THE AGILE WAY

Uncovering better ways of developing software

Presentazione a cura di: ***Marco Abis***
abis@agilemovement.it
Italian Agile Movement Founder

Italian Agile Movement

- Nasce alla fine del 2001 (sito nel 2002)
- E' User Group ufficiale per l'Italia dell'Agile Alliance
  172 membri (pochi ma buoni)
- Fornisce notizie ed aggiornamenti
- Coordina, propone e supporta attività di studio, applicazione e ricerca
- Fulcro, con altri gruppi Europei, della prossima Agile Alliance Europe
- Trasversale ad altre iniziative (UGIdotNET)
- Versione Inglese

Agile Alliance

IAM Italian Agile Movement

Agile Software Development

In uno studio su 200 progetti software Michael Math della QSM ha comunicato che i ricercatori non sono stati in grado di trovare i piani originali di oltre la metà dei progetti;

- Adeguarsi ai piani non era più l'obiettivo primario. Lo era invece soddisfare il cliente al momento della consegna piuttosto che allo start del progetto;
- E' necessario operare continui cambiamenti ai piani originali.

Cambiare costa?

Accettiamo la teoria della differenza di costo nelle diverse fasi del ciclo di sviluppo delineata da Barry Boehm: il costo di un cambiamento cresce con l'avanzare del progetto.

Per mantenere un vantaggio competitivo, rispettare le condizioni di business e soddisfare i clienti la domanda non è più come evitare i cambiamenti bensì come gestire al meglio l'inevitabile.

La risposta Agile

La strategia è ridurre il costo dei cambiamenti per tutto il ciclo di sviluppo del software. Ad esempio eXtreme Programming richiede al team di:

- Produrre consegne funzionanti in settimane per ottenere un rapido feedback
- Usare soluzioni semplici cosi che costi meno cambiarle e sia più facile farlo
- Migliorare la qualità del design continuamente
- Testare continuamente per trovare i difetti il prima possibile

Design e Qualità

L'Agile Software Development spinge per la qualità nel design e spesso viene additato come ad-hoc coding perchè il design è sottoposto a continue modifiche (migliorative), in piccoli passi invece che tutto in una volta up-front.

Ogni metodo agile affronta la qualità in modi diversi ma complementari e spesso equivalenti:

DSDM → serie di prototipi per attaccare aree instabili o sconosciute come nuove tecnologie, nuove regole di business, nuove interfaccie utente, etc, etc

Scrum → intensi daily meeting di 15 minuti e iteration review ogni 30 giorni

Approcci Agili

- eXtreme Programming
- Scrum
- Adaptive Software Development
- Crystal family
- Dynamic System Development Method
- Lean Software Development
- Feature Driven Development
- Pratiche minori (Agile Modeling, Pragmatic Programmers)

Lightweight → Agile

11-13 Febbraio 2001 – meeting Snowbird, Utah

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler
James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick
Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Agile Alliance

IAM Italian Agile Movement

Manifesto

Stiamo portando alla luce modi migliori di sviluppare software facendolo in prima persona ed aiutando altri a farlo.

Con questo lavoro siamo giunti a questi valori:

Persone ed interazione più che processi e tools;

Software che funziona più che documentazione esaustiva;

Collaborazione con il cliente più che negoziazione contrattuale;

Rispondere al cambiamento più che il seguire un piano prestabilito;

Ecco perchè mentre teniamo conto delle voci sulla destra teniamo conto di quelle sulla sinistra molto di più.

Agile Alliance

IAM Italian Agile Movement

Principi

La priorità più alta è soddisfare il cliente attraverso il rilascio "precoce" e continuo di software funzionante e valutabile

Il principio del customer value è uno di quelli "più facile a dirsi che a farsi". Secondo l'approccio tradizionale al project management:

piano di progetto = successo del progetto = valore dimostrato

La volatilità associata ai progetti di oggi richiede che il customer value venga rivalutato frequentemente perchè il raggiungimento dei piani di progetto originali (fatti mesi se non anni prima) non implica automaticamente (quasi mai) il raggiungimento del successo oggi.

Principi

Accogliere il cambiamento dei requisiti, anche tardi nello sviluppo. I processi agili sfruttano il cambiamento per il vantaggio competitivo del cliente

La turbolenza, sia nel business che nella tecnologia, causa cambiamenti che possono essere considerati pericoli da cui proteggersi o opportunità da cogliere.

