# Processo e Sviluppo del Software

UniShare

Davide Cozzi
@dlcgold

# Indice

# Capitolo 1

# Introduzione

**Questi appunti sono presi a lezione. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link:** `https://github.com/dlcgold/Appunti`.
**Le immagini presenti in questi appunti sono tratte dalle slides del corso e tutti i diritti delle stesse sono da destinarsi ai docenti del corso stesso**.

# Capitolo 2

# Metodi Agili

I *metodi agili* sono stati definiti per rispondere all'esigenza di dover affrontare lo sviluppo di software in continuo cambiamento. Durante lo sviluppo si hanno vari passaggi:

- comprensione dei prerequisiti
- scoperta di nuovi requisiti o cambiamento dei vecchi

Questa situazione rendeva difficile lo sviluppo secondo il vecchio metodo *waterfall* (portando al fallimento di diversi progetti).
I *metodi agili* ammettono che i requisiti cambino in modo "naturale" durante il processo di sviluppo software e per questo assumono un modello di processo *circolare*, con iterazioni della durata di un paio di settimane (figura 2.1). Potenzialmente dopo un'iterazione si può arrivare ad un prodotto che può essere messo in "produzione". Dopo ogni rilascio si raccolgono *feedback* per poter rivalutare i requisiti e migliorare il progetto.
Si hanno quindi aspetti comuni nei metodi agili e nel loro processo:

- enfasi sul team, sulla sua qualità e sulla sua selezione
- il team è *self organizing*, non essendoci un *manager* ma essendo il team a gestire lo sviluppo, dando importanza ai vari membri del team
- enfasi al pragmatismo, evitando di produrre documenti inutili che sarebbero difficili da mantenere, focalizzandosi su una documentazione efficacie
- enfasi sulla comunicazione diretta e non tramite documenti, tramite meeting e riunioni periodiche

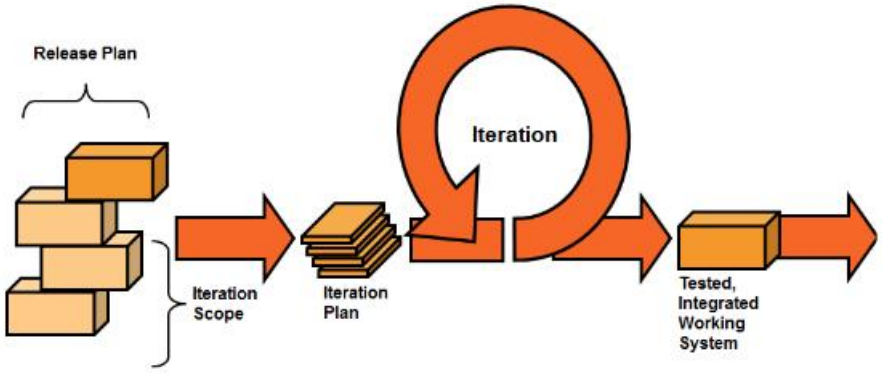


Figura 2.1: Rappresentazione grafica del modello agile. I blocchi a sinistra rappresentano i requisiti (da sviluppare secondo una certa priorità). Lo sviluppo si articola tramite varie iterazioni che porta ogni volta ad aggiungere una parte al prodotto finale (ogni iterazione produce, a partire da un certo requisito, un parte di prodotto di qualità già definitiva, con testing, documentazione etc...)

- enfasi sull'idea che nulla sia definitivo e che non bisogni seguire la perfezione fin da subito ma che, "pezzo per pezzo", ogni step porterà al raggiungimento di una perfezione finale (anche dal punto di vista del design)
- enfasi sul controllo costante della qualità del prodotto, anche tramite pratiche tecniche come quella del *continuous testing*, dove un insieme di test viene eseguita in modo automatico dopo ogni modifica, o anche pratiche di *analisi statica e dinamica* del codice al fine di trovare difetti nello stesso nonché di *refactoring*

*I metodi agili sono molto "elastici" e permettono la facile definizione di nuovi metodi facilmente adattabili al singolo progetto.*

## 2.1 Scrum

Tra i vari *metodi agili* uno dei più famosi è **scrum** (figura 2.2).
In questo caso la parte di sviluppo e iterazione prende il nome di *sprint* e ha una durata variabile tra una e quattro settimane, per avere un rilascio frequente e una veloce raccolta di feedback. I requisiti sono raccolti nel cosiddetto *product backlog*, con priorità basata sulla base delle indicazioni del committente. Ad ogni *sprint* si estrae dal *product backlog* lo *sprint backlog*,

