**SOFTWARE DEVELOPMENT**

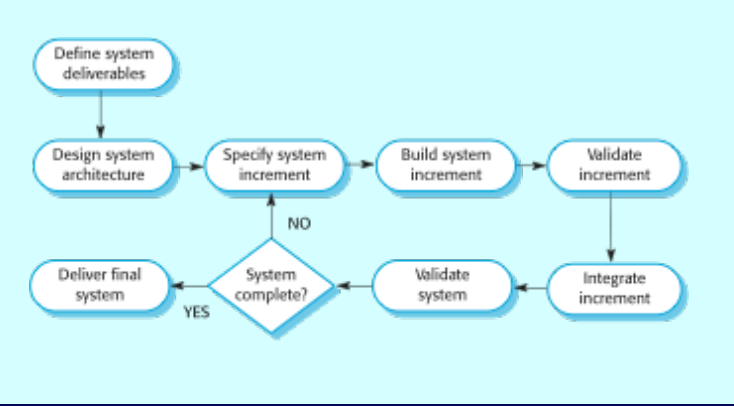
**Rapid application development (RAD):**

* Coinvolge modelli di sviluppo iterativi
* Costruzione di prototipi (che si evolverà in app completa)
* Utilizzo di strumenti CASE (*supportano lo sviluppo del software attraverso interfacce grafiche e librerie di funzionalità*)
* Riutilizzo di sistemi esistenti pronti all’uso (*ingegneria del software basato sul riutilizzo e i componenti*)

**RAPID SOFTWARE DEV**

* Essendo che l’ambiente aziendale è in rapida evoluzione
* Il cliente potrebbe accettare un software leggermente peggiore qualitativamente, purché venga consegnato prima o in tempo, altrimenti potrebbe rischiare che il software possa diventare non più funzionale
* Siccome l’ambiente cambia, è spesso impossibile arrivare ad una soluzione stabile e consistente secondo le richieste del cliente
* Un modello a cascata è impraticabile in questo caso, meglio uno sviluppo iterativo

**PROCESSO DI SVILUPPO ITERATIVO**



**VANTAGGI DI UN SVILUPPO INCREMENTALE**

* **Consegna e sviluppo accelerato**, vengono consegnate prima le funzioni con priorità più alta, appunto **sviluppato in una serie di incrementi**, si valuta ogni incremento e si fanno proposte per il successivo
* Richiedere la **partecipazione del cliente** nello sviluppo del sistema, così da venire meglio incontro alle loro richieste

**PROBLEMI CON LO SVILUPPO INCREMENTALE**

* **PROBLEMI DI GESTIONE**
  + Siamo concentrati su quello che stiamo facendo ora, ma non abbiamo una visione concisa a lungo periodo, sarà più difficile trovare problemi
* **PROBLEMI CONTRATTUALI**
  + Sappiamo cosa fare, ma non siamo capaci di predire le difficoltà e il tempo che ci vorrà
* **PROBLEMI DI CONVALIDA**
  + Senza una specifica, contro cosa viene testato il sistema?
* **PROBLEMI DI MANUTENZIONE**
  + Cambiamenti continui tendono a corrompere la struttura del software, rendendo più costoso l’evoluzione e la manutenzione per soddisfare nuovi requisiti

**SVILUPPO INCREMENTALE VS PROTOTIPO**

* Il **prototipo** viene fatto per scoprire i requisiti necessari al cliente, che magari non ha le idee chiare o non si espone bene, e poi viene buttato
* Invece se **sviluppo in modo incrementale**, quella parte che sto sviluppando, la sto sviluppando perché farà parte del prodotto finale, iniziando con i requisiti che si comprendono meglio

