Riassunto Libro Parte 1

Luiggi

May 30, 2020

1 Processo software e metodologia

1.1 Punti chiave

- **Processo software:** definisce le fasi di attività o cosa va fatto per costruire un sistema software.
- **Metodologia software:** specifica in maniera dettagliata i passi o come vanno svolte le attività di un processo software.
- **Sviluppo software:** ha bisogno di un processo software e una metodologia.

I programmi del mondo reale hanno milioni di linee di codice, rispetto a quelli accademici che ne hanno qualche migliaio. E oltre al numero di linee di codice, i software del mondo reale ha molti altri problemi da affrontare: ciò è chiamato *processo software*. Analizzeremo perchè per lo sviluppo di un sistema software non basta il processo software ma si ha bisogno anche di una *metodologia*.

1.2 Sfide dello sviluppo del software

Ci sono svariati problemi inerenti allo sviluppo di sistemi e progetti software nel mondo reale, per i quali vanno sapute prendere delle decisioni per capire come gestire queste circostanze. Ad esempio, i problemi più comuni nello sviluppo dei progetti software possono essere i lunghi tempi per lo sviluppo, budget, scadenze, oppure il bisogno di suddividere il lavoro totale tra diversi dipendenti. Anche la questione di collaborazione e cooperazione tra diversi tipi di personale potrebbe essere un problema, per via della comunicazione che potrebbe non essere adeguata. Inoltre, molti sistemi del mondo reale devono rispettare tanti requisiti e stare attenti ai vincoli, che possono anche cambiare nel tempo. Quindi bisogna capire come fare in modo di sviluppare questi sistemi di modo che si adattino nel corso del tempo. Per tutte queste cose un approccio scientifico non è sufficiente, per questo sono importanti il processo software e la metodologia.

1.3 Processo software

Definizione: Un *processo software* è una serie di fasi di attività svolte per costruire un sistema software. Ogni fase produce qualche artefatto che sono poi input per le altre fasi. Ogni fase ha un insieme di criteri di entrata e un insieme di criteri di uscita.

1.4 Meriti e problemi del processo a cascata

Il processo a cascata è stato usato per molti anni poichè sintetizza e semplifica tutto il processo di sviluppo suddividendolo per fasi come analisi, progettazione, implementazione, testing e manutenzione. In più pareva rendesse possibile un processo prevedibile, quindi facilmente gestibile. Il cuore pulsante rivoluzionario

di questo metodo insomma è la suddivisione in fasi. Tuttavia ci sono anche tanti inconvenienti con questo approccio. Intanto le numerose restrizioni e milestone rendono difficile rispondere ai cambiamenti dei requisiti. Inoltre, i cambiamenti ai requisiti sono necessari in molte circostanze, come nuove politiche aziendali o competizioni varie. Inoltre gli utenti non possono sperimentare con il sistema durante lo sviluppo, quindi non possono dare dei feedback fino a che il prodotto non sarà rilasciato. L'esperienza ci ha mostrato che invece i feedback durante le prime fasi aiutano a fare meglio il prodotto e in tempi molto più brevi.

1.5 Modelli del processo software

Dopo il modello a cascata gli esperti nel tempo hanno provato a pensare a un nuovo modello che potesse risolvere le sue lacune, e ne sono usciti di diversi tipi.

1.5.1 Processo di prototipazione

Il modello del processo di prototipazione è quello che più risponde al problema di dare all'utente modo di fornire feedback subito durante le varie fasi, fornendo appunto dei prototipi di vario tipo che possono variare da versioni del prodotto molto semplificate a semplici immagini del prodotto stesso.

