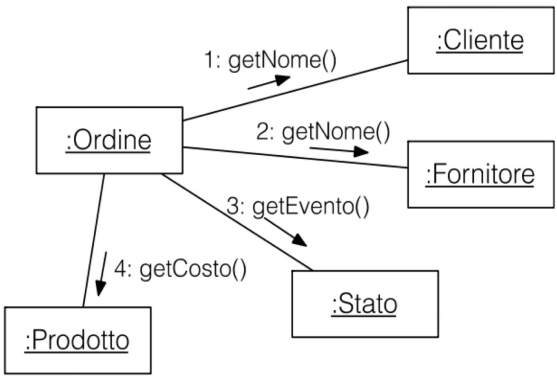
Diagrammi UML DI COLLABORAZIONE:
Ha come obiettivo mostrare il comportamento a run time del sistema software mostrando anche le interazioni fra oggetti

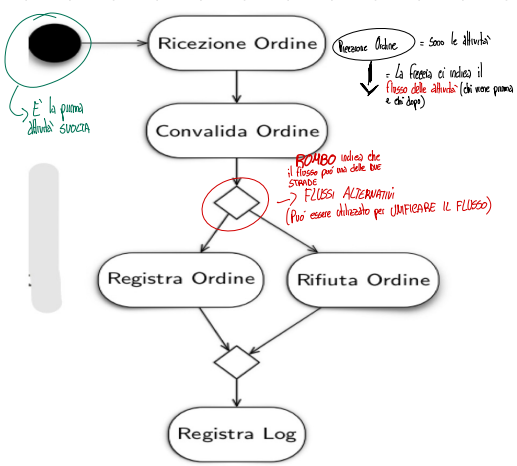
Si hanno Nodi (oggetti) e archi (rappresentano l'interazione tra oggetti.

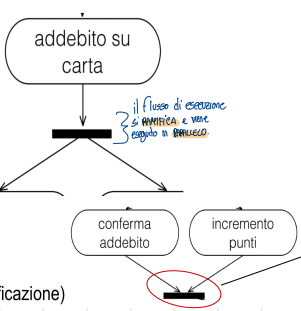


nomeMetodo() → Indica quale metodo é stato invocato sull'oggetto oppure va messa un'INDICAZIONE DI INTERAZIONE.
(Il n° indica l'ordine in cui avvengono)

- Si interagisce con una singola istanza ed é indicata da :Cliente (or c:Cliente) → istanza e di Cliente.
- La disposizione degli oggetti é libera.

Diagramma UML delle Attività:
Mostra le attività e i passi di esecuzione del software. Evidenzia le operazioni piu importanti del software





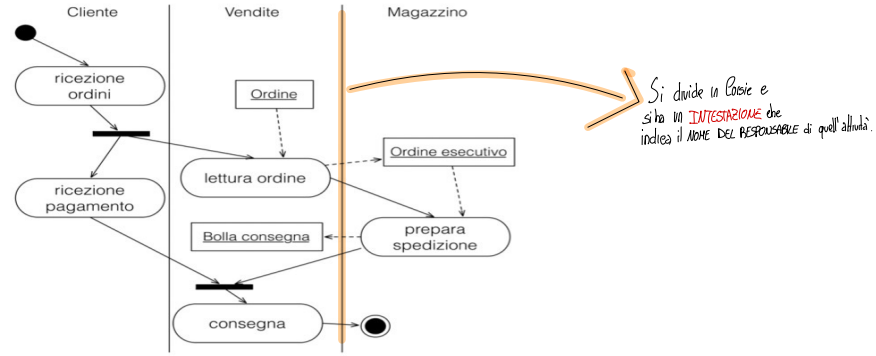
- ***Parallelizzazione*** dei flussi
- ***Sincronizzazione*** dei flussi (riunificazione)

In questo caso bisogna aspettare che vengano terminate entrambe le attività per Andare le ATTIVITÀ

