Agile

mercoledì, 1 luglio 2015, 3:05 p.

Prof. Tramontano docente Federico II ingegneria del software

Sviluppo Agile: metaprocesso

- Molti progetti software falliscono
- Sì parte dagli anni 2000
- Millennium Bug
- Se il processo è ad alta qualità il software è ad alta qualità
- Ciò ha portato ad una tale rigidità che si ripercuote sul software stesso quindi spessissimo fallisce.
- Lo standard mi permette di stimare bene i costi e l'affidabilità ma è pericoloso
- Waterfall model
- Spesso siamo costretti a mettere una pezza (challenging)
- È l'opposto dell'agile
- È la più rischiosa
- Va bene per le grandi aziende

Ciclo a cascata con validazione continua

- Verificare: il software deve dare il risultato corretto
- Validare: deve fare quello richiesto dal cliente
- Se ci sono problemi torniamo indietro di una fase

RUP

- È un processo molto preciso
- Minimizza i rischi di un fallimento
- È iterativo: rilasciamo spesso qualcosa da mostrare al cliente
- Iper burocratizzazione

Metodi Agili

- Non garantiscono la qualità
- Va bene per piccoli software
- Ad esempio software innovativi per i quali no conosciamo bene le caratteristiche
- Sviluppo di app

- Manifesto Agile
- Kent Beck
- Prima gli individui poi i tool.
- Non ci serve troppa standardizzazione
- Abbiamo il problema di non avere traccia di quanto DETTO tra le controparti. Non avendo una relazione approfondita
- Riduce i rischi di validazione
- Il software viene prima della documentazione
- Il problema è che la documentazione è importante per la manutenzione futura.

- Prima la collaborazione con il cliente che i contratti. Il cliente deve lavorare insieme agli sviluppatori: il prezzo si stabilisce a prodotto finito.
- Deve seguire i cambiamenti e no u piano preciso
- È inutile fare pianificazioni a lunghissimo tempo perché saremo sempre smentiti. Quindi Agile dice:non pianifichiamo proprio.
- Qualsiasi cosa facciamo dobbiamo valutare se l'abbiamo fatta bene
- Rapida, incrementale consegna del software
- Processo agile
- Story Card: ci scriviamo gli scenari. Deve essere piccolo. Non vogliamo affrontare troppe cose in un colpo solo. Cerchiamo di affrontare il problema un po alla volta.
- Mettiamo in salvo le piccole cose
- Ho prodotti con scarsa qualità
- Massimo orizzonte temporale: 2 settimane-3 mesi
- Funziona molto bene in ambienti non distribuiti. Idealmente sono affiatati e lavorano nella stessa stanza.
- Gruppi piccoli e tempo limitato
- Ci sono molti che hanno cercato di sviluppare i principi Agili

Principi:

- Il cliente partecipa attivamente
- Definisce le priorità perché metto in conto di non riuscire a finire il progetto
- Consegna incrementale
- Lo devo evitare se devo fare software di altissima qualità
- Le persone devono fare i processi
- Devo limitate al massimo le imposizioni
- Accettiamo i cambiamenti
- DOBBIAMO ESSERE SEMPLICI: il software lo devono capire tutti e deve essere semplice da modificare
- Per Agile il software DEVE FUNZIONARE, per gli schemi classici deve essere QUALITATIVAMENTE VALIDO

Agilità e Modellazione

- Modellazione: è un gradino oltre la progettazione. Serve ad individuare gli attori e stiamo già pensando in dettaglio al problema.
- Modello dei casi d'uso
- Modello concettuale
- Nel modello NON DEVE ESSERCI LA SOLUZIONE
- Nell'agro c'è il rischio di non fare la modellazione perché non ci interessa conoscere interamente il problema
- Spesso il cliente non riesce a spiegarci bene qual è il problema

Processo:

- Identifichiamo le funzionalità che vogliamo rilasciare
- L'interazione va da 2 a 13 settimane: sviluppo e testo insieme al cliente
- Quando finisce l'interazione lo diamo al cliente finale e lo supportiamo
- Nel caso in cui c'è bisogno di fare delle modifiche bisogna capire se interrompere il progetto o prendere una risorsa e dedicata alle modifiche
- Quando definisco un requisito devo anche definire le metodologie di verifica
- Man mano che si sviluppa si scrivono i test

- Conviene fare test automatico:

1. Scrittura
2. Esecuzione
3. Risultati

- Per gestire al meglio i test utilizzo Junit (un frame con classi e metodi)
- Nel sistema agile il testing dovrebbe essere più semplice
- Con Junit posso rendere automatico anche la gestione dei risultati
- Dobbiamo fare test continui non accumulare troppo da testare. NON SI SALTANO I TEST
- Sarebbe opportuno che i test li faccia chi non ha creato la funzione
- Nell'agile si propone di alternare sviluppatori e testing
- TDD: prima scriviamo il test poi facciamo il programma
- Il test di sistema lo faccio da solo
- Il test di accettazione lo faccio con il cliente
- Ogni volta che aggiungiamo un pezzo rifacciamo tutti i test. Se sono automatici non abbiamo problemi. Magari li facciamo fare di notte.
- Junit è punto fondante della metodologia Agile