1.5.2 Processo evolutivo

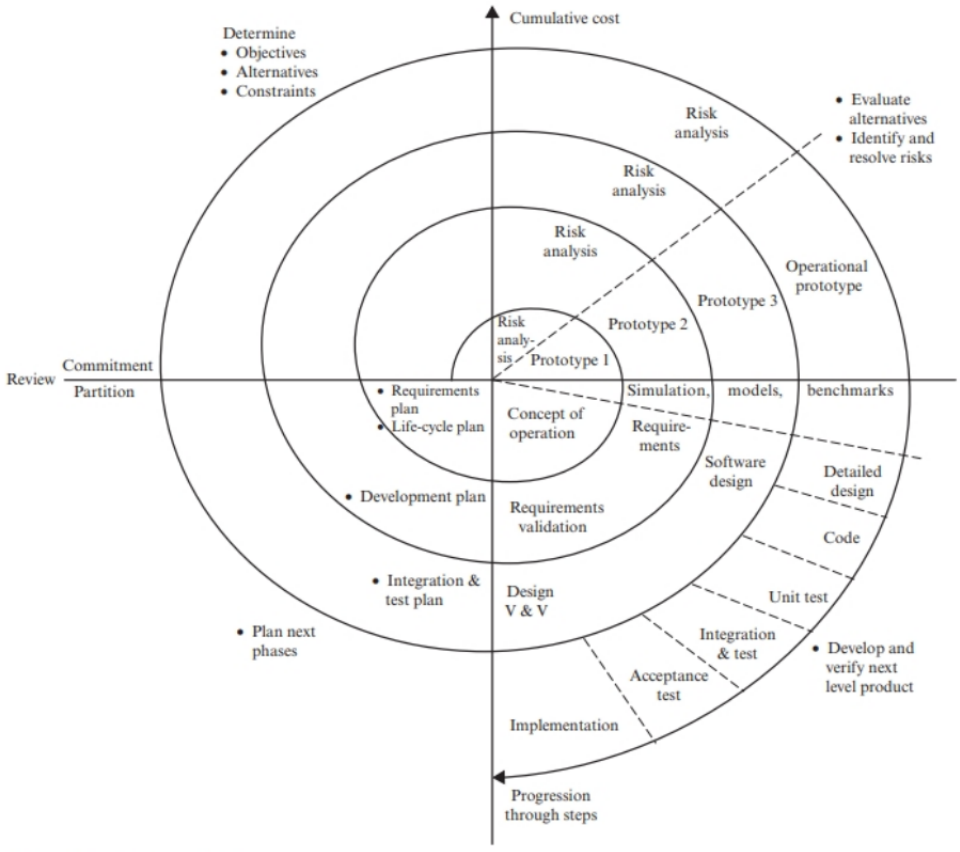
I prototipi aiutano molto per l'acquisizione dei requisiti, la validazione di essi e lo studio di ciò che gli utenti vogliono, oltre che per l'idea di progettazione. Tuttavia un'eccessiva prototipazione porta a un grande sforzo che in parte verrà sprecato. Il modello del processo evolutivo è mirato al fare evolvere i prototipi a seguito dei feedback, in maniera ciclica fino a che non ci sono più cose da aggiungere. Per questo motivo è anche chiamato modello di processo di prototipazione evolutivo, e riduce lo sforzo nella creazione di nuovi prototipi poichè si evolvono quelli già esistenti con delle iterazioni, ma non è comodo per progetti che richiedono un programma prevedibile di progresso.

1.5.3 Processo a spirale

Il processo a spirale è conosciuto per la sua unicità nella gestione dei rischi. Il processo segue appunto una spirale, della quale ogni ciclo è mirata a migliorare alcuni aspetti del sistema durante lo sviluppo come funzionalità, performance o qualità. In questo senso il sistema evolve in maniera incrementale. Il modello a spirale si basa su questi principi:

- Determinare gli obiettivi, alternative e vincoli del ciclo corrente.
- Valutare alternative, identificare e risolvere i rischi.
  - Se ci sono rischi, allora il sottopasso sarà nell'angolo in basso a sinistra.

  - Se nel ciclo precedente i rischi maggiori sono stati risolti, allora il sottopasso procede come nel processo a cascata.
  - Se il prototipo prodotto nella fase precedente è robusto abbastanza allora il sottopasso sarà esteso e utilizzerà il prototipo come nel modello di prototipazione evolutivo.
- Sviluppo e verifica del successivo livello di sistema.
- Pianificazione delle fasi successive.



