1 Introduzione

Lo sviluppo di un software segue le seguenti dinamiche: il cliente ha un problema e dei requisiti che vengono implementati nel software

Un software vive, e di conseguenza si evolve, durante questa evoluzione si sono notate delle caratteristiche come il cambiamento continuo, la complessità incrementale, il declino costante e la crescità continua.

Il processo software, o l'insieme di attività per lo sviluppo ed evoluzione di un programma, comprende specifica sviluppo convalida ed evoluzione.

Il processo segue un modello specifico (waterfall, iterative,...). A nostra disposizione abbiamo diversi metodi, strumenti e standard

2 Processo di sviluppo

Il processo di sviluppo stabilisce quando e come qualcuno fa cosa, per raggiungere un determinato obiettivo

2.1 Scelta e adattamento

Per identificare il modello più adatto dipende da vari motivi, tra cui anche il problema da affrontare o il team di sviluppo. In ongi caso le differenze principali fra i vari modelli sono:

- flusso delle attività
- dettaglio e rigore del processo
- coinvolgimento degli stake holders
- autonomia del team
- ...

2.2 Modelli

2.2.1 Modello a cascata

il processo di lavoro procede in maniera lineare, senza tornare indietro

$$\text{Comunicazione} \Rightarrow \text{pianificazione} \Rightarrow \text{modellazione} \Rightarrow \text{costruzione} \Rightarrow \text{deployment}$$

Questo modello porta il vantaggio della parallelizzazione, grazie alla sua struttura simile a una catenda di montaggio. Sfortunatamente, se durante la fase di costruzione si trova un problema nel design, necessità di ritornare alla fase di modellazione, rompento il modello

2.2.2 Modello di processo incrementale

Questo modello utilizza sempre il processo a cascata, ma a ripetizione, con incrementi costanti durante tutta l'evoluzione del software. Importante da ricordare che il sistema incrementale opera sul software solo per aggiunte, non torna mai ad aggiornare le funzioni precedentemente implementate.

2.2.3 Modello a prototipi

Nel caso il cliente non conosca precisamente le specifiche richieste nel progetto, allora si può procedere per prototipi. Seguendo questo modello, si entra in un ciclo di progettazione e feedback del cliente fino a quando non si raggiunge un prototipo che soddisfa i bisogni del cliente.

NB: viene definito nel Def ISO 13407

Il prototipo non deve essere per forza un prodotto finito o funzionante, ma anche un modello finto (o mock-up). Generalmente si costruscono un wireframe, ossia una bozza grafica per mostrare la user expirience, e poi successivamente ci si sposta sul mock-up

Uno svantaggio di questo modello è che il cliente potrebbe pensare che il prototipo sia un prodotto finito, e la successiva mediazione con gli sviluppatori può portare a uno sviluppo rapido e di qualità scadente.

2.2.4 Modello a spirale

Il modello a spirale sfrutta una ciclicità basata sul feedback del cliente, per poi riprendere le fasi di sviluppo partendo dai risultati del confronto

2.2.5 Sviluppo a componenti

In questo caso si sfruttano componenti software con funzionalità mirate e interfacce ben definite. I componenti più semplici risultano riusabili, ma non possono risolvere problemi complessi, per questo si può usare una composizione di componenti

2.2.6 Architettura orientata ai servizi

Una variante dello sviluppo a componenti va a sostituire le componenti con i servizi

2.2.7 Model-driven Development

2.3 Metodologie Agile

È un metodo di sviluppo che coinvolge il più possibile il committente, per ottenere una elevata reattività alle sue richieste.

2.3.1 Modello scrum

È un modello agile con l'obbiettivo di ridurre l'overhead organizzativo. È caratterizato da piccoli team di lavoro, costante testing e produzione di documentazione e frequenti incrementi software.

