificazione, sviluppo e convalida sono intrecciate anziché separate, con feedback veloci tra le varie attività.



©I. Sommerville. Ingegneria del software. Pearson, 2017.

Sviluppo incrementale

- Lo sviluppo incrementale in qualche forma è adesso l'approccio più comune per sviluppare sistemi di applicazioni e prodotti software
- Questo approccio può essere *plan-driven*, agile o una combinazione di questi approcci
- In un approccio *plan-driven*, gli incrementi del sistema sono stabiliti in anticipo
- In un approccio agile, vengono identificati gli incrementi iniziali, ma lo sviluppo dei successivi incrementi dipende dall'avanzamento del lavoro e dalle priorità del cliente

Sviluppo incrementale

- Lo sviluppo incrementale è migliore dell'approccio a cascata per quei sistemi i cui requisiti è probabile che cambino durante il processo di sviluppo: è questo il caso di molti sistemi aziendali e prodotti software
- Riflette il modo in cui risolviamo i problemi: raramente troviamo in anticipo la soluzione finale di un problema, ma arriviamo alla soluzione tramite una serie di passaggi, tornando indietro quando ci accorgiamo di aver commesso un errore
- È più economico e semplice apportare modifiche al software mentre viene sviluppato
- Ciascun incremento o versione del sistema incorpora qualcosa delle funzionalità richieste dal cliente
- Gli incrementi iniziali includono la funzionalità più importante e urgente richiesta dal cliente, così che il cliente possa valutare il sistema nelle prime fasi di sviluppo e, se necessario, modificando solamente l'incremento corrente

Sviluppo incrementale

- Il costo di implementazione delle modifiche dei requisiti è ridotto
- È più facile ottenere feedback del cliente sul lavoro di sviluppo che è stato fatto
- È possibile consegnare in anticipo al cliente una versione utilizzabile del software, anche se non sono state incluse tutte le funzionalità

però:

- Il processo non è visibile, i manager devono avere delle consegne regolari per misurare i progressi
- La struttura dei sistemi tende a degradarsi quando vengono aggiunti nuovi incrementi. Per ridurre il livello di degrado strutturale e di complicazione del codice, i metodi agili suggeriscono un costante *refactoring* (miglioramento della struttura) del software
- Nei sistemi grandi, complessi dove diversi team sviluppano parti differenti le responsabilità devono essere definite in modo chiaro richiede pianificazione in anticipo

Sviluppo incrementale VS ‘a cascata’

Sviluppo incrementale VS ‘a cascata’

- Passi di sviluppo brevi e feedback per chiarire i requisiti VS (grande) lavoro speculativo iniziale
- In base a dati statistici¹, i metodi iterativi sono associati a percentuali di successo e di produttività più elevate, nonché a livelli minori di difetti

¹Larman, C. 2003. *Agile and Iterative Development: A manager's Guide Reading*, MA., Addison-Wesley e in Larman, C. Basili, V. *Iterative and Incremental Development: A History*, IEEE Computer, June 2003

Sviluppo incrementale VS ‘a cascata’

In Larman, C. 2003. *Agile and Iterative Development: A manager's Guide Reading, MA.*, Addison-Wesley e in Larman, C. Basili, V. *Iterative and Incremental Development: A History, IEEE Computer, June 2003* gli autori mostrano come:

- il metodo a cascata sia una pratica **mediocre** per la maggior parte dei progetti software, anziché un approccio valido
- l'approccio a cascata sia caratterizzato da una **minore produttività** e da **maggiori percentuali di difetti**
- le stime iniziali a cascata dei tempi e dei costi **variano notevolmente** dai valori finali

Sviluppo incrementale VS 'a cascata'

In Larman, C. 2003. *Agile and Iterative Development: A manager's Guide* Reading, MA., Addison-Wesley e in Larman, C. Basili, V. *Iterative and Incremental Development: A History*, IEEE Computer, June 2003 gli autori mostrano come:

- il metodo a cascata sia una pratica **mediocre** per la maggior parte dei progetti software, anziché un approccio valido
- l'approccio a cascata sia caratterizzato da una **minore produttività** e da **maggiori percentuali di difetti**
- le stime iniziali a cascata dei tempi e dei costi **variano notevolmente** dai valori finali

Perché? Perché partono dal presupposto che

le specifiche sono prevedibili e stabili e possono essere definite correttamente sin dall'inizio, a fronte di un basso tasso di cambiamenti.