1.5.4 Unified Process

Unified Process (UP) consiste in una serie di cicli, ciascuno dei quali porta ad un rilascio del sistema. Ogni ciclo ha diverse iterazioni che si possono principalmente suddividere in quattro fasi:

- **Inception (inizio):** Questa fase produce un diagramma dei casi d'uso semplificato, un'architettura provvisoria e un piano di progetto.

- **Elaboration (elaborazione):** Questa fase produce gran parte dei diagrammi UML che rappresentano il design architetturale, e i casi d'uso più critici sono qui progettati e implementati.
- **Construction (costruzione):** Durante questa fase, i restanti casi d'uso sono sviluppati in maniera iterativa e integrati nel sistema.
- **Transition (transizione):** Durante questa fase ci si incentra sul rilascio del sistema software, e ciò include anche il training degli utenti, beta test, correzione difetti e miglioramenti funzionali.

Si può dire che UP è use case driven, ovvero si concentra sull'individuazione di questi e ci si basa per lo sviluppo e il rilascio del sistema. È anche detto *architecture-centric*, perchè la sua architettura e progettazione appare presto nel ciclo di vita del software e ci si sviluppa in maniera iterativa e incrementale il resto.

1.5.5 Processo Agile

Il processo Agile è ottimo per il tipo di problema che è proprio lo sviluppo di un sistema software in quanto si incentra sul lavoro di gruppo, sulle consegne frequenti e nel breve periodo, frequenti meet-up con i clienti e sviluppo rapido, nonchè predisposizione al cambiamento. Il manifesto Agile si può riassumere in 4 punti fondamentali:

- **L'agilità valorizza gli individui e le interazioni su processi e strumenti**
  Si pensa che un buon processo software è fondamentale per un buon progetto software. Ma ancor più alta è l'importanza per un gruppo di saper lavorare in gruppo, perchè alla fine è quest'ultimo a portare a termine il processo software. Quindi la comunicazione tra i vari membri del team è importante, e qui entrano in gioco i vari diagrammi UML, ma solo secondariamente al saper interagire e al saper essere in grado di comunicare, che è l'essenza chiave per questo processo.
- **L'agilità valorizza il funzionamento del software su una documentazione completa**
  Per molto tempo si è pensato che una buona documentazione fosse motivo di buon software, il che è vero ma solo in parte, poichè ci sono alcuni casi in cui è impossibile capire effettivamente i requisiti finchè non ci si mette sul codice. A quel punto una buona documentazione potrebbe risultare poco utile. Infatti Agile privilegia il lavoro in corso, poichè è la base da cui partire e perchè in fin dei conti si consegnerà il sistema software al cliente, non una buona documentazione. Questo non significa che Agile è contro a una buona analisi e progettazione, ma sostanzialmente secondo Agile servono comunque entrambe ma il minimo indispensabile per poter poi sviluppare.

- **L'agilità valorizza la collaborazione del cliente rispetto alla negoziazione del contratto**
  Nei normali processi si dava priorità a incontrarsi col cliente per arrivare a stipulare un accordo o contratto, dal quale poi si partiva per sviluppare ciò che egli voleva. Il cliente a questo punto di tanto in tanto si riincontra col team e da giusto qualche feedback su qualche test. Ma decisioni molto importanti quali decisioni di progettazione sono presi dal team di sviluppatori e non dal cliente. Secondo Agile dunque il cliente deve essere praticamente parte integrante dell'intero processo,di fatto la sua collaborazione è essenziale per poter capire bene ciò che vuole. Durante i frequenti incontri con il cliente infatti si ha una comprensione reciproca di un sacco di cose, quali anche i limiti della tecnologia IT per quanto riguarda il cliente stesso, oppure scelte implementative concordate con il cliente stesso. Insomma la collaborazione del cliente aiuta entrambe le parti a comprendersi a vicenda per quanto riguarda limitazioni e priorità.