Invece che resistere al cambiamento gli approcci agili si sforzano di adattarsi il più facilmente ed efficentemente possibile pur tenendo presenti le conseguenze. Nonostante tutti concordino che il feedback è importante spesso si ignora il fatto che il risultato del feedback è il cambiamento.

Agile Alliance

IAM Italian Agile Movement

Principi

Consegnare frequentemente software funzionante, ad intervalli che vanno da qualche settimana a qualche mese, con preferenza per i periodi più corti

Consegnare è ben diverso da rilasciare.

Le persone di business possono aver validi motivi per non mettere in produzione codice ogni tot settimane ma questo non impedisce di avere un rapido ciclo di rilasci interni che permettono a tutti di valutare il progetto ed imparare come farlo evolvere.

Principi

Le persone di business e gli sviluppatori lavorano insieme quotidianamente per tutta la durata del progetto

Comprare un programma è diverso da comprare un'automobile.

Per questo gli approcci agili all'inizio non si aspettano tanti requisiti dettagliati da sottoscrivere prima di cominciare un progetto. Si preferisce un insieme di requisiti di alto livello sottoposti a continui cambiamenti. Chiaramente questo non è sufficiente per sviluppare software quindi si colma il vuoto con interazioni frequenti tra le persone di business e gli sviluppatori. Frequenti è da intendere come "quotidiani" e non mensili.

Principi

Costruire progetti attorno a persone motivate. Dare loro l'ambiente ed il supporto di cui necessitano e, soprattutto, "dargli fiducia"

Non importa quali tools, tecnologie e processi vengano impiegati, sono le persone che alla fine fanno la differenza tra il successo ed il fallimento.

La fiducia spesso è la cosa più difficile da concedere ma per avere successo è necessario che le decisioni vengano prese da chi conosce meglio la materia volta per volta quindi i manager devono fidarsi del proprio staff e lasciar decidere le persone che sono pagate per far funzionare le cose.

Agile Alliance

IAM Italian Agile Movement

Principi

Il modo più efficiente ed efficace per comunicare al e con il team di sviluppo è la conversazione face-to-face

Il vero problema non è la documentazione ma la comprensione!

"Tacit knowledge cannot be transferred by getting it out of people's heads and onto paper," scrive Nancy Dixon in Common Knowledge (Harvard Business School Press, 2000). "Tacit knowledge can be transferred by moving the people who have the knowledge around. The reason is that tacit knowledge is not only the facts but the relationships among the facts—that is, how people might combine certain facts to deal with a specific situation."

Agile Alliance

IAM Italian Agile Movement

Principi

Il software funzionante è la primaria misura del progresso

Il software funzionante è la misura del progresso perchè non esiste alcun altro modo di catturare le sottigliezze dei requisiti: i documenti ed i diagrammi sono troppo astratti per permettere all'utente di provare e rendersi conto dello stato del progetto.

Principi

I processi Agili promuovono lo "sviluppo sostenibile". Sponsor, sviluppatori e utenti dovrebbero essere in grado di mantenere un ritmo costante indefinitivamente

Il nostro settore è caratterizzato da lunghe notti e fine settimana passati a cercare di rimuovere gli errori di una pianificazione insensibile ma quelle ore, ironicamente, non portano ad un aumento della produttività. Spesso si passa il giorno seguente a rimuovere gli errori introdotti durante la nottata.

Principi

Attenzione continua all'eccellenza tecnica e al buon design aumentano l'agilità

Nonostante quello che dicono i detrattori (che non hanno mai neanche provato sul campo per farsi un'idea) gli approcci agili enfatizzano la qualità del design.

I differentei approcci agili sposano differenti stili di desing:

- FDD → step all'inizio di ogni iterazione (solitamente con UML)
- XP → refactoring per permettere al design di evolvere

In ogni caso il design viene migliorato continuamente per tutta la durata del progetto.

Principi

La semplicità - l'arte di massimizzare il lavoro non fatto - è essenziale

E' più facile aggiungere qualcosa ad un processo che si rivela troppo semplice per le correnti necessità che togliere qualcosa da un processo che è troppo complicato. Forte minimalismo per includere solo ciò che tutti necessitano e facilitare l'adattamento.

Semplicità e chiarezza generano comportamenti complessi ed intelligenti - Regole complesse generano comportamento semplici e stupidi.

Nessuna metodologia può indirizzare tutta la complessità dei moderni progetti software: poche e semplici regole incoraggiano la creatività

Principi

Le migliori architetture, requisiti e design emergono da team self-organizzati

Design (architetture, requisiti) migliori emergono dallo sviluppo iterativo e dall'utilizzo piuttosto che dai piani iniziali.