VANTAGGI

- So quando posso consegnare il pezzetto
- Con pagamenti giorno per giorno ho una migliore monetizzazione
- Si evolve working progress
- Testiamo in continuazione
- Il cliente è soddisfatto

EXTREME PROGRAMMING (XP)

- È la prima metodologia 1999
- Concetto di storie: il cliente mi dice cosa vuole
- Creo degli scenari (requisiti) non ambigui, verificabili, coerenti e completi le ultime due sono spesso trascurate
- Se modifico qualcosa se ne creava una nuova versione di questo qualcosa. Inoltre faccio il test di queste versioni (controllo di versione)
- Si fonda su strumenti ben precisi come Junit
- Ho rilasci molto frequenti. Massimo un mese

PRATICHE XP

1. Pianificazione: la facciamo tutti insieme sotto forma di GAME. Deve durare poco.
2. Small releases: piccoli rilasci quindi riesco meglio a gestire tutto
3. Metaphor: tutti devono sapere tutto dei progetti. Anche i nomi delle variabili e dei metodi devono essere comprensibili a tutti. Magari con metafore.
4. Simple Design: la semplicità porta facilità di gestione ma sono di bassa qualità. (Design Pattern)
5. Testing: mai evitare ed automatizzare
6. Refactoring: modifiche al codice del programma per migliorare la qualità. Questo perché il codice di XP è già di scarsa qualità. Ce lo dobbiamo imporre
7. Pair programming: programmazione a coppia. Si controllano a vicenda.
8. Collective ownership: tutti devono sapere tutto
9. SECONDA GIORNATA. Integrazione continua. Ad ogni nuova aggiunta è sempre presentabile

10. Settimana di 40 ore: la stanchezza provoca errori. Dobbiamo stare bene. Bisogna avere meno paure per le scadenze anche perché ho sempre qualcosa che posso consegnare
11. Lavoriamo con il cliente.
12. Si deve scrivere con lo stesso stile

- Pianifico, disegno, scrivo il codice faccio il test ed il refactoring. Poi consegno e vado allo step successo
- I requisiti li faccio con le story Card: semplici foglietti dove scrivo cosa fare
- Punto debole di XP sono le modifiche. Il software non è longevo. Non ho documentazione.
- In teoria dopo il refactoring non dovrei fare I test. Ma di norma si fanno.
- Di norma il test NON VA SUBAPPALTATO. Ci sarebbero problemi di interazione e non abbiamo la possibilità di far loro capire cosa vogliamo

- I test vanno fatti ma non bisogna impegnare troppe risorse altrimenti è deleterio
- Vedi LA COPERTURA DEL CODICE
- L'ideale sarebbe fare i test contemporaneamente alla scrittura. Nel TDD il test va fatto addirittura prima.
- Con il pair programming si può anche pensare che uno scrive il codice ed uno scrive i test in contemporanea
- Ci possono essere problemi con i clienti: poca disponibilità o troppo coinvolgimento
- Il software che nasce DALL'Xp è sostanzialmente un software vecchio
- La validazione la faccio con le release brevi visto che il cliente può immediatamente convalidare ciò che abbiamo fatto

TDD

- È un processo Agile
- Sviluppo guidato dai test. Devo TESTARE tutto
- L'idea è quella di supporre che il codice non funziona
- Prima scriviamo il test poi scriviamo il codice
- Lo svantaggio è che il test ci impone come deve essere il programma
- Potrei risolvere questo svantaggio con le Interface che definiscono le classi astratte
- TEST - CODIFICA - PROGRAMMO
- L'unico scopo è quello di far funzionare il programma
- Tra i vantaggi si hanno quello di avere codice funzionale e pulito
- Non sempre il codice ha qualità

Agile verso il software di grandi dimensioni

- Di norma non è utile applicare Agile prr grossi progetti
- Va bene per progetti di piccoli e medie dimensioni
- Turnover quasi nullo nello sviluppo.
- In aziende mono progetto
- Eclettismo: condivisione del materiale
- Scaling up: scalo l'agile verso qualcosa di più grosso
- Scaling out Da un'azienda tradizionale creo un isola agile
- Posso anche rendere un progetto agile IBRIDO tenendo conto solo di alcuni paradigmi.
- Spesso i metodi Agili vanno contro i principi del valore legale perché lavoro in modo informale
- Ci potrebbero anche essere problemi di certificazione
- Ottimizzo lo sviluppo ma trascuro la manutenzione
- Con il metodo AGILE non realizzo software longevi
- Gli Skill devono essere molto alti ed eclettici: tutti fanno tutti e sono tutti allo stesso livello.

- Servono IDE!
- Vedi lo strumento DOS