- **Valori di agilità che rispondono al cambiamento seguendo un piano**
  In molti vecchi processi si pensava che il cambiamento dovesse essere ristretto e controllato per evitare sforamenti di costi e risorse. In realtà, secondo Agile l'approccio al cambiamento è essenziale e migliore piuttosto che seguire un piano prefissato, poichè soprattutto nel mondo dell'IT il cambiamento è molto rapido e consistente, ma soprattutto si è in un settore dinamico protratto all'esemplificazione delle metodologie di sviluppo.

1.6 Metodologia di sviluppo software

Definizione: Una metodologia software definisce gli step o il come eseguire le attività di un processo software.

1.6.1 Differenze tra processo software e metodologia software

Un processo si focalizza sostanzialmente sul cosa c'è da fare in ciascuna fase, ma non specifica il come deve esser fatta. Infatti, una metodologia si preoccupa di come eseguire le attività del processo. Un processo si preoccupa di specificare input e output ma non di come rappresentarli, una metodologia si preoccupa di definire i passi e le relazioni tra essi, il criterio di inizio e di fine. In sostanza:
Processo:

- Definisce una struttura delle fasi delle attività.
- Specifica il COSA delle fasi
- Spesso non detta la rappresentazione degli artefatti
- È quindi indipendente dai paradigmi
- Una fase può essere realizzata con differenti metodologie

Metodologia:

- Equivale a una concreta implementazione di un processo
- Descrive i passi e il COME
- Definisce la rappresentazione degli artefatti
- È dipendente dai paradigmi
- Ogni passo descrive una procedura specifica, una tecnica o delle linee guida

Riassumento, una metodologia può essere vista come una concreta implementazione di un processo, e di conseguenza quest'ultimo può avere anche più di una metodologia.

1.6.2 Benefici della metodologia

Una buona metodologia permette agli sviluppatori di concentrarsi sul come far eseguire dei task e quindi di prourre un sistema software corretto. Una buon metodologia poi migliora anche la comunicazione e collaborazione perchè essa definisce un linguaggio comune per analisi, progettazione e modellazione, e definisce chiaramente i passi da seguire per portare effettivamente a termine i lavori da fare, rendendoli comprensibili a tutti. Migliora anche la qualità del design e del software perchè fondamentalmente forza gli sviluppatori a usare correttamente la modellazione, analisi e progettazione, e inoltre il fatto che le linee guida siano rivisitate da tutti i membri di un team fa in modo che si identifichino più facilmente eventuali falle, che portano a una riduzione di lavoro in fase di testing, debugging e riducono i costi di manutenzione. Una buona metodologia inoltre consente di individuare punti di forza e debolezza, misurando quindi la qualità del software in termini di costi di produttività e tempi e inoltre molti passi possono essere automatizzati, rendendo l'automazione software più semplice. In più una metodologia facile da imparare ed applicare permette di creare software di qualità anche ai principianti, e man mano che diventano più esperti potranno poi saltare alcuni passaggi.

1.6.3 Metodologie strutturate

L'analisi strutturata e il design strutturato sono due metodologie nate negli anni 70, e sostanzialmente erano costituite da diagrammi DFD (data flow diagrams), dei grafi orientati i quali vertici consistevano in entità esterne, processi aziendali e memorizzazione dati, mentre le frecce indicavano il flusso tra questi. L'approccio sostanzialmente è quello del Divide et impera, in quanto si dividono e decompongono processi complessi in processi più piccoli e più semplici fino ad arrivare a processi 'foglia', di facile implementazione diretta.

1.6.4 Classiche metodologie OO

Prima degli UML, 3 metodologie di questo tipo erano molto utilizzate: Il metodo Booch, L'Object Modeling Technique e lo Use Case Driven. Queste 3 metodologie sono poi alla base dell'UML 1.0, ed erano molto usate da molti sviluppatori ai tempi fino a che gli eccessivi costi per l'utilizzo di diverse metodologie, e la difficoltà di integrare e mantenere sistemi basati appunto su molteplici metodologie, portarono gli esperti a concordarsi in un modello unificato come UML e UP.