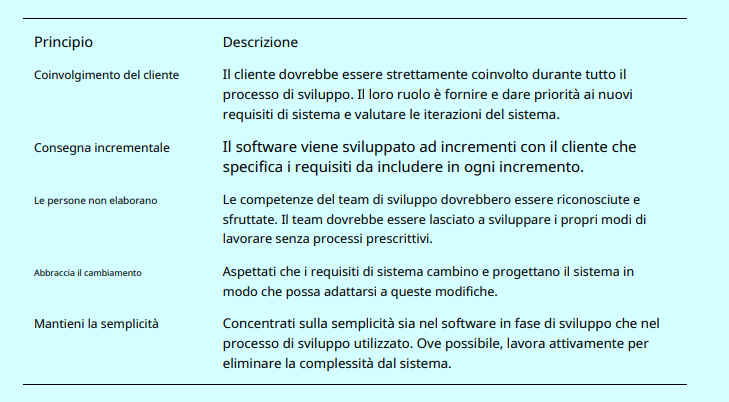
**AMBIENTI RAD**

* I RAD sono progetti per sviluppare applicazioni aziendali con molti dati, e si basano sulla programmazione e presentazione di informazioni di un database
* Qui ci torneranno utili:
  + Linguaggio di programmazione del database
  + Generatore di interfaccia
  + Generatore di rapporti

**METODOLOGIE DI SVILUPPO AGILE**

* L’insoddisfazione per le spese generali ha portato alla creazione di metodi agili:
  + Ci si concentra di più sul **codice** che sul design
  + Ci si basa su un **approccio iterativo** sullo **sviluppo software**
  + Destinati a **fornire rapidamente un software funzionante** e a farlo evolvere rapidamente per soddisfare le mutevoli esigenze
* Questi metodi agili sono **più adatti a sistemi di piccole/medie imprese** o prodotti per PC

**PRINCIPI DEI METODI AGILI**



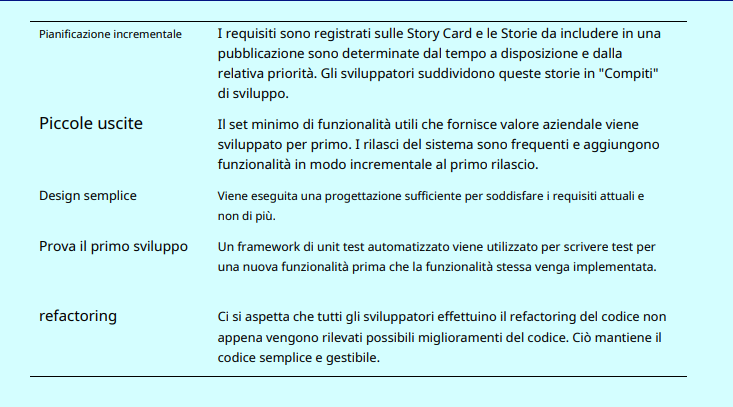
**PROBLEMI CON I METODI AGILI**

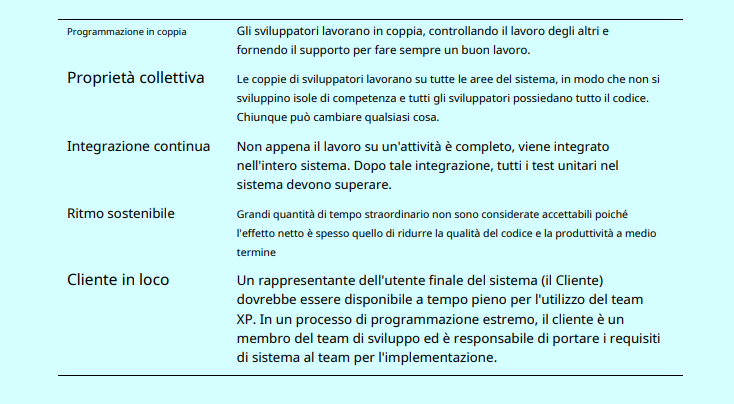
* Mantenere l’interesse del cliente coinvolto nel processo, dovremmo rendere facile per lui il coinvolgimento in quanto i feedback sono essenziali
* Mantenere la semplicità richiede lavoro extra
* I contratti possono essere un problema
* Il team potrebbe non essere adatto all’intenso coinvolgimento dei metodi agili

