Agile software development

Introduzione

Obiettivi:

- Comprendere il giusto punto di equilibrio tra documentazione e sviluppo

Spesso attività di contorno potrebbero rappresentare processi fuorvianti rispetto alla concreto sviluppo del sistema software, nonostante siano fondamentali per la comprensione dei requisiti funzionali posti. Potrebbero essere visualizzate come *pratiche burocratiche*, ossia l'insieme di elementi modellativi trattati non vengono mai sviluppati per ragioni legate alla costruzione del sistema software. A causa di questo elevato standard processuale non è riconducibile alcuna immediatezza tra la progettazione teorica e l'implementazione software, provocando una perdita della qualità.

L'impegno dovuto a garantire coesione a livello strutturale comprende un vasto insieme di azioni, le quali potrebbero provocare un'elevata perdita di tempo qualora l'analisi adottata non sia ben chiara fin dall'inizio. Da cui ne deriva l'inutilità di una pianificazione perfetta, poichè l'intero contesto è soggetto a dinamicità, soprattutto un campo che riguardi lo sviluppo e la progettazione software.

Non solo il *sistema software* dovrà essere flessibile ai cambiamenti, ma l'intero contesto sviluppato dovrà reagire prontamente a modifiche.

Manifesto per lo sviluppo agile

Il motivo che ha portato alla creazione del **manifesto**, è dovuto al netto spreco di risorse durante fasi di sviluppo software. Il termine correlato, *Agile software development*, non deve essere considerato come un metodo di progettazione, ma rappresenta un insieme di *pratiche* guidate da *principi* e qualità sia *interne* che *esterne*. Attraverso il processo di analisi prodotto occorre valorizzare un insieme di prospettive, quali:

- Interazioni tra progettisti e sviluppatori al di sopra di processi sequenziali e strumenti tecnici
- Sviluppare e adoperare codice piuttosto che predilire una documentazione esaustiva
- Imbastire una collaborazione con il *costumer*, denigrando una *negoziazione* conflittuale
- Reagire prontamente a cambiamenti, evitando di sottostare alle linee guida originarie

Il compito del *manifesto* prevede di attribuire maggiore rilevanza all'entità poste alla sinistra dell'elenco, provando a descrivere un approccio che possa portare ad un concreto beneficio per progetti futuri, senza escludere un prossimo utilizzo di tutti gli elementi posti alla destra.

Prinicipi

In relazione all'introduzione precedente, sono formulati di seguito i principi su cui stabilisce il proprio approccio il *manifesto*, suddivisi in:

- La priorità principale richiede che il *costumer* sia pienamente soddisfatto del risultato ottenuto, ciò può concretizzarsi solamente se sviluppatori optino per un continuo dialogo con i clienti e fornendo sequenzialmente versioni del sistema software implementato
- Reagire prontamente a modifiche e a variazioni di requisiti funzionali, sfruttando il cambiamento per ottenere vantaggio competitivo
- Valorizzare il *lifecycle* del modello a spirale, il quale impone certe temporalità in cui richiedere confronti e discussioni
- Giornalmente sviluppatori e il *business team* devono rendersi protagonisti nella realizzazione di task scelte
- Predilire il dialogo tra i singoli, poichè permette di diffondere in maniera efficiente ed efficace tutte le informazioni ritenute importanti
- Porre particolare attenzione a tecnologie abili ed eccellenti, relative non solo all'ambiente di sviluppo utilizzato, ma rispetto anche a novità progettuali o metodi differenti di cooperazione, pur di riuscire nell'intento della richiesta
- Adottare un approccio dedicato ad *improvement*, che tendi a migliorare ad ogni passo, piuttosto che formalizzarsi sulla pianificazione dettagliata all'inizio della progettazione

Metodi agili

Esistono molte pratiche che adottano l'insieme dei principi descritti precedentemente, alcune delle quali sono in totale contraddizione, tuttavia proprio questa discordanza dovrebbe dare vita a un processo legato al continuo miglioramento, pur di aumentare la qualità di progettazione e sviluppo. Le metodologie si suddividono in:

- *Code review*

  Definizione informale
  Con il termine *code review* si intende una pratica in cui una nuova porzione di codice deve essere analizzata e approvata prima di poter implementare nuove funzionalità.