Nella maggior parte dei progetti software si è soliti riutilizzare il Software (anche modificandolo e integrandolo nel nuovo sistema).

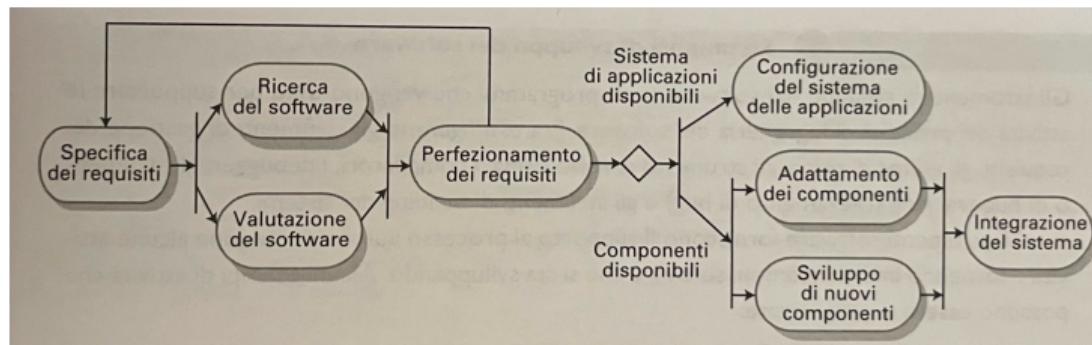
Dal 2000, i processi di sviluppo del software che sfruttano il riutilizzo di software esistente si stanno diffondendo sempre più:

- Sistemi di applicazione indipendenti che sono configurati per essere utilizzati in un particolare ambiente (sistemi generici con numerose funzionalità, che devono essere adattate a una specifica applicazione)
- Collezioni di oggetti che sono sviluppate come un componente o un pacchetto da integrare tramite un framework di integrazione dei componenti come JavaSpring
- Servizi web che sono sviluppati in conformità agli standard e che sono disponibili per essere utilizzati da siti remoti su Internet

Integrazione e configurazione

Le fasi principali sono:

- Specifica dei requisiti (iniziali del sistema)
- Ricerca e valutazione del software che possono 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 non disponibile un sistema pronto all'uso



Integrazione e configurazione

- Ha il vantaggio di **ridurre** la quantità di software da sviluppare, riducendo così i costi e i rischi e portando a consegne più veloci
- Sono però inevitabili **compromessi** nei requisiti
- Si **perde** il controllo sull'evoluzione del sistema, in quanto nuove versioni dei componenti riutilizzati non sono sotto il controllo dell'organizzazione che li usa

Sviluppo iterativo ed evolutivo

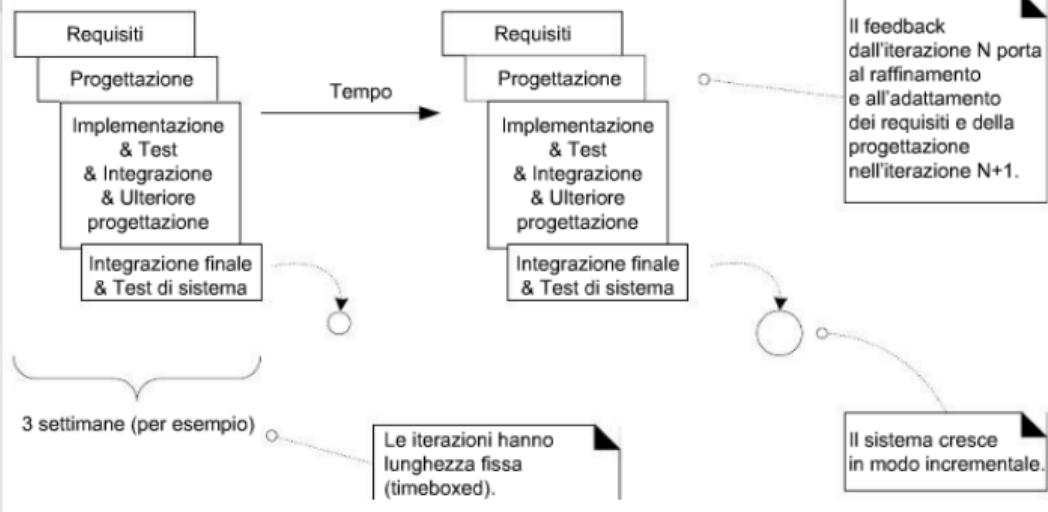
Lo sviluppo incrementale, iterativo ed evolutivo