**PROGRAMMAZIONE ESTREMA**

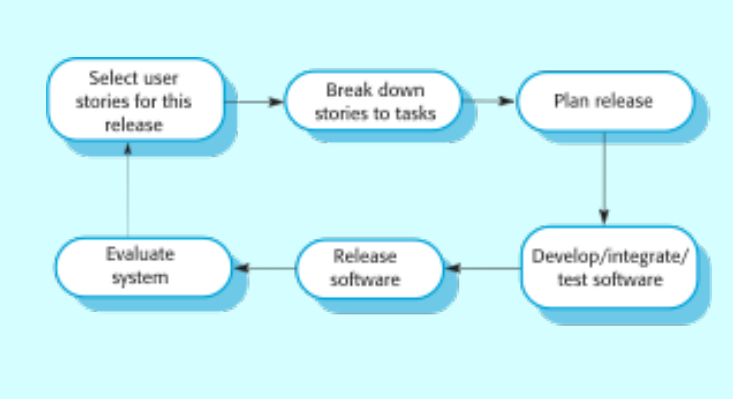
* Metodo agile più conosciuto ed utilizzato
* Adotta un approccio estremo allo sviluppo iterativo
  + Le nuove versioni possono essere create più volte al giorno
  + Incrementi consegnati ogni 2 settimane
  + Tutti i test devono essere eseguiti per ogni build e la build viene accettata solo se i test vengono eseguiti correttamente

**PRATICHE DI PROGRAMMAZIONE ESTREMA**

****

****

**CICLO DI RILASCIO DELLA PROGRAMMAZIONE ESTREMA**

****

**Stories = requisiti**

**SCENARI DEI REQUISITI**

* Nella programmazione estrema, i requisiti degli utenti sono espressi come scenari o storie utente
  + Scritte su carta (in genere post-it)
  + Suddividere in task, queste attività sono la base della pianificazione e delle stime dei costi
* Il cliente sceglie le storie da inserire nella prossima release in base alle loro priorità e alle previsioni del programma

**ESEMPIO DI STORIE SCRITTE SU CARTA (STORY CARD)**

Immagine che contiene testo

Descrizione generata automaticamente

**Test nella Programmazione Estrema**

* Sviluppo dei test incrementali da scenari
* Coinvolgimento dell’utente nello sviluppo e nella convalida dei test
* Test automatizzati utilizzati per eseguire tutti i test dei componenti ogni volta che viene creata una nuova versione

**SVILUPPO TEST-FIRST**

* La scrittura dei test prima del codice chiarisce i requisiti da implementare
* I test sono scritti come programmi
* Tutti i test precedenti e nuovi vengono eseguiti automaticamente quando viene aggiunta nuova funzionalità, così da verificare che non abbia introdotto nuovi errori

**SCHEDE DELLE TASK RICAVATE DALLE STORY CARD**

**Immagine che contiene testo

Descrizione generata automaticamente**

**DESCRIZIONE DEL CASO DI PROVA DA TESTARE**

**Immagine che contiene testo

Descrizione generata automaticamente**

**RIUTILIZZO DEL SOFTWARE**

* I sistemi sono spesso progettati usando componenti esistenti utilizzati in altri sistemi
  + Un riutilizzo sistematico del software può consentire di ottenere un software migliore più rapidamente e a costi inferiori

**SOFTWARE BASATO SUL RIUTILIZZO**

* **Riutilizzo del sistema applicativo**
  + L’intero sistema applicativo può essere riutilizzato incorporandolo senza modifiche in altri sistemi (**COTS** 🡪 ***componenti hard e soft disponibili sul mercato, acquistabili, da poter utilizzare nei progetti***)
* **Riutilizzo dei componenti**
  + Si possono riutilizzare componenti di un’applicazione dai sottosistemi dei singoli progetti
* **Riutilizzo di oggetti e funzioni**
  + Componenti software che implementano un singolo oggetto o funzioni ben definite possono essere riutilizzate

**APPROCCIO DI RIUTILIZZO**

**Immagine che contiene testo

Descrizione generata automaticamente**

**Immagine che contiene testo

Descrizione generata automaticamente**

**PROBLEMI NELL’AFFIDARSI AD UN PROGRAMMA RIUTILIZZATO**

* Mancanza di controllo su funzionalità e prestazioni
* Problemi con l’integrazione del sistema
  + Diversi COTS possono fare ipotesi diverse, rendendo più difficile integrazione
* Nessun controllo sull’evoluzione del sistema
  + I fornitori di COTS, non gli utenti del sistema, controllano l’evoluzione
* Supporto dai fornitori COTS
  + I Fornitori COTS potrebbero non offrire supporto per tutta la durata del prodotto