Le proprietà emergenti sono generate meglio da self-organizing team in cui le interazioni sono molte e le regole di processo poche.

"*Emergere*" come proprietà chiave dei complex systems, rozzamente tradotto in innovazione e creatività nelle organizzazioni umane.

Agile Alliance

IAM Italian Agile Movement

Principi

Ad intervalli regolari riflette su come diventare più efficace affina ed adatta il proprio comportamento di conseguenza.

Non esistono silver-bullets perchè non è possibile un processo che sia adatto a tutte le situazioni quindi ogni team agile deve affinare e riflettere sul proprio percorso migliorando costantemente le pratiche adattandole alle circostanze.

Scrum@XP

ASD@XP

DSDM@XP

Lean@XP

eXtreme Programming

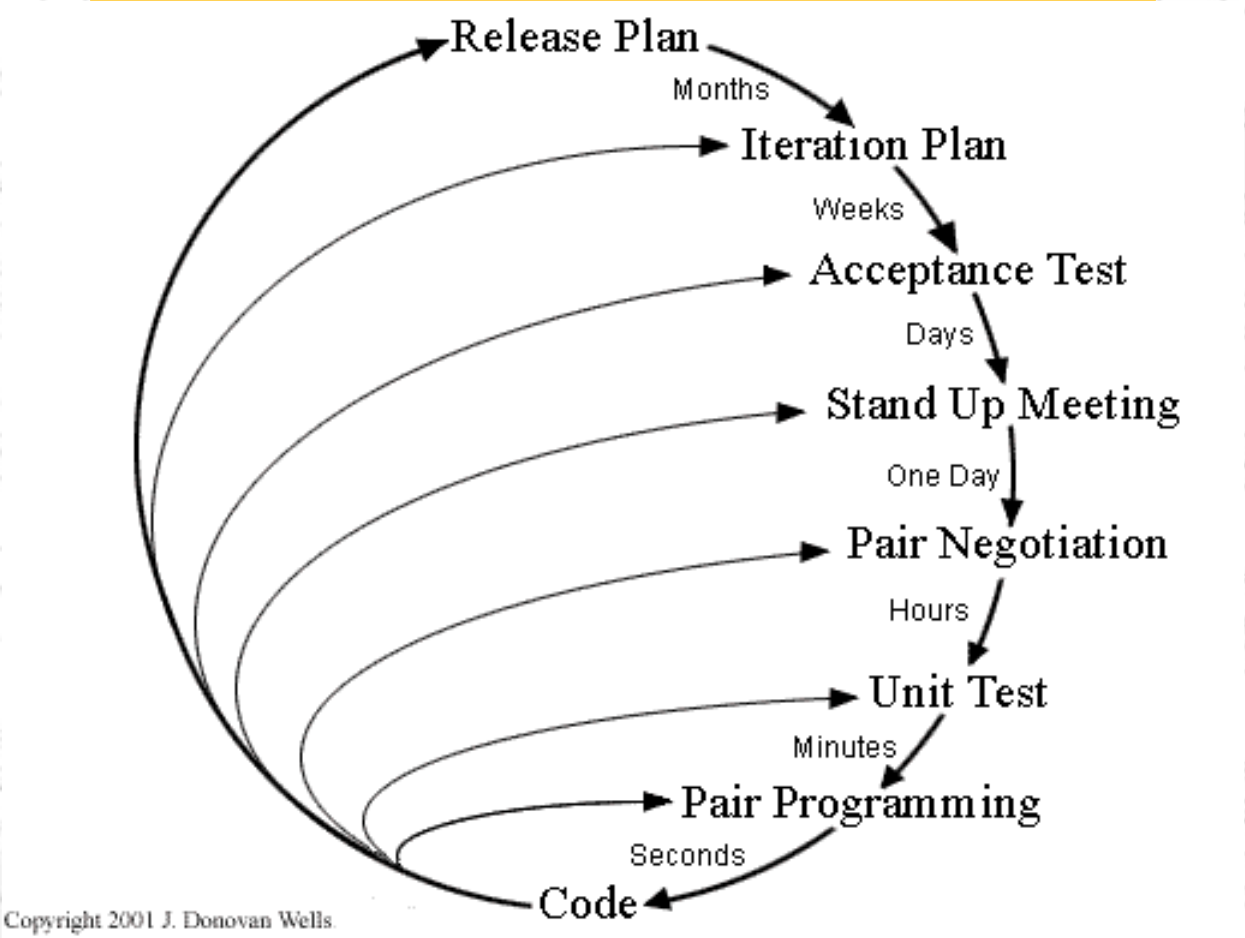
XP è il più famoso anche se non il più "vecchio" tra i metodi agili per diversi motivi:

- Pratiche ingegneristiche di sviluppo molto più interessanti e stimolanti delle sole pratiche di management
- Tantissimi libri, articoli, conferenze, sostenitori e detrattori
- Alcune pratiche "forti", su tutte il Pair Programming
- Il nome

Agile Alliance

IAM Italian Agile Movement

Planning/Feedback Loops



Le 12 pratiche di XP

Planning game
Small releases
Metaphor
Simple design
Refactoring
Testing
Collective Ownership
Pair programming
Continuous integration
40-hour week
On-site customer
Coding standard

Planning game

L'enfasi è sul guidare il progetto piuttosto che sul prevedere esattamente cosa sarà necessario e quanto tempo si impiegherà per l'intero progetto. Due pratiche:

Release planning: il Customer presenta le features desiderate, gli sviluppatori stimano la loro difficoltà ad alto livello. Con le stime in mano il Customer ordina le feature per priorità - da quelle che gli portano maggior valore a quelle di minor valore.

Iteration planning: determina la direzione per la prossima iterazione. I team XP sviluppano il software in iterazioni di due settimane consegnando alla fine software funzionante e utile al customer E' il Customer ad indicare le feature che vorrebbe per le prossime due settimane. Gli sviluppatori dividono le feature indicatein tasks, stimano i loro costi (dettaglio maggiore rispetto al Release planning).

Small Releases

Il team rilascia versioni funzionanti e testate del software consegnando valore di business scelto dal Customer, ogni iterazione. Il Customer può usare questo software per qualsiasi scopo: sia di valutazione sia di rilascio agli utenti finali. La cosa più importante è che il software è visibile, tangibile.

Il team rilascia giornalmente (daily build) il software anche solo ad uso interno

Metaphor

Viene sviluppata una visione comune a tutto il team di come il programma funziona e viene chiamata "metafora". Nel migliore dei casi la metafora è una semplice frase evocativa che descrive il funzionamento del programma. Come ad esempio:

"questo programma funziona come un'alveare di api, che escono in cerca di polline per portarlo all'alveare"

per descrivere un sistema di information retrieval agent-based.

Simple design

In XP i team partono con un desing il più semplice possibile e attraverso il testing ed il refactoring lo mantengono semplice. Il design viene mantenuto esattamente per come è utile e sufficiente a soddisfare le funzionalità correnti del sistema.

Refactoring

Il processo di refactoring si focalizza sulla rimozione di duplicazioni (segno sicuro di un cattivo design) e sul miglioramento della coesione del codice, mentre riduce l'accoppiamento. Un'alta coesione ed un basso accoppiamento è riconosciuto come segno caratteristico di codice ben disegnato per almeno 30 giorni. Il risultato è che con XP si parte con un buon e semplice design e lo si mantiene per semplice e buono per tutto lo sviluppo.

Il refactoring è pesantemente supportato dalle pratiche di testing per assicurarsi che mentre evolve nulla venga "rotto".

Le pratiche di XP si supportano l'un l'altra andano ognuna a coprire possibili lacune lasciate da altre.

Testing

Customer Tests: nel momento in cui il Customer presenta le features desiderate definisce anche uno o più test automatici di accettazione che dimostrino il corretto funzionamento della feature. Il team costruisce questi test e li usa per provare a se stesso e al Customer che la feature è implementata correttamente. L'automazione è importante perchè quando la pressione sale i test manuali vengono sistematicamente saltati, come se si spegnesse la luce nel momento in cui arriva il buio.

Test-Driven Development: il feedback è tutto e nello sviluppo software un buon feedback richiede un buon testing. I team XP praticano il Test Driven Development scrivendo i test ancora prima di scrivere il codice da testare e quindi sviluppando il codice fino al punto di non far fallire il test precedentemente scritto.

Collective Ownership

Ogni coppia di sviluppatori può (e deve) migliorare il codice, in qualsiasi momento. Tutto il codice beneficia dell'attenzione di molte persone aumentando la qualità e riducendone i difetti.

Quando il codice è di "proprietà" di una sola persona nuove funzionalità vengono spesso messe nel posto sbagliato dagli sviluppatori che scoprono di dover modificare il codice di altri. Il "proprietario" è troppo occupato per farlo e cosi il codice perde in qualità.