1.7 Principi Agili

I principi agili si possono riassumere in questi 10 punti:

- **Il coinvolgimento attivo dell'utente è fondamentale** e richiesto da molti metodi agili. Questo perché ci si è accorti che il problema fondamentale di ciascun essere umano sia proprio di non riuscire ad identificare bene tutti i requisiti subito sin dall'inizio, e quindi non ha senso impegnare tutto lo sforzo e gli incontri con l'utente soltanto nella fase di analisi, ma è meglio aumentare i meet-up con il committente durante tutto il processo di sviluppo, che avviene in via incrementale con piccole modifiche, cosicché l'utente si renda meglio conto di ciò che effettivamente vuole. Inoltre coinvolgerlo in maniera attiva è fondamentale perché comunque si migliora appunto la comprensione reciproca di determinate cose e questo ci porta a sviluppare un sistema software che è molto portato a soddisfare pienamente tutti i requisiti che il committente voleva. In questo modo il feedback dell'utente può avvenire in maniera costante e frequente nel tempo, di modo che se ci sono dei fraintendimenti si riesca a risolverli al momento giusto, senza dover aspettare di avere un prodotto finito o un prototipo, poiché i meet avvengono ogni settimana circa.

- **Il team deve essere rafforzato nel prendere decisioni,** nel senso che ciascun membro del team deve essere incoraggiato e eventualmente istruito nel comunicare con gli altri membri del team e col committente stesso, poiché questo lo porterebbe a prendere decisioni e a far sue le responsabilità del processo stesso, di modo da velocizzarsi sia nella comprensione che nello sviluppo.

- **I requisiti si evolvono ma i tempi sono fissi.** A differenza dei metodi standard infatti Agile è protratta verso il cambiamento, quindi si adatta sul cambiamento anche dei requisiti eliminando il vincolo di congelamento degli stessi. Questo significa che lo scopo del lavoro deve essere un buon approccio al cambiamento e all'evoluzione, ma nel rispetto dei tempi e del budget che invece sono cose che rimangono fissate.

- **Catturare i requisiti in maniera leggera e visuale a un livello alto.** Questo vuol dire che durante la fase di raccolta dei requisiti non si deve produrre un'eccessiva quantità di documentazione poiché questa rende difficile il cambiamento, che è un punto fondamentale. Infatti ci

si riduce a esprimere i requisiti con User Stories, casi d'uso, screenshot e altri elementi visivi e facilmente manipolabili, nonché comprensibili.

- **Sviluppare piccole parti, rilasciare in maniera incrementale e iterare.** Questo permette di rendere la fase di sviluppo visibile a piccole dosi per l'utente, che avendo meno feature da testare o guardare via via, gli sarà più semplice fornire un feedback. Inoltre ha come vantaggio la notevole riduzione di rischi a livello di errata progettazione.
- **Concentrarsi sulla frequente consegna di prodotti software.** In questo Agile fondamentalmente differisce dai suoi predecessori poichè la consegna, in base alla metodologia usata, varia da qualche giorno a un massimo di 3 mesi, e questo appunto come già detto migliora la comprensione e i feedback.
- **Completare ogni feature prima di passare alla successiva.** Questo significa che ciascuna feature deve essere opportunamente completata al 100% e anche testata. Per fare ciò ci sono vari metodi, uno è il Test Driven Development, dove ogni feature viene testata prima ancora della sua implementazione, oppure si usa testare i criteri di copertura, ovvero testare ogni singolo statement almeno una volta via via che si produce.
- **Ricorrere alla regola 80-20.** La regola sostanzialmente si basa sul fatto che l'80% del risultato è prodotto dal 20% delle funzionalità del sistema, quindi è bene concentrarsi sul 20% dei requisiti che produrranno quell'80%. Un esempio concreto potrebbe essere quello di un algoritmo perfettamente ottimizzato per rispettare un requisito, per la quale il tempo e lo sforzo per implementarlo non è valso la pena in quanto un algoritmo più semplice e similmente efficace in termini di tempo sarebbe stato meno difficoltoso.
- **Testare è integrato durante tutto il processo del ciclo di vita: testare spesso e presto.** In sostanza il testing è importante da svolgere per ogni nuova feature che si sta per consegnare, in quanto è importante che queste funzionino e siano pronte al 100%. Ci sono molti software ausiliari per questo tipo di approccio quali JUnit o altri strumenti di test di regressione.
- **Un approccio collaborativo e cooperativo tra tutte le parti interessate è essenziale.** Anche in questo principio si sottolinea l'importanza della collaborazione nei confronti del cambiamento e dell'evoluzione continuativa durante il ciclo di vita del processo. Importante quindi una buona cooperazione con altri membri del team e con il committente, piuttosto che una dettagliata e comprensibile documentazione all'inizio del processo.