**SISTEMI ERP**

* Un sistema ERP (Enterpirse resource planning) è un sistema generico che supporta processi aziendali comuni come ordini, fatturazione, produzione ecc…
* Molto utilizzati nelle grandi aziende
* Nucleo generico adattato incorporando conoscenza dei processi e delle regole aziendali

**SOFTWARE EVOLUTION**

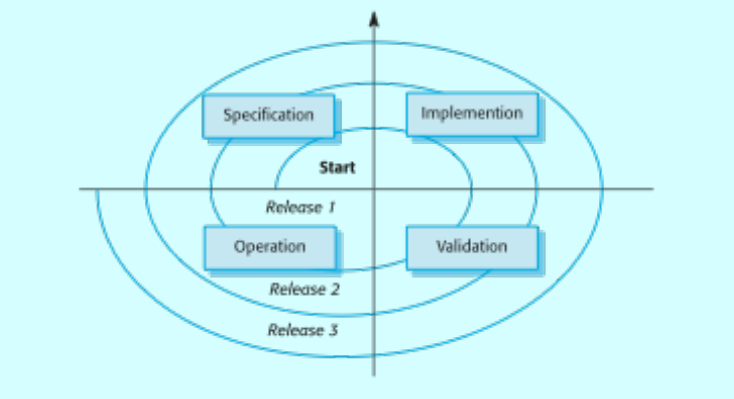
**CAMBIO SOFTWARE**

* Il cambiamento del software è inevitabile
  + Utilizzando il software emergono nuovi requisiti
  + Cambia l’ambiente aziendale
  + Vengono aggiunti al sistema nuovi computer e apparecchiature
  + Potrebbe essere necessario migliorare prestazioni o affidabilità del sistema

**DINAMICHE DI EVOLUZIONE DEL PROGRAMMA**

* E’ LO STUDIO DEI PROCESSI DI CAMBIAMENTO DI SISTEMA
* Lehman e Belady hanno proposti una serie di legislazioni:
  + Applicabile a grandi sistemi sviluppati da grandi organizzazioni
  + Osservazioni sensate più che leggi

L’**evoluzione può essere vista come una spirale**, la quale per ogni release ripete degli step in modo ciclico

****

**Leggi di Lehman**

**Lehman** e **Belady** hanno proposto una serie di **leggi** sulle **dinamiche di evoluzione del programma**

**Immagine che contiene testo

Descrizione generata automaticamente**

**Nota:** non si arriverà mai ad un software senza bug, visto che aggiungiamo e togliamo sempre funzionalità, ma è buono arrivare ad un equilibrio stabile

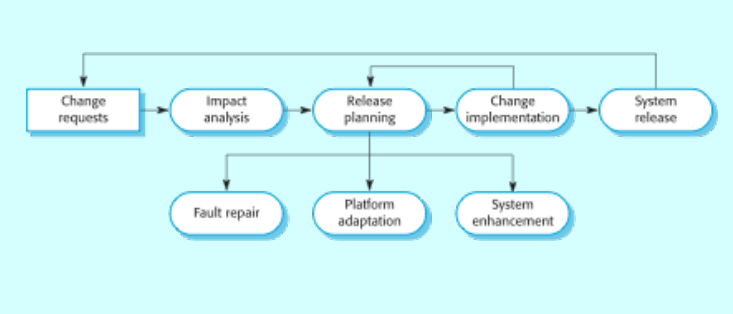
**MANUTENZIONE DEL SOFTWARE**

* Modificare un programma dopo che è stato messo in uso
* Le modifiche vengono implementate modificando i componenti esistenti e aggiungendo nuovi componenti al sistema
* Manutenzione è inevitabile
  + Ambiente cambia 🡪 Requisiti cambiano
  + Devono essere mantenuti se rimangono utili in un ambiente