  Come da definizione si adotta una revisione del codice affinchè possa essere poi incluso nella soluzione, da cui derivano un insieme di *tool* che certificano la correttezza o meno. Uno dei principali bonus di questa pratica consiste nella maggiore efficacia della soluzione illustrata, poichè è analizzata da un numero crescente di sviluppatori i quali si accertano del *code proposto*. Inoltre, adottando un approccio simile, si tende a condividere capacità e conoscenza, a causa della continua ricerca di nuove soluzioni legate a problemi che possano essere individuati conseguentemente ad azioni di *review*.
- *Test-driven design*

  Definizione informale
  Rappresenta uno stile di programmazione in cui tre attività si alternano, scrittura del *codice*, *test* della soluzione proposta ed infine *progettazione*.

  Test driven design può essere riassunto in un insieme di regole le quali prevedono un comportamento simile; innanzittutto si tende a descrivere le singole unità che indichino il processo esecutivo del sistema, per poi inizializzare la fase di testing, dove con molta

probabilità, trattandosi di una stesura iniziale, sarà carente di specifiche. Per superare l'errore si cerca di scrivere il codice più semplice possibile, che possa garantire la correttezza del test, anche se potrebbe comportare a soluzioni non adeguate. Infine, si attua *refactoring* del codice affinchè sia comprensibile e riutilizzabile; terminato, si ripete l'intero comportamento per ulteriori funzioni.

- *User stories*

  Definizione informale
  Indica un'illustrazione differente dei requisiti funzionali.

  Spesso sono espressioni scritte nel linguaggio del dominio che servono a catturare le aspettative dell'utente, per cui in grado di poter indirizzare lo sviluppo software. Molte *agile software development* tendono ad adottare *user stories* per rappresentare i requisiti, i quali non devono essere confusi con la progettazione modellativa posta da *use case*. Semplicemente, la propria sintassi prevede una costruzione come segue:

  As a <role>, I want <goal> so that <benefit>

  Ultima nota stabilisce il corretto uso del costrutto, in quanto deve essere più specifico possibile, altrimenti perderebbe di utilità, e garantire una stesura che promuova prima la descrizione dell'obiettivo per poi illustare il valore raggiungibile.

INVEST

User stories rappresenta uno dei *agile software development* di maggiore uso, dove ad un primo impatto potrebbe essere di facile uso e implementazione; tuttavia, la propria complessità comincia ad emergere di pari passo all'aumento della struttura del sistema software. Per questa difficoltà crescente si adottano strumenti per comprendere la qualità e l'efficacia delle *user stories*. I criteri sono riassunti come:

- *Indipendente*, ogni user story non dovrebbe dipendere da nessun altro elemento. Ciò potrebbe accadere qualora siano descritte user stories che illustrino un livello tecnico molto specifico, attuando sottili caratteristiche.
- *Negoziabile*, le user stories non possono essere intese al pari di contratti. Anzi la loro natura è il risultato di una *negoziazione*, il quale è sottoposto in un qualsiasi momento a nuove rivalutazioni
- *Valorizzabile*, come già accennato nell'introduzione, qualsiasi user story deve portare da un obiettivo ad un risultato *valorizzabile*
- *Stimabile*, il team di sviluppo dovrebbe apprendere il livello di complessità e la totalità del lavoro che caratterizza la user story realizzata
- *Piccola*, illustrando una realtà apparentemente minuta, che contraddistingue un'unica funzionalità, essa deve essere eleborata in una sola *iterazione*. Al termine del processo esecutivo, la funzionalità elaborata è considerata conclusa. Qualora non siano di piccola dimensione, per cui ritraendo problematiche di spessore, si attua la strategia dei *pattern*, in relazione ad una particolare avversità si adotta una soluzione ricorrente, anche se essa non prevede un uso di codice semplice e non articolato

- *Testabile*, indica uno dei punti cardine dei *metodi agili*, in cui l'implementazione di una user story è conclusa solamente quando è conforme al *test di accesso*. Qualora non sia verificata la conformità allora la progettazione della funzionalità non può essere ritenuta terminata