1.8 Metodi Agili

La metodologia Agile segue i principi sopra esposti per filo e per segno oltre che rispettare un processo incrementale e iterativo come i predecessori quali

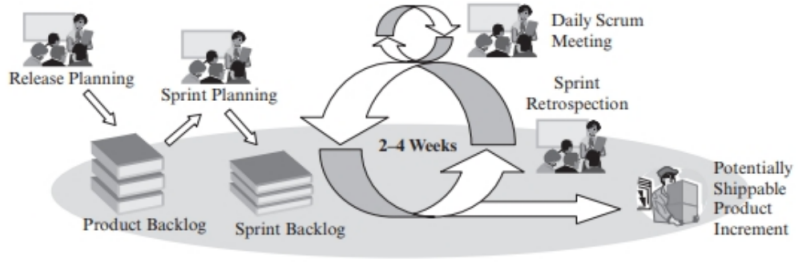
metodologia a spirale e prototyping evolutivo. Anche se tra loro cambiano alcuni nomi e notazioni, circa tutti coprono i requisiti, la progettazione, l'implementazione, l'integrazione, il testing e il rilascio durante ogni iterazione.

1.8.1 Metodologia di sviluppo dinamico di sistemi DSDM

In questa metodologia abbiamo sostanzialmente 5 fasi, delle quali le prime due sono fatte una sola volta mentre le altre sono iterative.

- **Studio di fattibilità**, ossia lo studio di quanto effettivamente il sistema sia fattibile e alla fine del quale si produce un Report ed eventualmente anche un prototipo, che potrebbe poi evolvere nel prodotto finale.
- **Studio commerciale**, durante il quale si dà diversa priorità ai vari requisiti, si abbozza un sistema di architettura preliminare e si produce un'area di definizione commerciale, un contorno di prototipazione e una definizione di sistema architetturale.
- **Iterazione del modello funzionale**, durante la quale si produce un prototipo incrementalmente e iterativamente costruito, basandosi sui vari feedback degli utenti e fornendo in fine una lista dei requisiti funzionali con i feedback dei committenti, i requisiti non funzionali e un'analisi dei rischi per gli sviluppi futuri.
- **Iterazione della progettazione e costruzione.** Durante questa fase invece si progetta e costruisce un sistema che rispetti i requisiti funzionali e non funzionali, e testato dai committenti, che danno un feedback.
- **Implementazione,** ovvero la fase in cui il prodotto viene installato nell'ambiente finale e dove si conduce un addestramento degli utenti, producendo un manuale utente e e un report con una recensione che riassume il risultato del progetto, scrivendo anche cosa fare in futuro.

1.8.2 Scrum



Scrum è una struttura che permette di migliorare il processo di sviluppo software, è formato da un team Scrum, con i vari ruoli, intervalli di tempo, artefatti e regole Scrum. È un approccio incrementale e iterativo che mira a ottimizzare la predicibilità e il controllo dei rischi, pianificando molteplici meeting.