- comporta fin dall'inizio la programmazione e il test di un sistema software
- comporta che lo sviluppo inizi prima che tutti i requisiti siano stati definiti in modo dettagliato
- viene usato il feedback per chiarire e migliorare le specifiche in evoluzione del sistema

Sviluppo iterativo ed evolutivo

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

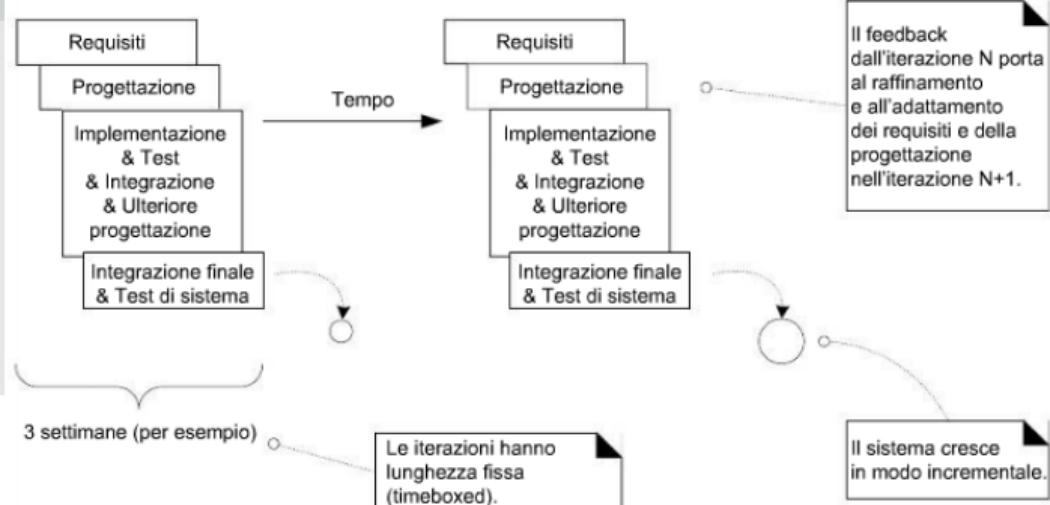


©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Sviluppo iterativo ed evolutivo

Sviluppo iterativo e

- **incrementale:** il sistema cresce in modo incrementale nel tempo, iterazione dopo iterazione
- **evolutivo:** il feedback e l'adattamento fanno evolvere le specifiche e il progetto



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

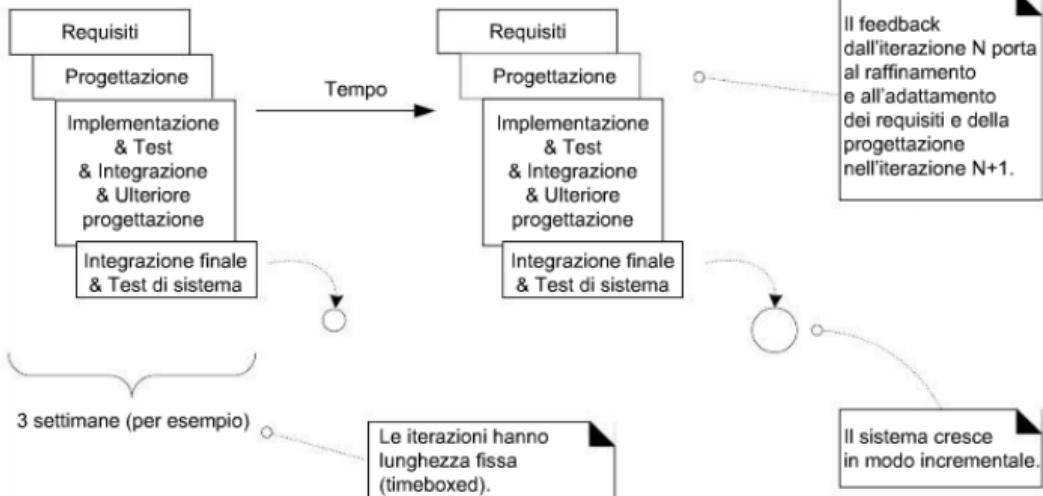
Sviluppo iterativo ed evolutivo

Nel processo iterativo

Non vi è fretta di iniziare la codifica, né una fase di progettazione lunga che provi a perfezionare tutti i dettagli della progettazione prima della programmazione.

