POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e di Produzione

Corso di Laurea Magistrale in Ingegneria Gestionale

Tesi di Laurea Magistrale

Metodologia Agile e DevOps

Approfondimento sulle metodologie e analisi di fattibilità circa l'applicazione delle stesse nel settore Automotive in ambito Digital Marketing, in progetti orientati allo sviluppo di siti web

Relatore

Prof. Marco Torchiano

Candidata

Jessica Oppedisano

Matricola

S144985

Anno Accademico 2020/2021

A mio marito e a mia figlia Diletta

Ai miei genitori

SOMMARIO

I. INTRODUZIONE

Le motivazioni che mi hanno spinto ad approfondire il tema riportato nel titolo di questo elaborato, hanno una duplice natura.
Da una parte il desiderio di approfondire e conoscere, dal punto di vista teorico, le principali metodologie, dall'altro l'interesse di analizzare quale approccio viene utilizzato all'interno di una grande organizzazione che opera nel settore Automotive, in particolare su progetti orientati allo sviluppo di prodotti digitali come i siti web.

Il tutto è stato influenzato e sicuramente incentivato da un'esperienza lavorativa di oltre dieci anni nella gestione di progetti orientati allo sviluppo di software/applicativi, in ambito Information and Communication Technology.

Nel corso degli anni '90 sono emerse un certo numero di metodologie di sviluppo software, le cosiddette metodologie "agili", che hanno dato vita al corpo di metodologie definite *Agile Project Management*.
Molte aziende, anche quelle di grandi dimensioni, stanno iniziando sempre più ad avere un approccio Agile ed è forte il legame tra questa metodologia ed il mondo della *Digital Transformation.*
Senza dubbio le aziende stanno iniziando, oltre ad informarsi ed adottarlo, a personalizzarlo rispetto alle proprie esigenze.
La domanda che pongo e che sarà oggetto di analisi e studio di questa tesi di laurea è se, Project Management Tradizionale e Agile, possono essere adottati insieme.

Vedremo che l'Agile Project Management può non essere considerato come una estremizzazione del Project Management tradizionale, ma bensì come una sua evoluzione in contesti particolarmente difficili e dinamici, come quelli sviluppati nel settore IT.

La convivenza simultanea dell'approccio tradizionale con uno più Agile, vedremo che sarà fattibile e che porta a dei benefici. La presenza di una metodologia non implica l'assenza dell'altra. L'elemento che non deve mai mancare per una corretta riuscita di un progetto, è la presenza di una buona Governance e di un approccio che sappia integrare diversi modi di lavorare, traendo da ciascuno di essi la migliore sinergia in termini di processi e best practice che possa risultare utile ed efficace.

Nei capitoli che seguiranno, verrà innanzitutto contrapposto il modello di sviluppo Agile al modello classico di tipo Waterfall. In particolare, verranno sottolineati i punti deboli di un modello 'a cascata' non più idoneo alle esigenze di business attuali, introducendo uno sviluppo agile, che intende massimizzare l'efficienza, riducendo i costi prestazionali e temporali.

Seguiranno approfondimenti di tre principali Framework Agile (Scrum, Lean Software Development, