Lezione 4- Sviluppo Agile del Software

Riferimenti bibliografici

- I. Sommerville – Ingegneria del Software – 8a edizione – Cap. 17

- R. Pressman- Principi di Ingegneria del Software- 4 edizione- Cap. 4

Argomenti

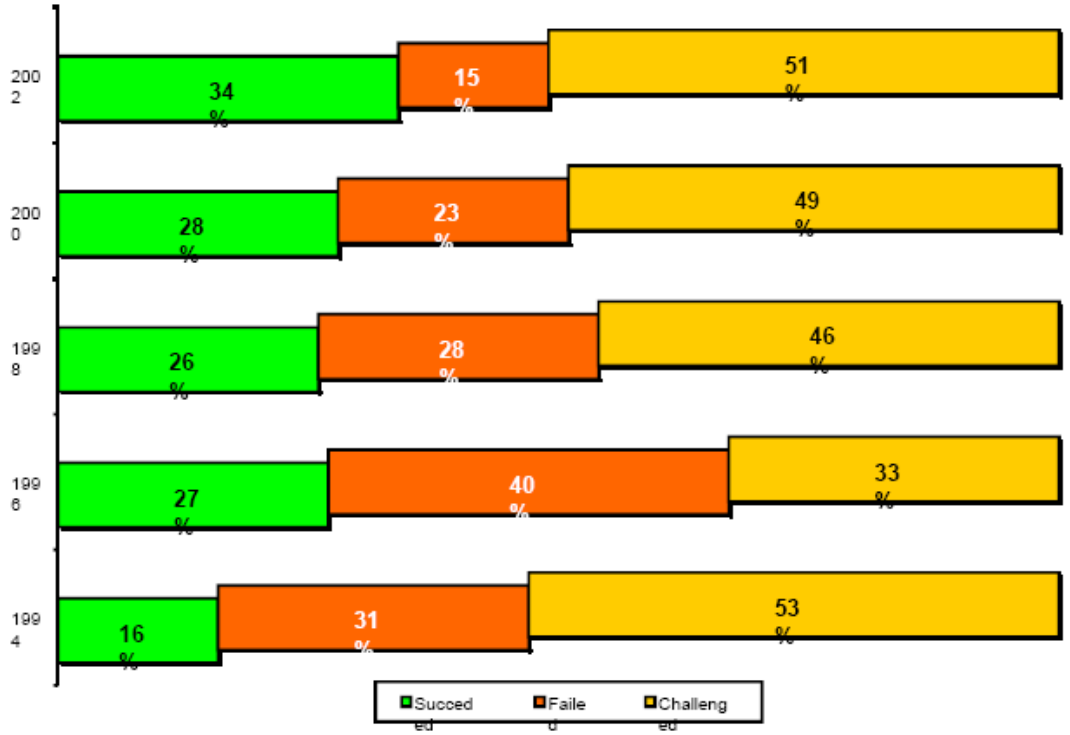
1. Introduzione allo Sviluppo Agile

1. eXtreme Programming (XP)

Premessa-
Molti progetti software falliscono!

CHAOS Report, Standish Group

- 66% dei progetti falliti, o contestati nel 2002
- Progetti grandi falliscono più spesso dei piccoli
- Solo il 52% delle features richieste vengono prodotte



http://www.standishgroup.com/sample_research/chaos_1994_1.php
http://www.standishgroup.com/sample_research/PDFpages/chaos1998.pdf

Metodi Agili

- L'insoddisfazione per l'eccessivo overhead richiesto dai tradizionali approcci allo sviluppo software ha portato negli anni '90 alla proposta dei metodi agili.
- Tali metodi:
  - Si concentrano sul codice, piuttosto che sulla progettazione;
  - Sono basati su un approccio iterativo allo sviluppo software;
  - Sono pensati per rilasciare software funzionante rapidamente, e per farlo evolvere rapidamente per soddisfare nuove esigenze.
- *Il movimento dell'Agile Software Development nasce nel febbraio 2001 in Utah*

Il Manifesto dello Sviluppo Agile

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- ***Individuals and interactions* over processes and tools**
- ***Working software* over comprehensive documentation**
- ***Customer collaboration* over contract negotiation**
- ***Responding to change* over following a plan**

Kent Beck et al

Cos'è l' "Agilità"?

- Efficace risposta (rapida e flessibile) ai cambiamenti
- Efficace comunicazione fra tutti gli stakeholders
- Portare il cliente nel team di lavoro
- Organizzare un team in modo da controllare il lavoro svolto

Consente di avere …

- Rapida, incrementale consegna del software

Un Processo Agile

- E’ guidato dalle descrizioni dei clienti di quanto si richiede al software (scenari)
- Riconosce che i piani hanno vita breve
- Sviluppa il software iterativamente dando enfasi alle attività di costruzione
- Rilascia multipli ‘incrementi software’
- Si adatta a fronte dei cambiamenti

- Esistono diverse proposte di Metodi Agili

Alcuni Metodi Agili

- Extreme Programming (Kent Beck 1999)
- SCRUM (Sutherland)
- Feature Driven Development (De Luca & Coad)
- Crystal (Cockburn)
- DSDM (Dynamic System Development Method)
- Lean Software Development (Poppendieck)

- Ogni metodo propone un diverso processo, ma tutti condividono gli stessi principi

Principi dei Metodi Agili (1)

- Coinvolgimento del cliente
  - Il cliente dovrebbe partecipare attivamente allo sviluppo. Il suo ruolo consiste nel definire i requisiti e assegnarvi le priorità, e nel valutare le iterazioni del sistema.
- Consegna Incrementale
  - Il software è sviluppato in incrementi, ed il cliente stabilisce i requisiti da includere in ciascun incremento.
- Persone, non processi
  - Le capacità del team di sviluppo devono essere riconosciute e sfruttate. Le persone devono poter liberamente sviluppare con i propri metodi, senza imposizioni di processi prescrittivi

Principi dei Metodi Agili (2)

- Accettare i cambiamenti
  - Prevedere i requisiti che cambieranno e progettare il sistema in modo da accettare i cambiamenti
- Mantenere la semplicità
  - Semplicità sia nel software che si sviluppa, che nel processo adottato.

Agile Modeling

- Adottare un metodo agile non vuol dire evitare di fare modellazione; anche XP accetta la modellazione, purché agile;
- lo scopo dei modelli e della modellazione è supportare la comprensione e la comunicazione;
- non modellare tutto;
- usare gli strumenti più semplici possibili;
- non modellare da soli;
- creare modelli in parallelo;
- sapere che tutti i modelli sono incompleti e imprecisi.

Tipico Sviluppo Agile

- Le Applicazioni evolvono attraverso multiple brevi iterazioni
- Le Iterazioni hanno durata costante, variabile fra 2-13 settimane
- Si rilascia una applicazione funzionante al termine di ogni iterazione
- Con ogni nuova release si aggiungono quante più caratteristiche è possibile tra quelle con la più alta priorità richieste dal cliente
- Per ogni release vengono elaborati solo i requisiti ed il progetto necessari a supportare le caratteristiche di quella release.
- Si testano in maniera intensiva le caratteristiche in ogni iterazione
- Il cliente rivede ogni nuova release— e può ridefinire le priorità per la prossima iterazione.
- Si tiene sotto controllo il progresso del progetto attraverso le caratteristcihe completate.
- Non si fallisce mai una data di consegna, piuttosto si rimanda lo sviluppo di qualche feature.

Potenziali Vantaggi dell'Agile

- Consegne più predicibili
- Veloce ritorno dell'investimento; il software viene consegnato e va in uso più rapidamente
- Rapida risposta ai cambiamenti dei bisogni utente
- Rischi attenuati grazie a cicli di consegna più brevi
  - Più opportunità di scoprire errori
  - Convalida dei requisiti
  - Conferme sull'approccio tecnico adottato
  - Verifica realistica dei progressi del lavoro
- Alta produttività e qualità
- Clienti soddisfatti, successo dei progetti

Outline della lezione

1. Introduzione allo Sviluppo Agile

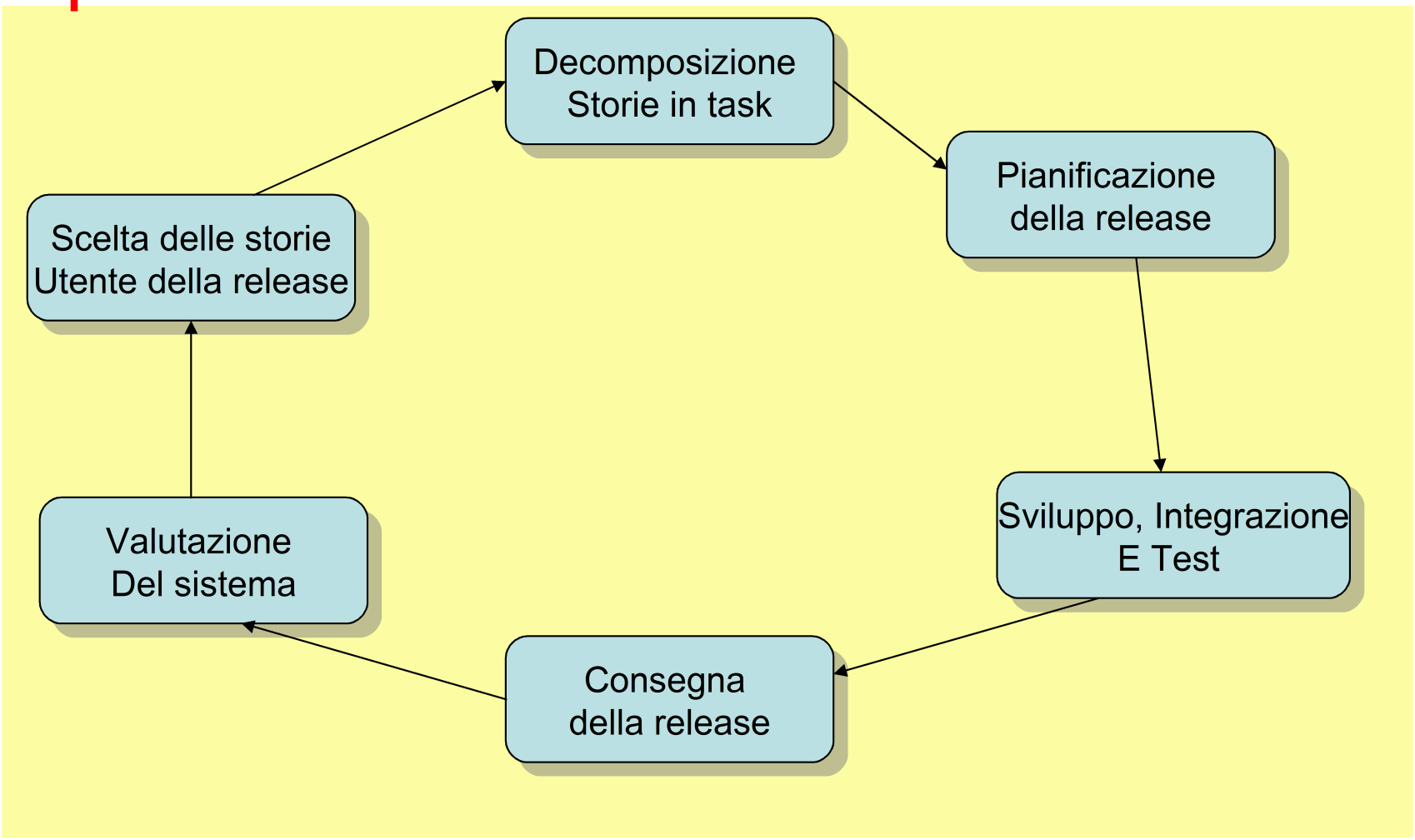
1. **eXtreme Programming (XP)**

eXtreme Programming

- Forse è il metodo agile più noto ed usato.

- Extreme Programming (XP) adotta un approccio estremo allo sviluppo software.
  - L'approccio si basa su requisiti espressi come scenari (storie utente) che sono implementati direttamente;
  - Nuove versioni possono esser prodotte più volte al giorno;
  - Gli incrementi sono rilasciati al cliente frequentemente, in genere ogni 2-4 settimane;
  - Ogni nuova versione viene completamente testata, e viene accettata solo se supera tutti I test.

Il processo XP



Pratiche dell'Extreme Programming

1. ***The Planning Process (o*** Planning Game).
   - La pianificazione del processo in XP consiste nella definizione del *business* value di ogni caratteristica desiderata e nella stima dei costi per decidere in che ordine le caratteristiche debbano essere realizzate. Si tratta quindi di una strategia estremamente duttile, che può essere ripensata durante lo svolgimento del progetto.
1. ***Small Releases.***
   - La vita e lo sviluppo dell'applicazione sono scanditi dai rilasci di versioni del prodotto funzionanti. Ogni rilascio rappresenta il punto conclusivo di un'iterazione di sviluppo e l'inizio di una nuova pianificazione. Per poter tener conto di cambi di prospettiva, errori di valutazione, nuovi requisiti, restrizioni di bilancio, ogni iterazione dovrebbe durare non più di qualche settimana (in genere, da due a quattro).

Pratiche dell'Extreme Programming

1. ***Metaphor.***
   - I team di lavoro XP condividono un "sistema dei nomi", nel quale siano compresi I concetti fondamentali relativi al software sotto sviluppo (in pratica il dominio dei dati)

1. ***Simple Design.***
   - Un programma costruito con XP dovrebbe essere il programma più semplice in grado di soddisfare i requisiti. Non si cercano soluzioni che possano portare benefici in un prossimo futuro. La qualità del progetto é comunque tenuta alta, grazie anche alle attività di refactoring

Pratiche dell'Extreme Programming

1. ***Testing.***
   - Il testing é un elemento fondamentale dell'XP. I programmatori devono scrivere i test contemporaneamente alla scrittura del programma stesso (strumenti come XUnit sono particolarmente utili). Il testing di accettazione invece é demandato al cliente finale

1. ***Refactoring.***
   - Durante le fasi di integrazione del codice appena prodotto con il resto del sistema, bisogna operare in maniera da tenere alta la qualità della progettazione, evitando ridondanze dei dati e duplicazioni del codice. Fondamentale rimane ancora la comunicazione tra I vari gruppi

Pratiche dell'Extreme Programming

1. ***Pair Programming.***

   I programmatori XP lavorano in coppia: due programmatori sulla stessa macchina. Uno dei programmatori é addetto alla scrittura fisica del codice, mentre l'altro ne controlla la correttezza, salvo scambiarsi i ruoli di tanto in tanto. Esperimenti dimostrano come la produttività sia superiore rispetto a quella di due programmatori separati.

1. ***Collective Ownership.***

   Ogni programmatore deve essere in grado di avere un'idea generale del progetto e di poter toccare qualunque punto del codice, anche di quello che non ha scritto lui personalmente. Quest'obiettivo può essere raggiunto seguendo il principio di *Simple Design*

1. ***Continuous Integration.***

   Il processo di integrazione del sofware con i nuovi moduli avviene più volte al giorno. Ciò aiuta I programmatori ad essere sempre aggiornati sullo stato del progetto e riduce la maggior parte dei gravi problemi di integrazione che si verificano quando questa fase non é eseguita così spesso.

Pratiche dell'Extreme Programming

1. ***40-hour Week.***
   - I programmatori stanchi commettono più errori. Essi non dovrebbero fare troppo straordinario, in modo da essere freschi, in salute e produttivi
1. ***On-site Customer.***
   - Il committente deve essere coinvolto nello sviluppo perché è l'unica fondamentale fonte di convalida del sistema. Partecipa perciò alla stesura dei test di sistema e verifica, periodicamente, che il sistema realizzato corrisponda effettivamente alle proprie esigenze.Inoltre, è la principale fonte di informazione per la conoscenza sul dominio di applicazione.
1. ***Coding Standard.***
   - I programmatori dovrebbero condividere uno stesso stile di scrittura del codice, in modo da rendere più semplice la comunicazione

XP e i principi agili

- Lo sviluppo incrementale è realizzato attraverso release piccole e frequenti.
- Il coinvolgimento del cliente nel processo di sviluppo deve essere a tempo pieno.
- Si da' attenzione alle persone (e non al processo) attraverso la programmazione a coppie, il possesso collettivo del codice e con tempi di lavoro non eccessivo.
- Le modifiche sono supportate da release costanti del sistema.
- Il mantenimento della semplicità è raggiunto attraverso pratiche continue di refactoring.

Coinvolgimento del cliente

- Il coinvolgimento del cliente è essenziale in XP.
- Il ruolo del cliente è molteplice:
  - Aiuta a sviluppare storie che corrispondono ai requisiti
  - Aiuta a definire le caratteristiche da includere in ogni release
  - Aiuta a sviluppare I test di accettazione per controllare che il sistema soddisfi I suoi requisiti.

