Sviluppo Agile [Cockburn 2002]

- Sono più importanti auto-organizzazione, collaborazione, comunicazione tra membri del team e adattabilità del prodotto rispetto ad ordine e coerenza delle attività del progetto
- Privilegiare
  - Individui *rispetto* a processi e strumenti
  - Disponibilità di software funzionante *rispetto* alla documentazione
  - Collaborazione con il cliente *rispetto* alla negoziazione dei contratti
  - Pronta risposta ai cambiamenti *rispetto* all'esecuzione di un piano
- Agilità
  - Considerare positivamente le richieste di cambiamento anche in fase avanzata di sviluppo
  - Fornire release del sistema software funzionante frequentemente
  - Costruire sistemi software con gruppi di persone motivate
  - Continua attenzione all'eccellenza tecnica

Extreme Programming (XP) [Beck 2000]

- Approccio basato sullo sviluppo e la consegna di piccoli incrementi di funzionalità
  - Solo 2 settimane per lo sviluppo degli incrementi
  - Piccoli gruppi di sviluppatori (da 2 a 12 persone)
  - Costante miglioramento del codice
  - Poca documentazione: uso di Story Card e CRC (Class Responsibility Collabor)
  - Enfasi su comunicazione diretta tra persone
  - Iterazioni corte e di durata costante
  - Coinvolgimento di sviluppatori, clienti e manager
  - Testabilità dei prodotti e prodotti testati sin dall'inizio
- Adatto per progetti in cui
  - I requisiti non sono stabili, XP è fortemente adattativo
  - I rischi sono grandi, es. tempi di consegna brevi, software innovativo per gli sviluppatori

Extreme Programming (XP)

- Princìpi di XP
  - Avere feedback rapidamente
  - Assumere la semplicità
  - Cambiamenti incrementali
  - Supportare i cambiamenti
  - Produrre lavoro di qualità
- Libro Consigliato
  - Beck. Extreme Programming Explained. Addison-Wesley
- Siti web
  - www.xprogramming.com
  - www.extremeprogramming.org

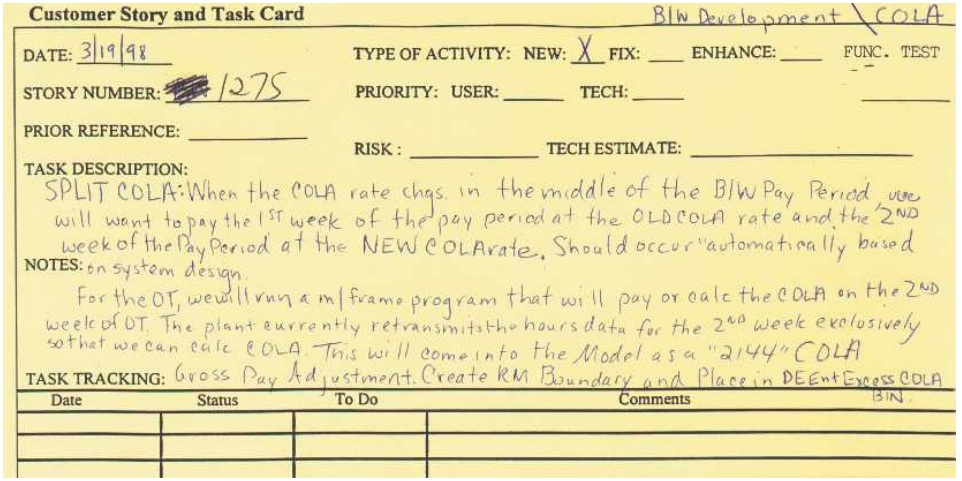
12 Pratiche di XP [Beck]

- 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

Story Card

- Storie utente (story card) = casi d'uso leggeri
- Dimensioni card 5" x 3" circa 12x7 cm
- Descrizione storie: 2-3 frasi su una card che
  - Sono importanti per il cliente e sono scritte dal cliente
  - Possono essere testate
  - Permettono di ricavare una stima del loro tempo di sviluppo
  - Possono essere associate a priorità
- Template per story card (ovvero campi di una story card)
  - Data, Numero, Priorità, Tempo stimato, Riferimenti
  - Descrizione requisito
  - Lista di task per ciascun requisito, ovvero ciò che lo sviluppatore dovrà fare
  - Note

Story Card



Customer Story and Task Card — BIW Development \ COLA

DATE: 3/19/98 — TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST

STORY NUMBER: 1275 — PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___ — RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:
SPLIT COLA: When the COLA rate chgs. in the middle of the BIW Pay Period we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur "automatically based on system design.

NOTES:
For the OT, we will run a mainframe program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEEntExcessCOLA BIN.

Date	Status	To Do	Comments

Story card

Titolo progetto	Autore	Rigo piano progetto	Priorità
Gestione Immagini	Jim	1	1

Storia

Mostrare le immagini (jpg, png) presenti su una cartella del file system su una griglia di 10x6 immagini, indipendentemente dalla risoluzione dello schermo.