In qu

Diagramma UML con corsie

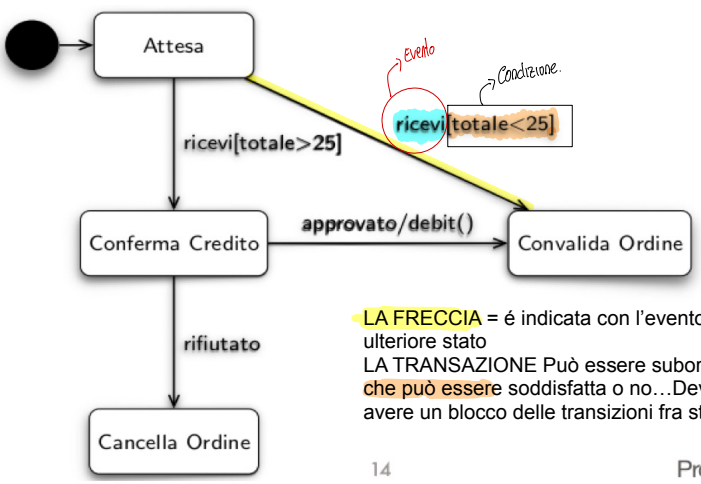
Si guardano solitamente le ***MACRO-attività***. I **responsabili** di determinate **attività** devono essere individuati. Si possono disegnare delle suddivisioni orizzontali/verticali, dove ogni corsia ha un'**intestazione** che indica il **nome del responsabile** di quell'attività.



Ordine è un **dato** ed è ***IN INGRESSO*** (*freccia entrante nell'attività*) o ***RISULTANTE*** dall'attività (*freccia uscente dall'attività*).

Diagramma UML degli stati

Sono rappresentati da rettangoli con i soli ***ANGOLI ARROTONDATI***.



LA FRECCIA = é indicata con l'evento che viene chiamato che fa transitare in un ulteriore stato
LA TRANSAZIONE Può essere subordinata ad un EVENTO o ad una CONDIZIONE che può essere soddisfatta o no…Devono essere ESCLUSIVE in modo da non avere un blocco delle transizioni fra stati.

La freccia è indicata con l'evento che viene chiamato che fa transit

ALCUNE ATTIVITÀ DA FARE PERTINENTI ALLO STATO:
Entry() cerca() —>L'attività in ingressi nella transazione é cerca()

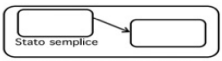
do -> attività da fare durante la permanenza di uno stato

exit() finish() -> l'attività da fare prima di uscire dalla transizione (praticamente é l'ultima attività da effettuare)

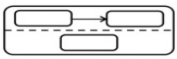
Stati composti

=> Consiste in vari sottostati sequenziali o concreti. => Solo 1 stato può essere attivo in un certo momento. => Stato esterno rappresenta la condizione di essere in un qualsiasi degli STATI INTERNI

All'interno di uno stato *ci sono altri stati* che a loro volta, ovviamente, possono essere semplici o composti:



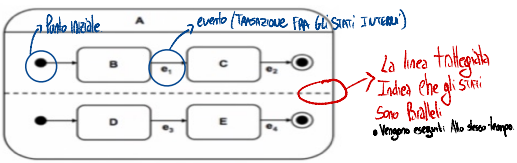
Stato composto sequenziale



Stato composto concorrente

L'**etichetta** si può mettere sempre in alto (come negli stati semplici) oppure **si può omettere** *senza annotarlo nel diagramma* stesso ma da qualche altra parte.

- Occorre specificare ***uno stato iniziale*** (*non rappresentato nel disegno soprastante*) e si usa sempre ***un pallino nero*** collegato con una ***freccia*** ad uno stato che viene considerato iniziale.
- La **transizione fra gli stati interni** si ***indica allo stesso modo*** con una freccia etichettata dal ***nome dell'evento***



Gli stati composti possono essere **sequenziali** o **paralleli** (***linea tratteggiata orizzontale***).

- Vengono assunti allo stesso tempo (diversamente dal sequenziale -> prima uno stato S1 e poi uno stato S2)
- In questo caso, quando si entra nello stato generale (*il più grande*) **si entra contemporaneamente** negli stati sopra e sotto

Processi di sviluppo del software

PROCESSI DI SVILUPPO SOFTWARE:

Sono delle DESCRIZIONI per lo sviluppo di un sistema software.

- Analisi dei requisiti (specifiche)
- Progettazione(design) -> si determinano quale componenti servono per sviluppare il software (si usano gli UML per avere idee chiare)
- Codifica o Implementazione (codice)-> si scrive il codice inerente al diagramma strutturato durante la progettazione
- Convalida o Testing (approvazione) -> si testa il software con vari input per vedere se gli output sono quelli aspettati
- Manutenzione (il codice deve essere elaborato in modo tale da poterlo modificare facilmente)

Specifiche

Analisi dei requisiti:

Si va a definire quello che il software dovrà fare e questo rappresenta proprio una specifica.