Il risultato di ciascuna iterazione è un **sistema eseguibile** ma **incompleto** (lo sarà dopo 10/15 iterazioni).

Il risultato di un'iterazione **non è un prototipo** ma un **sottoinsieme del sistema finale**.



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

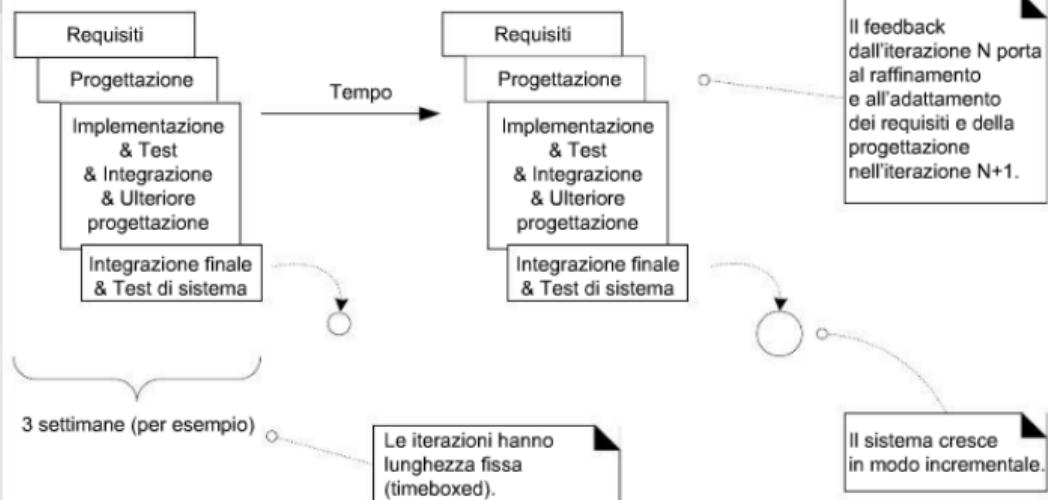
Sviluppo iterativo ed evolutivo

Nel processo iterativo

Ciascuna iterazione comporta la scelta di un piccolo sottoinsieme di requisiti, una rapida progettazione implementazione e test.

Questo permette **feedback rapidi** da parte degli utenti, degli sviluppatori e dei test.

Un'opportunità per **modificare o adattare** la comprensione dei requisiti e il progetto.

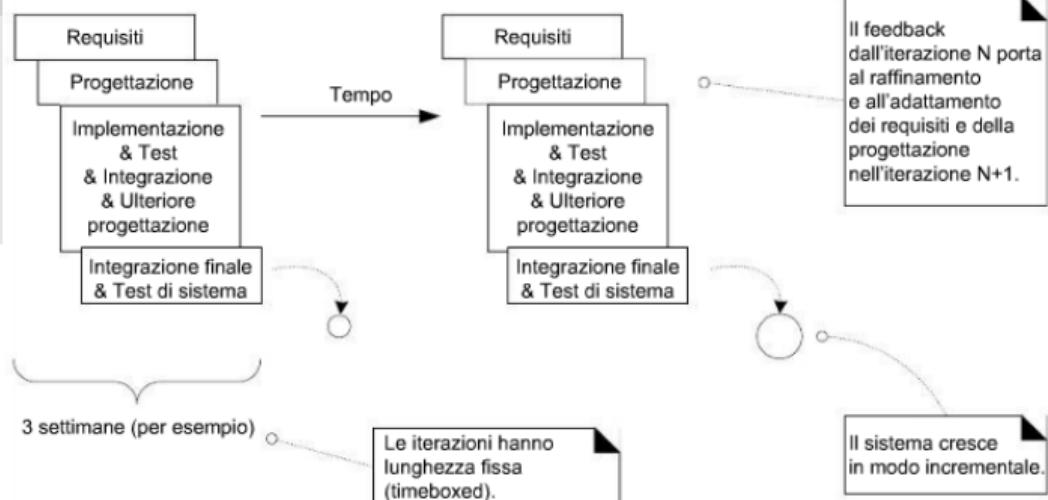


©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Sviluppo iterativo ed evolutivo

Nel processo iterativo

Attraverso il feedback iterativo e l'adattamento, il sistema **evolve** e **converge** verso i requisiti corretti e il progetto più appropriato.