**FATTORI DI COSTO DI MANUTENZIONE**

* **Stabilità del team**: i Costi di manutenzione si riducono se lo stesso personale è coinvolto nel tempo
* **Competenza personale**: Il personale di manutenzione è spesso inesperto e ha una conoscenza limitata del dominio
* **Età e struttura del programma**: più i programmi invecchiano, più la loro struttura si degrada e diventano più difficili da comprendere e modificare

**PROCESSO DI EVOLUIONE DEL SISTEMA**



**REINGEGNERIZZAZINE DEL SISTEMA**

* **Ristrutturazione o riscrittura parziale o totale** di un sistema legacy senza modificarne le funzionalità
* Applicabile quando alcuni sottosistemi di un sistema più grande richiedono frequenti manutenzioni
* La reingegnerizzazione comporta aggiunta di sforzi per renderli più facili da mantenere, **il sistema può essere ristrutturato e ri-documentato**

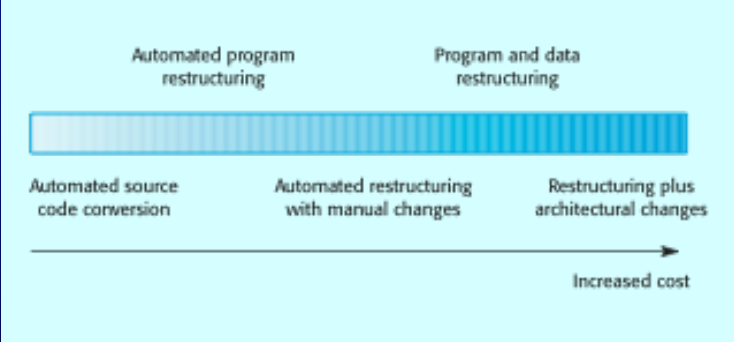
**VANTAGGI DELLA REINGEGNERIZZAZIONE**

* **Rischio ridotto**: in quanto c’è un alto rischio nello sviluppo di un nuovo software (problemi di sviluppo, personale ,specifica)
* **Costo ridotto**: il costo della reingegnerizzazione è molto minore rispetto allo sviluppo di un nuovo software

**REINGEGNERIZZAZIONE ATTIVITA’ DEL PROCESSO**

* **Traduzione del codice sorgente**: converti il codice in una nuova lingua
* **Ingegneria inversa**: analizziamo il programma per caprilo
* **Miglioramento della struttura del programma**
* **Modularizzazione del programma**: riorganizzare la struttura del programma
* **Reingegnerizzazione dei dati**: pulizia e ristrutturazione dei dati di sistema

**SCHEMA DI INCREMENTO DEI COSTI SECONDO APPROCCI DI REING.**

****

**SISTEMA LEGACY =** Un **sistema** o **un’applicazione o un componente obsoleto** che continua ad essere usato a causa del **costo proibitivo di rimpiazzarlo o riprogettarlo**, malgrado la sua scarsa competitività e compatibilità con altri sistemi moderni. I sistemi legacy sono di solito grandi, monolitici e difficili da modificare

**EVOLUZIONE DEI SISTEMA LEGACY**

* Le organizzazioni che si affidano ai sistemi legacy devono decidere una strategia di evoluzione per questi sistemi
  + SI possono **smantellare completamente** (costo molto alto) e modificare i processi aziendali in modo tale che non sia più necessario
  + **Mantenere** il sistema
  + **Trasformalo** mediante reingegnerizzazione, migliorando la manutenibilità
  + **Sostituirlo** con un nuovo sistema (Potrebbe costare molto)
* Scelta che dipende dalla qualità del sistema e dal suo valore commerciale

**CATEGORIA DI SISTEMA LEGACY**

* **Bassa qualità, basso valore commerciale** (Per niente utili)
* **Bassa qualità, alto valore commerciale**
  + Forniscono un importante contributo commerciale ma sono costosi da mantenere, dovrebbe essere riprogettato o sostituito se è disponibile un sistema adatto
* **Alta qualità, basso valore commerciale**:
  + sostituire con un COTS oppure rottamare completamente o manutenere
* **Alta qualità, alto valore commerciale**:
  + continuare il funzionamento utilizzando normale manutenzione del sistema