Specifica -> DESCRIZIONE RIGOROSA di una caratteristica software (molti dettagli scritti bene)

Requisito -> CARATTERISTICA SINGOLA che il software deve avere.

Per mettere a punto i requisiti si fa:

- STUDIO FATTIBILITÀ -> vediamo cosa richiede il cliente e verificare se si é capaci di costruire un software o no (si può rifiutare se il cliente non ha le idee chiare o non può essere soddisfatto in termini di costo/tempi)
- ANALISI DEI REQUISITI
- SPECIFICHE DEI REQUISITI -> Si ha un documento dettagliato dei requisiti con una buona organizzazione dei dati
- CONVALIDA DEI REQUISITI -> Rilettura dei requisiti da parte dello sviluppatore e cliente dove si vanno a correggere eventuali errori.

I requisiti si distinguono in:

- FUNZIONALI -> COSA il Software dovrà fare.
- NON FUNZIONALI -> COME il software lo dovrà fare

Progettazione dettagliata del sistema:

In questa fase si determinano le CLASSI che servono, INTERFACCE; RELAZIONI TRA CLASSI….

Implementazione:

Si devono ricavare algoritmi sulla base delle classi della progettazione e rimuovere i difetti.Il programma scritto va anche testato per analizzarne il comportamento.

Convalida e Test:

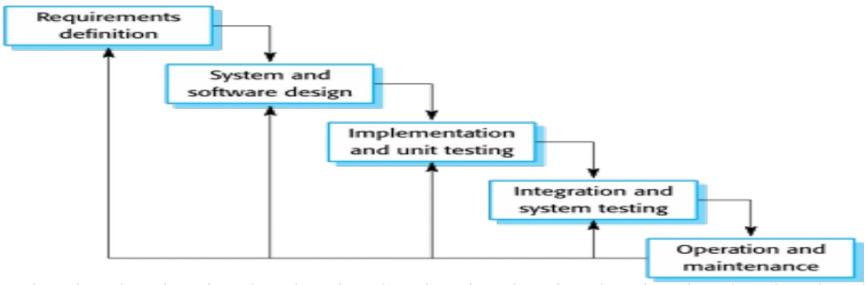
Si leggono i requisiti e si prova il software secondo questo documento, in ogni caso dobbiamo migliorare il codice affinché il programma possa adattarsi a tutti i requisiti del cliente.

Si Fanno Vari Test:

- Test di unità—> si devono testare le interazioni fra classi/metodi e vengono scritti durante l'implementazione
- Test di Sistema —> Si devono testare le interazioni fra classi/metodi
- Test di accettazione —-> si simula il funzionamento del sistema con i dati forniti dal cliente.
- Beta Test -> viene fornito il software a pochi clienti e si verifica l'adattabilita su i sistemi diversi attraverso il feedback dei clienti.

VERSIONE ALPHA (con difetti e parti mancanti)
VERSIONE BETA (con difetti)
VERSIONE GOLD —-> il software ha superato tutti i test..

PROCESSO DI SVILUPPO A CASCATA:



Le fasi descritte prima per essere eseguite vengono eseguite in modo ordinato.

-Per passare alla fase successiva bisogna aver COMPLETATO LA FASE PRECEDENTE
- Le modifiche si effettueranno solo dopo la manutenzione.

Un punto di forza é nella fase iniziale, in particolare se si hanno dei requisiti stabili allora lo sviluppo del software avverrà con facilita e di qualità per tutte le fasi successive

VANTAGGI DEL PROCESSO DI SVILUPPO IN CASCATA:

- Viene usato per sistemi grandi, complessi, CRITICI per garantire una qualità del prodotto.
- Vi é un'ampia documentazione.
- Utile se i requisiti sono stabili e chiaramente definiti.
- C'é un grande TEAM DI SVILUPPO e, ogni volta che viene prodotto del codice, le varie componenti vengono integrate nel sistema operativo.

SVANTAGGI DEL PROCESSO DI SVILUPPO IN CASCATA:

- La durata della raccolta dei requisiti é LUNGA
- Si hanno poche interazioni con i clienti visto che esse avvengono solo durante la fase della raccolta dei requisiti.
- Non é facile introdurre i cambiamenti richiesti dal cliente...si ignorano le richieste di ca,bramenti le cliente.