Titolo progetto	Autore	Rigo piano progetto	Priorità
Gestione Immagini	Jim	2	1

Storia

Mostrare le immagini scalate (ridimensionate) rispettando le proporzioni iniziali delle immagini.

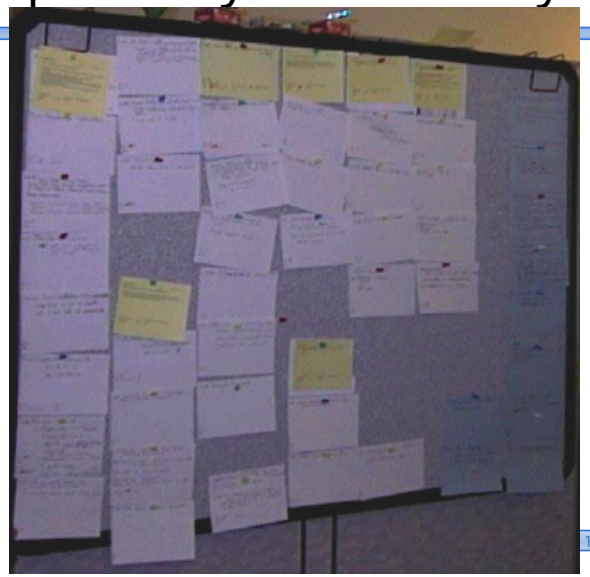
Titolo progetto	Autore	Rigo piano progetto	Priorità
Gestione Immagini	Jim	3	1

Storia

L'utente potrà selezionare immagini digitando lettere all'interno di una casella di testo.
Le immagini selezionate saranno quelle i cui nomi di file contengono il testo inserito.

Stimatore	Tempo stimato	Data	Sviluppatori	Tempo impiegato
Jack	2 giorni	15-03-2011		

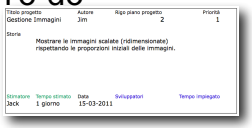
Board per Story Card = Story Board



Story Board

Prima settimana

To do

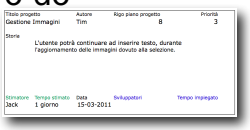


Seconda settimana

To do

Terza settimana

To do



Done

Gioco di pianificazione

- Gli utenti (clienti) scrivono le storie (sono i requisiti)
- Gli sviluppatori stimano il tempo per lo sviluppo di ciascuna storia
  - Se le storie sono troppo complesse da stimare, ritornare dal cliente e far dividere le storie
- Gli utenti dividono, fondono e assegnano priorità alle storie
  - Riempiono 3 settimane scegliendo le storie
  - Non preoccuparsi delle dipendenze
- Gli addetti al business prendono decisioni su
  - Date per le release, contesto, priorità dei task
- Pianificare l'intera release (grossolanamente) e la nuova iterazione
  - Non pianificare troppo in avanti
- Per l'attuale release, gli sviluppatori:
  - Dividono ciascuna storia in task, stimano i task, ciascuno si impegna per realizzare un task
  - Vengono svolti prima i task più rischiosi

Piccole release

- Rendere ogni release il più piccola possibile
  - Tempo di sviluppo della release 2 settimane
- Effettuare un design semplice e sufficiente per la release corrente
- Piccole release forniscono agli sviluppatori
  - Feedback rapidamente
  - Un senso di: "ho ottenuto qualcosa di valido"
  - Rischio ridotto
  - La fiducia del cliente
  - Possibilità di fare aggiustamenti per requisiti che cambiano