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

Sviluppo iterativo ed evolutivo

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

Vantaggi:

- Riduzione precoce dei rischi maggiori (tecnici, requisiti, obiettivi, usabilità, ...)
- Progresso visibile fin dall'inizio
- Feedback (dai modellatori, sviluppatori, programmati, cliente, mercato) precoce, coinvolgimento dell'utente e adattamento
- Gestione della complessità (evita la “*paralisi da analisi*”)

Fondamentale è un'iterazione da due a sei settimane, passi piccoli, feedback rapido e adattamento.

Un'iterazione di lunghezza fissata è detta **timeboxed**.

Sviluppo incrementale, iterativo ed evolutivo: storia

- Alla fine degli anni cinquanta, al progetto spaziale Mercury fu applicato lo sviluppo incrementale, iterativo ed evolutivo piuttosto che il metodo a cascata (IDD, Iterative and Incremental Development)



Sviluppo incrementale, iterativo ed evolutivo: storia

- Alla fine degli anni cinquanta, al progetto spaziale Mercury fu applicato lo sviluppo incrementale, iterativo ed evolutivo piuttosto che il metodo a cascata (IDD, Iterative and Incremental Development)
- Il programma Mercury fu il primo programma statunitense a prevedere missioni spaziali con equipaggio



Sviluppo incrementale, iterativo ed evolutivo: storia

- Alla fine degli anni cinquanta, al progetto spaziale Mercury fu applicato lo sviluppo incrementale, iterativo ed evolutivo piuttosto che il metodo a cascata (IDD, Iterative and Incremental Development)
- Il programma Mercury fu il primo programma statunitense a prevedere missioni spaziali con equipaggio
- Trident nuclear programme (missili balistici su sottomarini nucleari)



Sviluppo incrementale, iterativo ed evolutivo: storia

- Alla fine degli anni cinquanta, al progetto spaziale Mercury fu applicato lo sviluppo incrementale, iterativo ed evolutivo piuttosto che il metodo a cascata (IDD, Iterative and Incremental Development)
- Il programma Mercury fu il primo programma statunitense a prevedere missioni spaziali con equipaggio
- Trident nuclear programme (missili balistici su sottomarini nucleari)
- Anni '70: il software di controllo di volo dello Space Shuttle fu costruito con 17 iterazioni di circa quattro settimane l'una



Sviluppo incrementale, iterativo ed evolutivo: storia

- Alla fine degli anni cinquanta, al progetto spaziale Mercury fu applicato lo sviluppo incrementale, iterativo ed evolutivo piuttosto che il metodo a cascata (IDD, Iterative and Incremental Development)
- Il programma Mercury fu il primo programma statunitense a prevedere missioni spaziali con equipaggio
- Trident nuclear programme (missili balistici su sottomarini nucleari)
- Anni '70: il software di controllo di volo dello Space Shuttle fu costruito con 17 iterazioni di circa quattro settimane l'una
- IBM (1968): primo documento che promuoveva lo sviluppo iterativo



Sviluppo incrementale, iterativo ed evolutivo: un esempio

Un esempio di progetto iterativo (UP) composto da 20 iterazioni prima del rilascio al cliente (dal libro di testo, sezione 2.5).

Passo 1

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.

Sviluppo incrementale, iterativo ed evolutivo: un esempio

Un esempio di progetto iterativo (UP) composto da 20 iterazioni prima del rilascio al cliente (dal libro di testo, sezione 2.5).

Passi 2 e 3

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.

Sviluppo incrementale, iterativo ed evolutivo: un esempio

Un esempio di progetto iterativo (UP) composto da 20 iterazioni prima del rilascio al cliente (dal libro di testo, sezione 2.5).