Questo sviluppo non é "forte" in caso rilasci versioni di prova del software durante lo sviluppo.
Il prodotto viene consegnato direttamente alla fine.

PROCESSI DI SVILUPPO EVOLUTIVI:
Si chiamano così perché si procede nello sviluppo per evoluzione per modifiche che si fanno al software che si é realizzato.

PER ESPLORAZIONE:

Il processi di sviluppo PER ESPLORAZIONE:

- Si lavora a stretto contatto con il cliente per tutta la durata dello sviluppo del software, dove il cliente dara delle Specifiche iniziali.
- Sulle specifiche fornite si progetta il codice.
- Questa prima parte del codice viene mostrata al cliente e lui stesso verifica la correttezza e può aggiungere ulteriori dettagli.

Anche se le specifiche sono CHIARE, non vi sarà una visione globale del progetto perché le specifiche vengono sviluppate dalle singole parti.
Solo piu avanti si avrà una visione piu globale.

BUILD AND FIX:

- Si progetta e si sistemano i problemi nello stesso tempo.
- In particolare, quello che si sviluppa scaturisce da requisiti NON CHIARI all'inizio(dovuti da difficolta del progettista o il cliente che non riesce a descrivere i requisiti chiaramente).
nonostante si continua a sviluppare il codice e si va anche a modificare finche il cliente é soddisfatto
- Si ha una compressione limitata del sistema da produrre e la fase di design é pressoché INESISTENTE
- il codice prodotto é di BASSA QUALITÀ

Consigliato per eventuali prototipi che rappresentano delle versioni non complete che mostrano al cliente che si può arrivare a un determinato sistema software.
La realizzazione di prototipi é veloce dove si può adottare

Problemi e applicazioni:

-PROBLEMI
- i tempi sono LUNGHI
- Il codice é di BASSA QUALITA
- Il COSTO é DIFFICILMENTE STIMABILE inizialmente.

-APPLICAZIONI:
- Sistemi di PICCOLE DIMENSIONI
- Singole parti di sistemi grandi
- Sistemi con VITA BREVE

CBSE (Componenti-Based Software Engigneering):

Realizzazione Software basata su componenti.

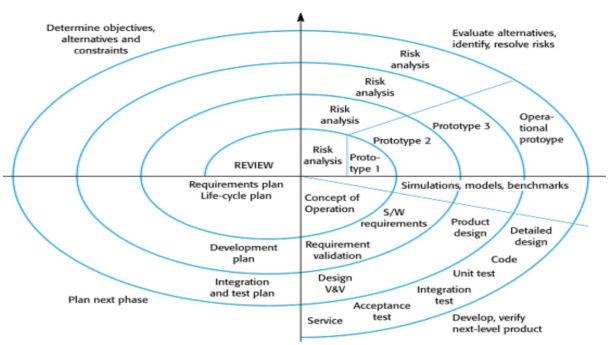
Si usano componenti già esistenti e realizzati. Si fa un RIUSO DEI COMPONENTI per un successivo software da realizzare.

- Si raccolgono i requisiti dal cliente e si cerca di trovare una corrispondenza tra i requisiti del cliente e quelli che si hanno.
- Si propone al cliente una variazione dei requisiti per vedere se é possibile interagire con componenti già usate in modo da arrivare ad un accordo bilaterale.
- Per tutte le variazioni accettate dal cliente si devono solo integrare le componenti che si vogliono usare che già esistono.
- Si cerca di essere VELOCI nello sviluppo e di rendere MENO COSTOSA la realizzazione.

A SPIRALE:

Tutti i processi di sviluppo prendono esempio da altri processi utilizzati da altre aziende e si valuta se aggiungere o togliere passaggi.

Tutto é organizzato secondo questa forma geometrica. I vari passaggi vengono fatti tramite dei cicli dette FASI in senso ORARIO.



Ogni quadrante dell'asse cartesiano si preoccupa di un determinato processo in fase di sviluppo..

Nel primo (alto a sinistra) si fa una PIANIFICAZIONE DEGLI OBIETTIVI:
-si decido gli obiettivi di un solo ciclo
- si valutano le priorità
-In questa fase si può pianificare anche un'eventuale raccolta dei requisiti.

Nel secondo adorante si fa una valutazione dei RISCHI:
-Si analizza ciò che potrebbe non essere soddisfatto quando si procede con le due fasi successive e questi rischi devono essere valutati e, in questo caso si possono prendere delle precauzioni

Nel terzo quadrante si fa la PRODUZIONE:
- Si fanno le attività per gli obiettivi proposti durante la fase di Pianificazione.