Scenari dei requisiti

- In XP, I requisiti sono espressi come scenari, le cosiddette storie utente.
- Gli scenari sono scritti su schede (Story Card) e il team di sviluppo le suddivide in task da implementare. Tali task sono la base per la stima dei costi e la schedulazione del lavoro.
- Il cliente sceglie le storie da includere nella prossima release sulla base della loro priorità e del costo stimato.

Esempio di Story Card per la generazione di un ordine

Il cliente inserisce il proprio codice cliente e richiede la visualizzazione del catalogo degli articoli disponibili.

Il cliente sceglie gli articoli da ordinare, specifica le quantità richieste, e conclude l'ordine

Il sistema calcola l'importo totale dell'ordine

Il cliente conferma l'ordine e inserisce la modalità di pagamento (carta di credito o conto pre - autorizzato)

Il sistema verifica i dati inseriti, acquisisce l'ordine e visualizza il numero di giorni previsti per la consegna

Il sistema genera una copia dell'ordine in PDF e la invia all'indirizzo del cliente

XP e le modifiche software

- Un principio basilare dell'ingegneria del software è di progettare pensando al cambiamento. Si dovrebbero spendere risorse e tempo per anticipare le possibile modifiche, riducendo così I costi di manutenzione futura.
- Invece in XP si rinuncia a gestire in anticipo i cambiamenti, perchè si ritiene che le modifiche non si possano prevedere con certezza.
- In alternativa, XP propone un continuo miglioramento del codice (refactoring) per rendere più semplice l'implementazione delle modifiche.