Passo 3

- 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.

Sviluppo incrementale, iterativo ed evolutivo: un esempio

Un esempio di progetto iterativo (UP) composto da 20 iterazioni prima del rilascio al cliente (dal libro di testo, sezione 2.5).

Passi 4, 5, 6 e 7

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.

Sviluppo incrementale, iterativo ed evolutivo: un esempio

Un esempio di progetto iterativo (UP) composto da 20 iterazioni prima del rilascio al cliente (dal libro di testo, sezione 2.5).

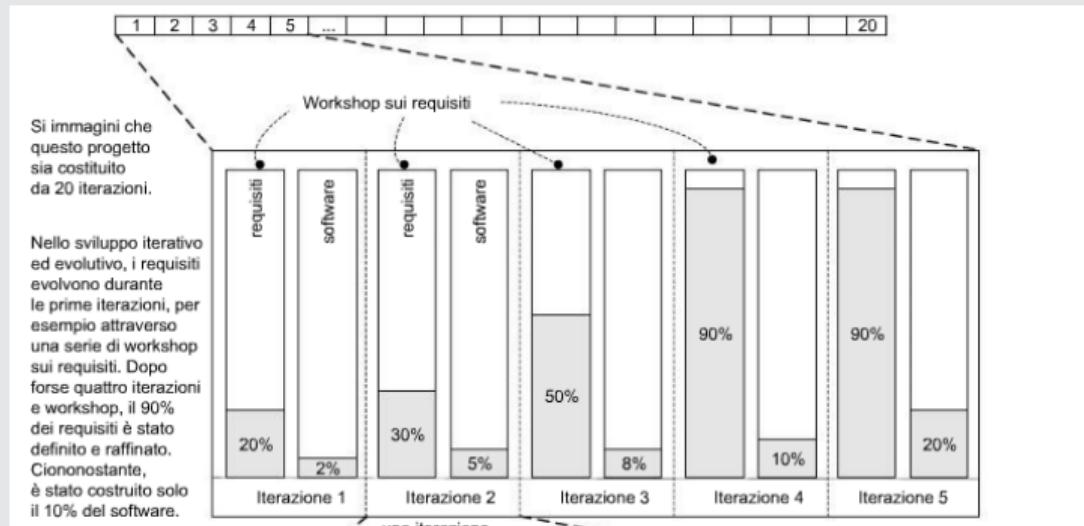
Passi 8 e 9

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-

Sviluppo incrementale, iterativo ed evolutivo: un esempio

Un esempio di progetto iterativo (UP) composto da 20 iterazioni prima del rilascio al cliente (dal libro di testo, sezione 2.5).

Overview



Pianificazione guidata dal rischio e dal cliente

Nei processi iterativi, in ogni iterazione viene stabilito il piano di lavoro per una sola iterazione, **pianificazione iterativa o adattativa**:

- UP: alla fine di ciascuna iterazione per l'iterazione successiva
- Scrum: all'inizio di ciascuna iterazione, per stabilire il piano dell'iterazione corrente

Gli obiettivi dell'iterazione non vengono cambiati: accettabile perché le iterazioni sono brevi e il feedback frequente. L'interesse è che il team di sviluppatori lavori al suo meglio, concentrandosi sul lavoro stabilito per l'iterazione.

Pianificazione guidata dal rischio e dal cliente

La pianificazione è **guidata dal rischio e guidata dal cliente**:

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

Sviluppo agile del software

Metodi agili: contesto

- Le aziende oggi lavorano in un ambiente globale dal cambiamento rapido; devono cogliere nuove opportunità, e rispondere a nuovi mercati alle condizioni economiche variabili e alla presenza di prodotti e servizi concorrenti
- Il software è parte essenziale delle operazioni aziendali, deve trarre vantaggio dalle nuove opportunità
- La rapidità dello sviluppo e della consegna è quindi 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 per i clienti è impossibile prevedere come un sistema influenzerà le pratiche operative, come interagirà con gli altri sistemi e quali operazioni degli utenti dovranno essere automatizzate
- I requisiti reali diventano chiari solo dopo che il sistema è stato consegnato e utilizzato dagli utenti, fattori esterni possono indurre a modificare ulteriormente i requisiti