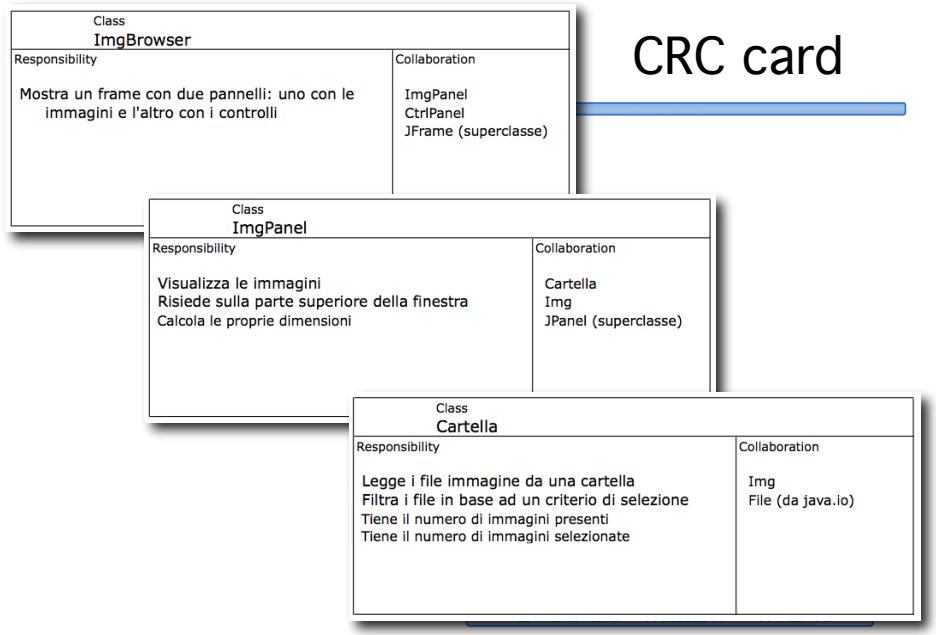
Metafora

- Guidare il progetto con una singola metafora
  - Es.: "La UI è un desktop"
  - Deve rappresentare l'architettura
    - Rende le discussioni più semplici
  - Il cliente deve essere a suo agio con essa

Design Semplice

- Il giusto design per il software si ha quando
  - Passa i test
  - Non ha parti duplicate
  - Esprime ciascuna intenzione importante per i programmatori
  - Ha il numero più piccolo di classi e metodi
- Non preoccuparsi di dover apportare cambiamenti dopo
  - Fare la cosa più semplice che può funzionare
  - Paga quanto usi
- Usare le CRC card (Class Responsibility Collaboration) per documentare il design
  - Permettono di ragionare meglio in termini di oggetti
  - Contribuiscono a fornire una visione complessiva del sistema

CRC card



Testing

- Si testa tutto ciò che potenzialmente può andar male, per tutto il tempo
  - Si eseguono i test più volte al giorno, non appena è stato prodotto del nuovo codice
- I test sono la specifica dei requisiti!
  - Una specifica in formato eseguibile!
- Due tipi di test
  - Test funzionali
  - Unit test

Test

- Test funzionali
  - Scritti dall'utente (punto di vista dell'utente)
  - Effettuati da: utenti, sviluppatori e team di testing
  - Automatizzati
  - Eseguiti almeno giornalmente
  - Sono una parte della specifica dei requisiti, quindi documentano i requisiti
- Unit test
  - Scritti dagli sviluppatori (punto di vista del programmatore)
  - Scritti prima della codifica (TDD, Test Driven Development) ed anche dopo la codifica
  - Supportano design, codifica, refactoring e qualità

Pair Programming

- Programmatori esperti e motivati
- Ruolo di uno dei partner
  - Usa il mouse e la tastiera
  - Pensa al miglior modo di implementare il metodo
- Ruolo dell'altro
  - L'approccio funzionerà?
  - Pensa ai test
  - Potrebbe essere fatto più semplicemente?
- Scambio dei partner
- Pair programming aiuta la disciplina, sparge la conoscenza sul sistema

Possesso del codice collettivo

- Chiunque può aggiungere qualunque codice su qualunque parte del sistema
- Unit test proteggono le funzionalità del sistema
- Chiunque trova un problema lo risolve
- Ciascuno è responsabile per l'intero sistema

Integrazione continua

- Integrazione del codice testato ogni poche ore (max un giorno)
- Tutti gli unit test devono essere superati
- Se un test fallisce la coppia che ha prodotto il codice deve ripararlo
- Se non può ripararlo, buttare il codice e ricominciare

40 ore a settimana

- Se per te non è possibile fare il lavoro in 40 ore, allora hai troppo lavoro
- 40 ore a settimana ti lasciano "fresco" per risolvere i problemi
- Previene l'inserimento di errori difficili da trovare
- Pianificazioni frequenti evitano a ciascuno di avere troppo lavoro
- Ore extra di lavoro è sintomo di un problema serio

Cliente sul sito

- Scrive i test funzionali
- Stabilisce priorità e fornisce il contesto per le decisioni dei programmatori
- Risponde alle domande
- Porta avanti il suo proprio lavoro

- Se non puoi avere il cliente sul sito, forse il progetto non è così importante?

Standard di codifica

- Costruzioni complicate (per il design) non sono permesse
  - Mantenere le cose semplici
- Il codice appare uniforme
  - Più facile da leggere
- Usare tutti la stessa convenzione, così non si ha necessità di riformattare il codice, sapere come usare
  - Spazi e Tab per indentazione
  - Posizione parentesi graffe
  - Scelta di nomi classi, metodi, attributi
  - Posizione commenti

Vedere convenzioni per linguaggio Java!

http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

Refactoring

- Refactoring significa migliorare la struttura del codice senza influenzarne il comportamento
- Fatto in piccoli passi
- Supportato dagli unit test, design semplice e pair programming
- Puntare a codice senza ripetizioni
- Refactoring fatto in coppia dà più coraggio

In breve

- XP focalizza sul codice
  - Fare solo le cose che sveltiscono la produzione del codice
  - Codifica e test
- XP si orienta sulla gente
  - La conoscenza del sistema è trasferita attraverso la comunicazione tra la gente
- XP è leggero
  - Rimuovere i costi aggiuntivi
  - Creare prodotti di qualità tramite test rigorosi
- I princìpi di XP non sono nuovi