Il progetto viene diviso in piccoli blocchi di lavoro (sprint), che poi confronta i risultati con il cliente ed elabora le attività dell'immediato futuro (backlog). Infine, con cadenza giornaliera, i vari gruppi si organizzano in una riunione (daily scrum) per fare il punto della situazione

2.3.2 Extreme programming

È una prassi che si concentra su diversi punti riassumibili in feedback a scala fine, processo con/nuo, comprensione condivisa, benessere dei programmatori. Altri 3 punti molto importanti sono un'integrazione

delle modifiche frequente, così da limitare i conflitti dati dal nuovo codice, un refactoring piuttosto pesante e piccole ma frequenti realeases

La fase di plannign è caratterizzata dall'utilizzo delle user stories che descrivono funzionalità e caratteristiche, le quali vengono poi classificate e organizzate nelle tempistiche del progetto. Il cliente e il team di sviluppatori collaborano nella gestione e scelta delle user stories per la release successiva.

Nella fase di programmazione, gli sviluppatori lavorano a coppie (con ruoli distinti) e poi il loro lavoro viene integrato da un team apposito a quello del resto del team. Questo genera un'integrazione continua che riduce i problemi di compatibilità e interfacciamento.

Infine la fase di testing si avvale di strumenti per l'automazione degli unit test, mentre la valutazione del cliente si limita sulle funzioni e caratteristiche globali di sistema

2.3.3 DevOps

Il DevOps è un modello che si applica a grandi infrastrutture che necessitano di una gestione rapida e efficente tra il Development team e l'operation team. Il modello si basa sulla virtualizzazione nel cloud, su sistemi automatizati e metodi agile per l'interazione tra sviluppatori e sistemisti

2.4 Modularizzazione

L'Archittettura più diffusa è quella dei microservizi, molto ad

3 Linguaggi di modellazione

Un ingegnere del software per sviluppare un progetto necessità di un linguaggio comune per farsi da comprendere da altri ingegnieri.

La scelta del linguaggio può variare in base a vari parametri come facilità d'uso, comprensibilità per non tecnici, formalità e precisione

3.1 Linguaggio entity-relationship

Usato principalmente nel mondo dei database

3.2 Business Process Modelling notation

Lo standard OMG per la rappresentazione dei processi business è BPMN

3.3 Sicurezza

Diversi approcci SRE (Security Requirements Engineering) utilizzano dei linguaggi di modellazione, questo permette la creazione di documentazione più precisa e analisi automatizate. Questi linguaggi variano in base al obbiettivo che vogliono rappresentare

Alcuni di questo metodi usano delle basi di UML, o si concentrano sull'obbiettivo degli attaccanti

3.4 Linguaggio OCL

L'Object Constraint Language è un linguaggio funzionale usato per esprimere espressioni e vincoli sui modelli object oriented. È basato sull'utilizzo di query che vengono usate per specificare valori iniziali e derivati di un attributo, o indicare la condizione di un diagramma dinamico, oppure le query applicano dei vincoli di tipo invariante, precondizione, postcondizione o guardia.

In questo tipo di linguaggio il contesto di un'espressione è decisivo

4 Requisiti

I bisogni di un cliente possono essere espressi come scopi. Nello sviluppo di un progetto dovremmo creare delle funzioni che permettano di conseguire questi scopi.

Secondo Micheal Jackson, I requisiti sono la dati dalla somma di assunzioni sull'ambiente e dalle specifiche del sistema

I requisiti passano una definizione, ovvero quando vengono definiti per il cliente con un linguaggio comune, e poi la specifica, ossia un documento più strutturato e specifico scritto per gli sviluppatori. Il processo di acquisizione dei requisiti si chiama elicitazione

4.1 Difficoltà nell'analisi dei requisiti

Effettuare un'elicitazione può presentare delle difficoltà: per esempio uno stakeholder può non avere un'idea chiara o utilizzare un linguaggio tipico del dominio che l'analista non conosce, o magari diversi stakeholders possono avere richieste differenti. Questo poi provoca errori che più a lungo rimangono nascosti, più aumentano i costi di riparazione

4.2 Categorie di requisiti

I requisiti devono descrivere cosa il sistema deve offrire, e si dividono in funzionali e non. I primi descrivono cosa un sistema dovrebbe fare, mentre i secondi esprimono il come.

Requisiti funzionali

Descrivono le funzionalità del sistema software, in termini di servizi che deve offrire l'applicativo, di come reagisce a determinati input e altri comporamenti specifici. Questo tipo di requisito deve essere particolarmente completo e coerente

Requisiti non funzionali

Descrivono le proprietà del sistema in relazione a determinati servizi offerti. I requisiti non funzionali sono difficili da verificare, dato che sono formati sia da obiettivi generali che da obiettivi vaghi, e soggetti all'interpretazione dello sviluppatore.
Generalmente contiene misurazioni quantitative come velocità, dimensione, affidabilità,...