Nel quarto quadrante si fa la REVISIONE:
- Si verifica che tutti gli obiettivi stabiliti sono stati raggiunti durante la fase di PRODUZIONE

Un ciclo aspirale dura parecchio (6+ mesi) ma non tutti i cicli hanno la stessa durata.
Vi é una grande interazione col cliente.
Il tempo di realizzazione é ampio.(2 anni o piu per rilasciare il software)

Processo di Extreme Programming:

XP é un manifesto di un processo di sviluppo che contiene i dettagli:

- Si deve porre l'accento su certe cose piuttosto che su altro
- Quando si struttura un software dobbiamo concentrarci sull'importanza degli individui.
- Bisogna collaborare con il cliente
- La priorità tende alla capacita di essere AGILI nello sviluppo del software (incorporare diverse richieste del cliente nel software) che comportano cambiamenti senza stravolgere l'intero codice.

Il processo di sviluppo XP ha le seguenti caratteristiche:

Il processo di sviluppo XP ha le seguenti CARATTERISTICHE:

- poco tempo per lo sviluppo incrementale
- Piccolo team per ogni incremento
- Poca documentazione
- Costante miglioramento
- Molta comunicazione fra team lavorativo
- Tanta interazione
- Tanti test del codice

Ciò permette di avere tante piccole release di ottima qualità.

Principi di XP

- Si ha un **commento rapido** di quello che si sta sviluppando da parte del cliente
- Si fanno **cose semplici**
- Si hanno dei **cambiamenti graduali** supportati
- Si produce **codice di qualità**

Le **PRATICHE XP** sono 12 che consentono di produrre codice di qualità, di essere *AGILI* ed è adottato da tutte le aziende.

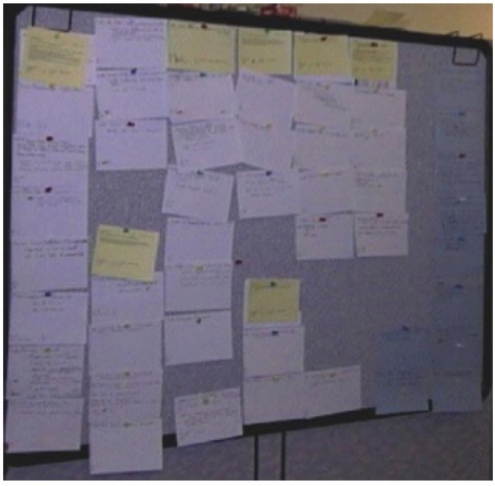
- Gioco di pianificazione
- Piccole release
- Metafora
- Testing
- Refactoring
- Pair Programming (programmazione a coppie)
- Cliente in sede
- Design semplice
- Possesso del codice collettivo
- Integrazione continua
- Settimana di 40 ore
- Usare gli standard per il codice

Gioco di pianificazione

*Questa fase produce dei requisiti **scritti bene**, sono **granulari** e **comprensibili**.*

Si fa quando si vuole sviluppare **codice di alta qualità**. Si raccolgono i requisiti e si categorizzano dando loro una valutazione. Tale raccolta viene svolta nel seguente modo:

- I clienti si riuniscono con gli sviluppatori (*che sono **pochi***) al fine di raccogliere i requisiti
- Il cliente scrive un **singolo requisito** (detta *storia*) sulla **STORYCARD** (una scheda piccola, *12x7* cm). Quest'ultima è piccola affinchè il cliente si focalizzi su una **singola cosa**
- La scheda viene **passata** ai sviluppatori che si fanno un'idea sul requisito
- Nel frattempo il cliente scrive altri requisiti. Alla fine si raccolgono un certo numero di schede