Una caratteristica distintiva di Scrum è la loro suddivisione delle iterazioni, chiamate sprint, che durano mediamente 30 giorni, all'inizio delle quali c'è un meeting anche con il committente per pianificare e sviluppare ciò che va poi consegnato alla fine di esse. Un'altra caratteristica distintiva è il fatto di fare un meeting tra i membri del team ogni giorno, di 15 minuti, che permette loro di fornirsi aggiornamenti riguardanti progressi ai fini di migliorare la comprensione reciproca. Anche la retrospezione al termine dello sprint da parte dei membri del team è fondamentale

1.8.3 Feature Driven Development

Come suggerisce il nome, questa metodologia è guidata appunto dallo sviluppo delle feature, e consiste in 6 fasi, riassumibili in 4 punti:

- **Sviluppare un modello complessivo**, durante la quale un esperto del dominio fornisce un indicativo sommario di tutto il sistema, eventualmente scomponendolo in sottosistemi e fornendo una procedura dettagliata sul cosa va fatto. Questo permette anche di suddividere il team in base alle varie sotto-componenti delle quali realizzeranno un modello, ed infine coopereranno per realizzare il prodotto complessivo.
- **Costruire una lista di feature.** Si crea una lista di feature che rappresentano le funzioni commerciali che devono essere fornite dal sistema, e possono poi essere rifinite a feature a un livello più basso, dopodiché questa lista è revisionata da utenti e sponsor.
- **Pianificare in base alla feature.** Durante questa fase si produce un piano di progetto principale che guida allo sviluppo incrementale e alla consegna delle feature. Le feature vengono date ai capi programmatori che prendono le decisioni principali del team tra cui scegliere a chi distribuire le varie classi individuate nel modello generale, e costoro sono chiamati proprietari di classe. Questa organizzazione è anche chiamata organizzazione del capo programmatore.
- **Progettare e costruire in base alla feature e poi consegnare.** Queste fasi sono fatte in modo iterativo e gli incrementi sono progettati, implementati, revisionati e testati per poi essere infine consegnati. Ogni incremento dura da qualche giorno fino a qualche settimana.

1.8.4 Extreme Programming

XP è una metodologia Agile adatta a team piccoli che si trovano di fronte dei requisiti vaghi e variabili. Il principio guida di XP è quello di portare i principi e le pratiche di buon senso a livelli estremi, e consiste di 6 fasi:

- **Esplorazione:** è la fase durante la quale i membri del team di sviluppo e i committenti producono assieme le user stories per poter comprendere se si hanno sufficienti risorse per sviluppare il rilascio del prodotto. Le

user stories sono nella forma: As [role] I want [action]. Questa fase non dovrebbe durare più di 2 settimane.

- **Pianificazione:** è la fase in cui team di sviluppo e committenti lavorano insieme per identificare le user stories del prossimo rilascio, anche le meno significative secondo i clienti. Questo sviluppo potrebbe richiedere anche sei mesi, per questo questa fase è importante. Non dovrebbe durare più di 2 giorni.

- **Iterazioni per il primo rilascio:** è la fase in cui un'architettura di sistema generale è definita, il committente sceglie le feature da essere iterativamente implementate e successivamente testate dallo stesso fino a che non si raggiunge un prodotto utilizzabile. Ciascuna iterazione dovrebbe durare da 1 a 4 settimane.

- **"Produzionamento":** è quella fase in cui si individuano problemi di affidabilità e performance e infine si rimuovono per rendere il sistema utilizzabile in produzione, dopo essere stato opportunamente testato e certificato.

- **Manutenzione:** durante questa fase il prodotto subisce continue modifiche e miglioramenti, come ad esempio un maggior refactoring, un cambiamento di tecnologie o l'aggiunta di nuove stories da parte del committente. Si itera fino ad arrivare a un successivo rilascio.

- **Morte:** si arriva in questa fase o perché il sistema non può più vivere secondo le aspettative del committente oppure perché, durante la fase di manutenzione, sono stati via via soddisfatti tutti e completamente i requisiti del cliente e non ci sono più stories aggiuntive. Quindi in questa fase viene anche rilasciato un manuale per training, riparazione e riferimento.