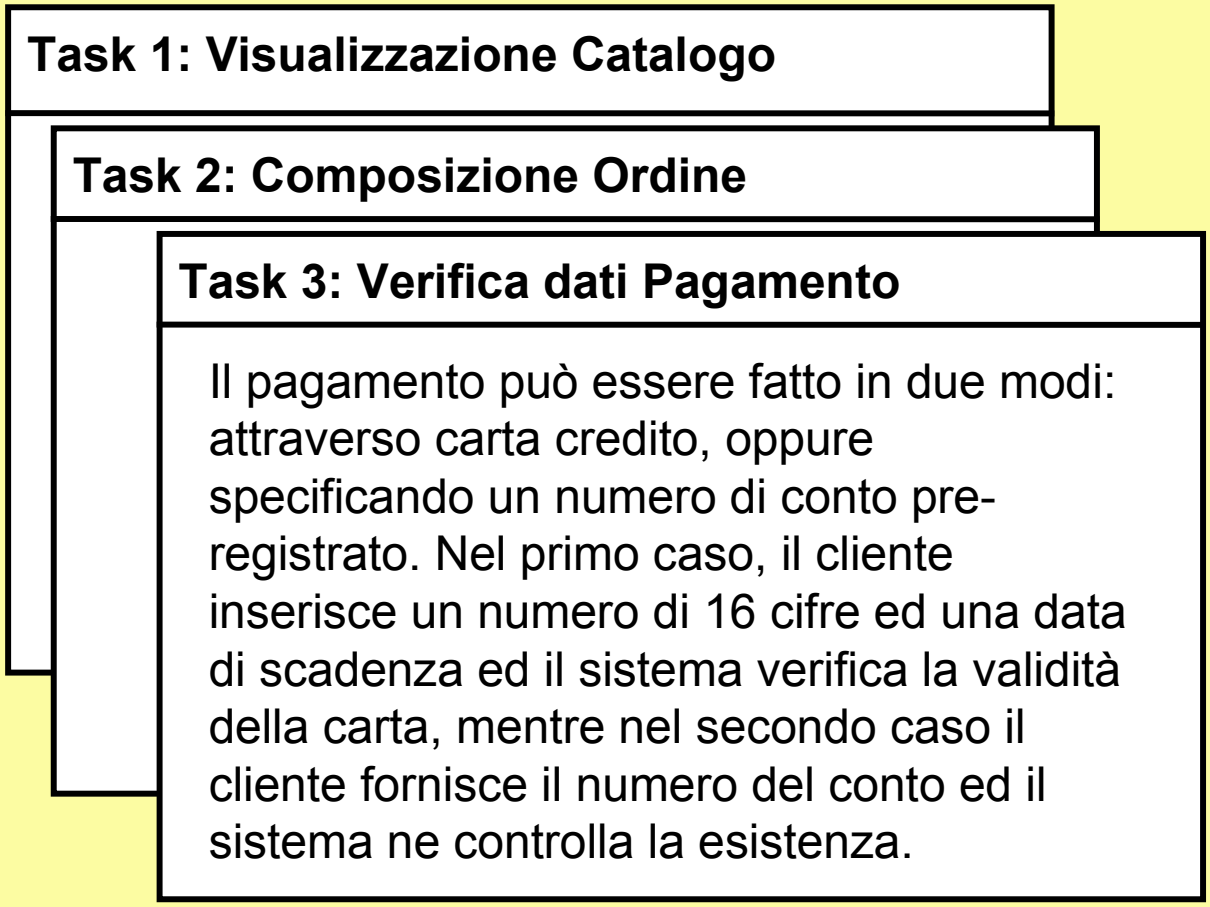
Refactoring

- Refactoring è un processo di miglioramento del codice che viene riorganizzato e riscritto per renderlo più efficiente,comprensibile, etc.
- Il Refactoring è necessario perchè con i rilasci frequenti il codice (sviluppato incrementalmente) tende a deteriorarsi .
- Il Refactoring non dovrebbe cambiare la funzionalità del sistema.
- Il testing automatico semplifica il refactoring perchè consente di verificare che il codice modificato superi ancora i test con successo.

Testing in XP

- Negli approcci rapidi è difficile che il software possa essere testato da un team esterno (manca la necessaria documentazione)!
- XP dedica invece molta attenzione al testing e richiede di progettare i test prima di scrivere il codice.
- Lo sviluppo dei test avviene in modo incrementale a partire dagli scenari.
- Coinvolge l'utente nello sviluppo dei test e nella validazione.
- Si automatizza l'esecuzione dei test in modo da testare tutte le unità ogni volta che viene prodotta una nuova release.

Task cards per la creazione dell'ordine



Descrizione dei Test case

Test 3: Test per il controllo validità della carta di credito

Input: una stringa di 16 caratteri e due interi (per mese e anno di scadenza)

Test:
Controllare che tutti i caratteri della stringa siano interi
Controllare che il mese sia compreso fra 1 e 12 e l'anno sia maggiore o uguale di quello corrente
Controllare la validità della carta facendone richiesta al gestore della carta

Output: OK, oppure un messaggio di errore per carta non valida

Sviluppo preceduto dai Test

- Scrivere i test prima del codice chiarisce i requisiti da implementare.
- I test sono scritti come programmi, per essere eseguiti automaticamente. Ogni test simula l'invio degli input e controlla l'output.
- Sia i test preesistenti che i nuovi test vengono eseguiti quando una nuova funzionalità viene aggiunta. Ciò permette di verificare che la nuova funzionalità non abbia introdotto errori.

Pair programming- Programmazione a coppie

- In XP, I programmatori lavorano in coppie, e le coppie variano continuamente.
- Aiuta a sviluppare il senso di proprietà del codice e a diffondere la conoscenza nell'ambito del team.
- Permette un processo di revisione informale, perchè ogni linea di codice è controllata da più di 1 persona.
- Il refactoring viene incoraggiato, perchè è più facile che l'intero team ne benefici.
- Le misure eseguite mostrano che il pair programming richiede uno sforzo simile a quello di 2 persone che lavorano indipendentemente.

Problemi dell'XP

- Coinvolgimento del cliente
  - É forse il problema maggiore. Può essere difficile trovare un cliente rappresentativo di tutti gli stakeholder e che possa lasciare il normale lavoro per far parte del team di XP.
  - Per sviluppare prodotti generici, non esiste uno specifico cliente, rappresentativo degli utenti finali.

Problemi dell'XP

- Progetto architetturale
  - Lo stile incrementale di sviluppo comporta che possano farsi scelte architetturali inadeguate nelle prime fasi del processo.
  - I problemi conseguenti potrebbero diventare evidenti solo dopo l'implementazione di molte funzionalità, quando eseguire il refactoring dell'architettura può essere costoso.

- Compiacimento per i Test
  - Può sembrare scontato al team pensare che essendoci molti test il sistema sia stato testato adeguatamente (invece..).
  - A causa del testing automatico, c'è la tendenza a sviluppare test facili da automatizzare, piuttosto che test realmente validi (ma difficili da automatizzare).