La Collective ownership può essere un problema se le persone lavorano cecamente sul codice che non comprendono. XP evita questo con due tecniche chiave: i test trovano gli errori e lo sviluppo a coppie permette di avvicinare un estraneo al codice lavorando con l'esperto di turno. In aggiunta questa pratica diffonde la conoscenza attraverso il team.

Pair Programming

Tutto il software viene sviluppato da coppie di programmatori che siedono fianco a fianco davanti alla stessa macchina. Questa pratica assicura che tutto il codice venga revisionato almeno da un altro sviluppatore e risulta in un miglior design, miglior testing e miglior codice in generale.

La più grande critica al pair programming è la credenza che sia inefficiente e sconveniente far fare a due sviluppatori il lavoro di uno ma è vero il contrario. Ricerche sul pair programming mostrano che lavorare a coppie produce codice migliore oltre che nello stesso tempo richiesto dal lavoro individuale. Due teste sono veramente meglio di una.

Alcuni programmatori criticano il pair programming senza averlo mai neanche provato. Richiede certamente un pò di pratica (qualche settimana) per ottenere risultati. Il 90% degli sviluppatori che hanno imparato a lavorare in coppia lo preferiscono.

Continuous Integration

Il sistema viene mantenuto completamente integrato per tutto il tempo, dal primo all'ultimo giorno. Non solo daily builds ma anche molteplici build al giorno.

I vantaggi sono evidenti per chiunque abbia lavorato a progetti non banali dove l'integrazione avveniva, nel migliore dei casi, settimanalmente portando al famoso "integration hell" in cui tutto smette di funzionare e nessuno sa perchè.

40-hour week

I team lavorano per il lungo periodo, lavorano duro e ad un ritmo sostenibile indefinitamente. Questo significa che fanno gli straordinari quando questo porta un reale vantaggio e che normalmente lavorano in un modo che massimizzi la produttività settimanalmente. E' ormai comunemente riconosciuto che le notturne non sono ne quantitativamente ne qualitativamente produttive. I tema XP sono li per vincere, non per morire.

On-site customer

Il cliente deve far parte effettivamente del team perchè nel successo o nel fallimento del progetto ha un ruolo fondamentale tanto quanto gli sviluppatori.

Per questo XP sponsorizza la presenta full-time nel team di un rappresentate significativo del cliente. L'unico che possa guidare lo sviluppo verso quello che più crea valore per il customer al momento.

Coding standards

I team XP seguono i comuni standard di codifica in modo che tutto il codice del sistema appaia come se fosse scritto da una singola persona (molto competente). Le specifiche degli standard non sono importanti: ciò che è importante è che tutto il codice sembri familiare a supporto della collective ownership.

Riferimenti

Italian Agile Movement: http://agilemovement.it
Agile Alliance: http://agilealliance.com (Articles and Roadmap)
eXtreme Programming: http://www.extremeprogramming.org
Scrum: http://www.scrumalliance.org
ASD: http://www.jimhighsmith.com/learn.html
FDD: http://www.featuredrivendevelopment.com
DSDM: http://www.dsdm.com
Lean SD: http://www.leanprogramming.com
Crystal: http://alistair.cockburn.us/crystal/crystal.html

Libri principali

Agile Software Development Ecosystems
Jim Highsmith - Addison-Wesley – ISBN 0201760436

Extreme Programming Explained: Embrace Change
Kent Beck - Addison-Wesley - ISBN: 0201616416

Agile Software Development with SCRUM
Ken Schwaber, Mike Beedle - Prentice Hall - ISBN: 0130676349

Adaptive Software Development: A Collaborative Approach to Managing Complex Systems
Jim Highsmith, Ken Orr - Dorset House - ISBN: 0932633404

Dynamic Systems Development Method : The Method in Practice
Jennifer Stapleton - Addison-Wesley - ISBN: 0201178893

Lean Software Development: An Agile Toolkit for Software Development Managers
Mary & Tom Poppendieck - Addison-Wesley - ISBN: 0321150783

A Practical Guide to Feature-Driven Development
Stephen R. Palmer, John M. Felsing - Prentice Hall - ISBN: 0130676152

Agile Alliance

IAM Italian Agile Movement

GRAZIE

Domande?

Presentazione a cura di: ***Marco Abis***
abis@agilemovement.it
Italian Agile Movement Founder

Agile Alliance

IAM Italian Agile Movement