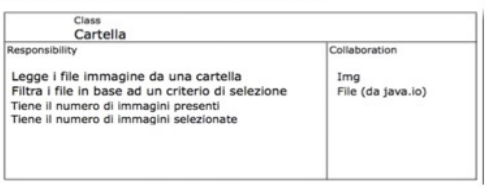
- Si fa una stima per la realizzazione di un requisito, quindi di una sola scheda.
- Le **schede incomprensibili e che richiedono troppo tempo (*oltre ai due giorni*) vengono strappate**. Se il tempo è troppo lungo allora sta a significare che il requisito comprende anche altri *sotto-requisiti* al suo interno
- **Si scelgono delle priorità** alle storieda parte del cliente (*in base ai propri requisiti*) e degli sviluppatori (*in base alla loro esperienza*) e quindi si cerca di arrivare a un *compromesso*.
- Si fa una ***stima generale del tempo necessario*** per sviluppare tutte le storycard *cercando di non superare le **3 settimane***
  - Durante questo periodo di sviluppo è possibile far subentrare qualche modifica al codice visto che il tempo di sviulppo è molto poco
- Si fa una pianificazione **più precise nelle prime 2 settimane**. La terza settimana è per quello che deve essere fatto dopo (**meno prioritario** e con una **stima grossolana**).
- Si fornisce una **piccola release dopo le 2 settimane**, chiedendo dei commenti al cliente e dopodiché si rifà il gioco di pianificazione.
  - Quello che era stato pianificato per la terza settimana possono diventare priorità per il prossimo ciclo di 2 settimane oppure possono essere cambiate durante il nuovo gioco di pianificazione
- Bisogna scegliere cosa fare all'inizio **in base ai rischi**. Se si tratta di una cosa difficile da realizzare o comunque si tratta di algoritmi sofisticati allora deve avere **priorità maggiore** e quindi realizzata durante il primo sviluppo per avere tempo per fare i dovuti fix.
- Inoltre, *ogni story deve poter essere testata* in modo da *validare* ciò che si produce
- Le *StoryCard* vengono messe nella **StoryBoard**
  
- Le storyboard ha 3 colonne:
  - Ogni colonna rappresenta una settimana
  - Le storycard da realizzare si mettono in alto mentre le storycard realizzate si mettono in basso

Design semplice

Quando la coppia di programmatori legge la storycard pensano alle dovute classi che servono e agli algoritmi da usare.
Si pensa alle **classi e algoritmi più semplici possibili**. Non si fa più progettazione del dovuto ma viene fatta *al volo*.

Caratteristiche:

- Il codice deve essere di qualità
- Classi **piccole e modulari**
- Quello che si produce deve superare i test
- Non ci sono parti duplicate
- Si deve **esprimere**, nel codice, **l'intento** in **maniera chiara**
- Non ci si preoccupa molto di quello che si implementa inizialmente (*quindi si prende una decisione e si applica*) perchè comunque in futuro potrebbero essere necessarie delle modifiche
- Le **scelte della progettazione** (*rapida*) sono **documentate nel *CRC***(*Class Responsability Collaboration*) che è una scheda (*circa grande quanto la storycard*) e contiene:
  - Il **nome** della classe
  - Le sue **responsabilità**
  - Le **interazioni** con le altre classi



> ***In XP non si fa uso di diagrammi UML.***

Testing

Bisogna **testare sempre tutto**. I test vengono eseguiti appena si pensa che il codice è completo e vengono fatti **in locale**. Dopodiché vengono **depositati** su sistemi remoti dell'azienda (*Git*).

Successivamente vengono scritti ede seguiti i **test di sistema** (*o di integrazione*)che vengono eseguiti **sull'intera applicazione**, quindi test delle classi che interagiscono fra loro.

(**ESAME**) *Quando si eseguono i test nel processo XP?* Più volte al giorno:

- Quando la coppia di programmatori finiscono di implementare la storycard o nel frattempo
- Quando il codice è condiviso sulla repository (push)

I **test** rappresentano la **specifica dei requisiti** (in formato eseguibile).

Unit test: test delle singole unità o singolo metodo
Test funzionali: test dell'integrazione fra più parti del codice

I test possono essere **scritti prima di scrivere il codice**, ovvero applicare la **TDD**(*Test Driven Development*)

Possesso collettivo del codice

- Chiunque partecipi al progetto può leggere e modificare il codice sorgente.
- Ogni membro del team è responsabile per l'intero sistema
- Visto che il codice viene spesso modificato, i test **proteggono** le funzionalità del sistema

Integrazione continua

Man mano che si sviluppa del codice non ci si deve preoccupare solo di quello che si scrive ma **si deve integrare con il codice dell'intero software**.

- Quando la coppia fa l'integrazione può capitare che questa non vada a buon fine. Quindi si potrebbe decidere di buttare il codice prodotto e ricominciare
- Se le integrazioni (e quindi lo sviluppo del codice) si fanno con una durata di 6-8 ore e poi il codice si deve buttare, allora vuol dire che si sono sprecate solo quelle poche ore di lavoro