4.3 Forma del requisito

Può essere scritta in 3 modi: classico, user experience e caso d'uso

User story

seguono la forma del "as a ... I want ... so that ..." e contengono intrisicamente il criterio d'accettazione.

Use case

Il caso d'uso è uno studio degli scenari operativi degli utenti di un sistema, ossia i modi in cui il sistema può essere utilizzato. Nello specifico è una sequenza di azioni che producono un risultato osservabile da un attore. Queste sequenze vengono chiamate scenari e possono riguardare un caso base (di successo) o presentare degli scenari alternativi (in caso di insuccesso o varianti).

Il processo di creazione di uno schema richiede l'individuazione degli attori e degli scenari, e successivamente La creazione di relazioni tra i due

5 Diagrammi

5.1 Sequence diagram

This diagram shows how objects collaborate between them. It's composed typicaly by time (or the progression of it), messages (the communications between two objects) and the focus of control (the period while an object is performing an action)

5.2 State machine diagram

It describes the sequence of states and action an element can proceed. It's composed of a start and stop states and transitions divided in events, guards and actions. The action happen on entry, exit and while (do) in a state.

They can be semplified in a hierachical decomposition, but this causes information losses and cluttering

5.3 Activity diagrams

It follows the same logic of the state machine diagram, but adds forks, joins and decisions. They could be separated in swinlanes to show who is the actor of that action.

5.4 Diagramma di contesto

Modella l'interazione tra il software e entità esterne, tipicamenti sistemi superiori, subordinati, paritari e gli attori

La costruzione segue le seguenti fasi:

1. identificazione dei flussi d'informazione
2. identificazione delle entità esterne
3. rappresentazione del sistema da sviluppare
4. rappresentazione delle entità esterne
5. collegamento dei flussi d'informazione

Successivamente si può anche individuare la tipologia di sistemi e attori

5.5 Progettazione architetturale

A differenza del diagramma di contesto, questo tipo di progettazione si concentra sulla descrizione del sistema, in particolare al come deve essere realizzato.
Il risultato di di questa fase è il documento di progetto, composto dal progetto architetturale (dove descrive come il sistema è suddiviso in sotto sistemi) e il progetto dettagliato (le descrizioni dei singoli sotto sistemi)

5.5.1 Architettura

L'architettura è un'astrazione che permette di

- Valutare l'efficacia del design e le eventuali alternative
- definire l'organizzazione del software
- avere un modello di riferimento per il progetto e la comunicazione con gli stakeholders

Permette quindi la comunicazione delle parti interessate con un design partecipativo in modo da ridurre i rischi durante il progetto

5.5.2 Stili architetturali

Gli insiemi di uno stile architteturale sono le componenti, i connettori e i vincoli

5.5.3 Stili di controllo

Si occupano del flusso di controllo tra sottosistemi e si dividono in controlli centralizzati e controlli basati su eventi

5.5.4 Raffinamento

Definito il contesto si passa a una fase di specifica del sistema o raffinamento, dove si approfondiscono le componenti che compongono il sistema.

Questo passaggio avviene tramite l'astrazione delle procedure e il raffinamento progressivo dei dettagli di basso livello.
L'astrazione è di tipo:

- Strutturale: descrizione dell'aggregazione delle componenti
- Procedurale: descrizione della sequenza d'istruzioni di una funzione
- Dati: collezione di dati che descrivono un oggetto

5.5.5 Modularità

La suddivisione di un software in moduli rende il codice più comprensibile e mantenibile. Questo permette di ottenere l'indipendeza funzionale, ossia massima coesione interna ai moduli e minimo grado di accoppiamento tra moduli Questo metodo produce l'identificazione di un'architettura dei moduli (o structure chart).
Il metodo migliore per la creazione di moduli è il divide et impera

5.5.6 Coesione

Nella modularità, le funzioni possono essere concentrati in un singolo modulo o sparse fra tanti. La coesione è la misura che valuta quanto una funzione è svolta internamente a un singolo modulo o necessità di appoggi. Presenta sette livelli:

1. Casuale: nessuna relazione
2. Logica: elementi correlati, di cui uno viene selezionato dal modulo chiamante
3. Temporale
4. Procedurale
5. Comunicazionale: elementi correlati da una sequenza eseguita nella stessa struttura dati
6. Informazionale: ogni elemento ha del codice indipendente e un punto d'ingresso e d'uscita nella stessa struttura dati
7. Funzionale: tutti gli elementi sono correlati da una singola funzione

5.5.7 Coupling

Esprime la forza di interconnessione fra moduli in 5 livelli:

1. Content: riferimento diretto al contenuto di un'altro modulo

2. Common: due moduli che accedono alla stessa struttura dati
3. Control: un modulo controlla l'esecuzione di un'altro
4. Stamp: due moduli che si passano come argomento una struttura dati della quale usano alcuni elementi
5. Data: due moduli che si passano argomenti omogenei, cioé che usano tutti gli elementi

5.5.8 Riusabilità

Questo proprietà si riferisce all'utilizzo di componenti in differenti prodotti, riducendo costi e tempi di produzione. Si può applicare a moduli software, frameworks, design patterns e architettture software

5.6 Diagramma delle componenti

Modella le relazioni tra i componenti di alto livello di un sistema. È formato da entità e interfaccie fornite o richieste.

Le fasi di progettazione sono:

1. Identificare le componenti del sistema
2. identificare le interfacce appropriate per ogni componente
3. Collegare le interfacce che interagiscono fra di loro
4. Caratterizzare il dato trasmesso dalle interfacce tramite etichette

5.7 Businness Process Modeling Notation

A businness process is a set of activities that jointly realize a businness goal. This requires that organizations specify their flows of work.

BPMN is the OMG standard for representing businness processes. At his core we represent the components of a software system, the data they exchange and manipulate, and the various systems. In practice we represent:

- Events (circles)
  - Start
  - intermediate (double circle)
  - end (thick circle)
- Tasks (squares)
- Flows (arrows)
- Gateways (diamonds)

5.7.1 Events

Non-interrupt events are rappresented with a dotted line.

- Message
- Timer
- Conditional
- Link
- Signal
- Error
- Escalation
- Termination
- Compensation
- Cancel
- Multiple
- Multiple parallel

5.7.2 Tasks

Tasks, or activities, is a work performed within a businnes process. Are divided in tasks, subprocesses (+) and call activities (thick line)

Tasks:

- Abstract Task
- Service Task
- Send Task
- Receive Task
- User Task
- Manual Task
- Businness Rule Task
- Script Task
- Loop Task
- Sequential/Parallel Task

Sub-processes:

- Adhoc subprocess (tilde): The order of their constituent tasks is unspecified
- Transaction (double line): A set of activities that must be performed atomically, or indivisible
- Event Sub process (dotted line): Sub processes generated by events

Call activies: These are references to external processes or to recall processes already defined

- Abstract Call Activity
- User Call Activity
- Manual Call Activity
- Businnes Rule Call Activity
- Script Call Activity

5.7.3 Gateways

- Exclusive Gateways
- Inclusive Gateways
- Parallel Gateways
- Complex Gateways
- Event-based Gateways

  Event-based Gateways to start a process

  Parallel Event-based Gateways to start a process

5.8 Diagramma delle classi

Il diagramma delle classi differenzia le varie componenti con i loro attributi e operazioni, in più rappresenta le relazioni fra classi e relative informazioni come molteplicità e ruoli.
I vari attributi e operazioni di una classe mostrano anche il loro stato, cioé se pubblico (+), protetto (#) e privato (-).

Alcune associazioni possono diventare aggregazioni nel caso delle classi "fanno parte" di altre classi più estese. Le aggregazioni possono anche evolversi in composizioni, dove le classi dipendenti all'aggregazione non possono esistere da sole. Altre componenti del diagramma possono essere i vincoli e gli attributi derivati.

Le classi possono anche essere di tipo associativo, dove arrichiscono le informazioni date da altre classi alla quale sono collegate, o anche di tipo astratto

Il processo per la creazione del diagramma è:

1. Identificare le classi: definirne gli attributi e le operazioni
2. Stabilire associazioni: descrivere anche molteplicità e dipendenze