Metodi agili: contesto

- Anni '80 e '90: attenta pianificazione dei progetti, la garanzia di qualità formale, l'uso di metodi di analisi e progettazione supportati da strumenti software e processi di sviluppo software controllati e rigorosi (da progetti di grandi dimensioni, sistemi aerospaziali e governativi)
- Nell'applicare tali principi ai sistemi di aziende di piccole e medie dimensioni, gli overhead richiesti erano troppo alti
- Anni '90: metodi agili, concentrarsi sul software stesso anziché sulla progettazione e sulla documentazione
- I metodi agili sono particolarmente indicati per sviluppare applicazioni nelle quali i requisiti del sistema cambiano rapidamente durante il processo di sviluppo.
- Ideata per consentire una consegna rapida ai clienti che possono quindi proporre requisiti nuovi o modificati da includere in successive iterazioni del sistema

I metodi per lo **sviluppo agile** di solito applicano lo sviluppo iterativo ed evolutivo.

L'enfasi è data su una risposta rapida e flessibile ai cambiamenti (*agilità*): iterazioni brevi, raffinamento evolutivo dei piani, dei requisiti e del progetto.

Agile Modeling

Lo scopo della modellazione (abbozzare UML, ...) è principalmente quello di **comprendere**, di agevolare la **comunicazione**, non di documentare.

Ossia, esplorare rapidamente (più rapidamente che con il codice) le alternative e il percorso verso un buon progetto (object oriented).



I metodi agili suggeriscono che il software deve essere sviluppato e consegnato in modo incrementale.

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

- Lo sviluppo di prodotti di piccole e medi dimensioni
- Lo sviluppo personalizzato di sistemi all'interno di un'organizzazione dove c'è un chiaro impegno da parte del cliente di essere coinvolto nel processo di sviluppo

- Adottare un metodo agile non significa evitare del tutto la modellazione, bensì lo scopo della modellazione è quello di agevolare la comprensione e la comunicazione, non di documentare
- Non si deve modellare o applicare UML per eseguire per intero o per la maggior parte la progettazione del software. È norma rimandare i problemi di progettazione semplici o diretti alla fase di programmazione e risolverli nel corso della programmazione e dei test
- Va utilizzato lo strumento più semplice possibile, che aiutino la creatività con il minimo dispendio di energie: es. abbozzo di UML alla lavagna

- La modellazione non va fatta da soli, ma piuttosto a coppie (o in tre), con lo scopo di scoprire, capire e condividere ciò che si è comprensione
- Solo il codice verificato dimostra il vero progetto, tutti i diagrammi precedenti sono suggerimenti incompleti, ed è meglio considerarli come semplici 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 programmatore significa seguire pratiche a cascata, opposte alla filosofia agile



È uno dei metodi agili più noti.

- Lo sviluppo incrementale è supportato attraverso piccole e frequenti release del sistema. I requisiti si basano su semplici scenari, che sono utilizzati come base per decidere quale funzionalità deve essere inclusa in un incremento del sistema
- Il coinvolgimento dell'utente è supportato attraverso l'impegno costante del cliente nel team di sviluppo
- Le persone sono supportate dalla programmazione in coppia, dal processo collettivo del codice del sistema e da un processo di sviluppo sostenibile che non richiede periodi di lavoro eccessivamente lunghi

eXtreme Programming (continua)

È uno dei metodi agili più noti.

- Le modifiche sono supportate da regolari release del sistema, dallo sviluppo preceduto da test, dal refactoring per evitare la degenerazione del codice, e dall'integrazione continua di nuove funzionalità
- Il mantenimento della semplicità è supportato dal costante refactoring che miglioramenti la qualità del codice e dall'uso di semplici progetti che non necessariamente prevedono modifiche del sistema

Medodi agili: pratiche innovative

Tra le pratiche innovative dei metodi agili:

- **Storie utente**, sono scenari d'uso in cui potrebbe trovarsi Un utente del sistema. Il cliente del sistema lavora a stretto contatto con il team di sviluppo e discute questi scenari con altri membri del team
- **Refactoring**, non ha senso progettare per il cambiamento, le modifiche previste non si avverano mai, le modifiche dovranno essere sempre apportate al codice che si sta sviluppando, il codice è costantemente rifattorizzato, lo sviluppo incrementale deteriora la struttura del software
- **Sviluppo con test iniziali**, lo sviluppo non può procedere finché tutti i test non sono stati superati, sviluppo guidato dai test
- **Programmazione a coppie**, programmatore siedono realmente alla stessa postazione di lavoro per sviluppare il software, supporta l'idea della proprietà e della responsabilità comune del sistema, revisione informale, incentiva il refactoring