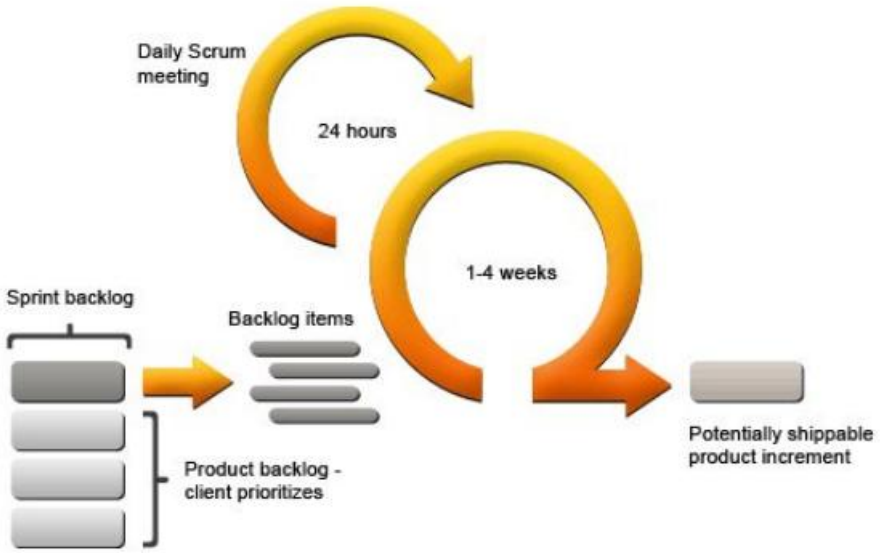


Figura 2.2: Rappresentazione grafica del processo scrum

ovvero il requisito (o i requisiti) da implementare nello *sprint*. Lo *sprint backlog* viene analizzato nel dettaglio producendo i vari *backlog items*, ovvero le singola funzionalità che verranno implementate nello *sprint*. Si ottiene quindi di volta in volta un pezzo di prodotto finale, testato e documentato. Durante le settimane di *sprint* si ha anche un meeting giornaliero utile per mantenere alti i livelli di comunicazione e visibilità dello sviluppo. Durante il meeting ogni dev risponde a tre domande:

1. Cosa è stato fatto dall'ultimo meeting?
2. Cosa farai fino al prossimo meeting?
3. Quali sono le difficoltà incontrate?

l'ultimo punto permette la cooperazione tra team members che sono consci di cosa ciascuno stia facendo.
Durante il processo *scrum* si hanno quindi tre ruoli:

1. il **product owner**, il committente che partecipa tramite feedback e definizione dei requisiti
2. il **team**, che sviluppa
3. lo **scrum master**, che controlla che il processo scrum sia svolto correttamente

Essi collaborano nelle varie fasi:

- product owner e team collaborano nella definzione dei backlog e nella loro selezione ad inizio *sprint*
- durante lo *sprint* lavora solo il team
- nello studio del risultato collaborano tutti coloro che hanno un interesse diretto nel progetto (team, product owner, stakeholders)

*Lo scrum master interagisce in ogni fase, fase che viene comunque guidata tramite meeting*:

- **sprint planning meeting**, ad inizio *sprint*
- **daily scrum meeting**, il meeting giornaliero
- **sprint review meeting**, in uscita dallo *sprint* per lo studio dei risultati
- **sprint retrospective meeting**, in uscita dallo *sprint* per lo studio tra i membri del team di eventuali migliorie al processo e allo sviluppo del prodotto (anche dal punto di vista delle tecniche e delle tecnologie)

## 2.2 Extreme Programming

Un altro tipo di metodo agile è l'*extreme programming* (figura 2.3), ormai poco usato. I requisiti prendono i nomi di *stories*, che vengono descritti come l'utente del sistema, detto *attore*, che cerca di fare qualcosa tramite una "narrazione". Vengono scelti quindi *stories* per la prossima iterazione, dove si hanno testing e revisione continua. Le release di ogni iterazione vengono catalogate per importanza (con anche la solita collezione di feedback). In *extreme programming* si hanno davvero molte pratiche:

- The Planning Game
- Short Releases
- Metaphor
- Simple Design
- Refactoring

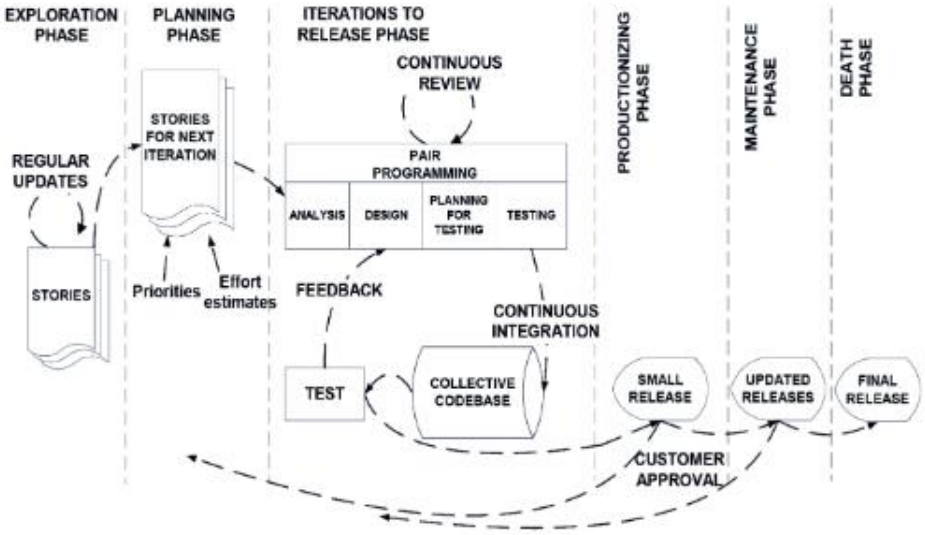


Figura 2.3: Rappresentazione grafica dell'extreme programming

- Test-First Design
- Pair Programming
- Collective Ownership (codice condiviso da tutti senza alcun owner, con tutti in grado di effettuare modifiche)
- Continuous Integration
- 40-Hour Week
- On-Site Customer
- Coding Standard (per avere il collective ownership avendo un solo standard di codifica comune a tutti)
- Open workspace