Scrum: gestione agile della progettazione

Lo sviluppo agile deve essere gestito in modo che venga fatto il miglior uso del tempo e delle risorse a disposizione del team. Il metodo agile **Scrum** cerca di risolvere questo problema.

Scrum

offre un framework per organizzare progetti agili e fornire una visibilità esterna su ciò che sta accadendo, ossia si occupa dell'organizzazione del lavoro e della gestione dei progetti.

Scrum è un approccio iterativo e incrementale, ciascuna iterazione ha una durata fissata denominata **Sprint** e non vengono mai estese.

Scrum: gestione agile della progettazione

Sono presenti tre ruoli:

- **Product Owner**, definisce le caratteristiche del prodotto software da realizzare e specifica le priorità tra queste caratteristiche attraverso il **Product Backlog** (en elenco di voci, funzionalità e requisiti)
- **Development Team**, possiedono le competenze necessarie a sviluppare il software
- **Scrum Master**, aiuta l'intero gruppo ad apprendere e applicare Scrum, al fine di ottenere il valore desiderato, **non** è il manager del Development Team piuttosto un istruttore e una guida

Scrum: gestione agile della progettazione

All'inizio del progetto, il Product Backlog descrive tutte le caratteristiche del prodotto, di iterazione in iterazione questo elaborato viene aggiornato e descrive le cose che devono essere ancora fatte.

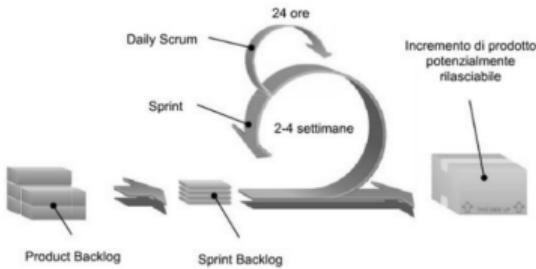
Il Development Team seleziona dal product Backlog un insieme di voci da sviluppare durante quell'iterazione (**Sprint Goal**), compila lo **Sprint Backlog**, i compiti dettagliati per raggiungere il goal.

Il risultato di ciascuno Sprint

deve essere un prodotto software funzionante, chiamato “incremento di prodotto potenzialmente rilasciabile”. Deve essere integrato, verificato, documentato per l'utente finale.

Scrum: gestione agile della progettazione

Nella **Sprint Review**, il Product Owner e il Development Team presentano alle diverse parti interessate l'incremento di prodotto, ne fanno la dimostrazione, ottengono un feedback e decidono cosa è utile fare nel successivo Sprint.



©C. Larman. Applicare UML e i Pattern. Pearson, 2016.

La caratteristica distintiva di Scrum

tra i metodi agili è l'enfasi sull'adozione di team auto-organizzati e auto-gestiti. Inoltre, Scrum è basato su un insieme di elaborati ed eventi che hanno lo scopo di rendere visibili gli obiettivi e il progetto delle iterazioni e di favorire un adattamento evolutivo del processo di sviluppo.

Problemi pratici con i metodi agili

- L'informalità dello sviluppo agile è incompatibile con l'approccio legale alla definizione dei contratti che di solito si usano nelle grandi società
- I metodi agili sono più indicati per lo sviluppo di nuovo software, non per la manutenzione del software; eppure la maggior parte dei costi del software nelle grandi società proviene dalla manutenzione dei loro sistemi software esistenti
- I metodi agili sono ideati per piccoli team fisicamente vicini; eppure la maggior parte dello sviluppo del software oggi coinvolge team distribuiti in tutto il mondo

La scalabilità

dei metodi agili richiede che alcune pratiche basate sui piani siano integrate con le pratiche agili (es. più documentazione, più rappresentanti del cliente, strumenti comuni per i team e allineamento delle release).

Sviluppo incrementale, iterativo ed evolutivo (agile)

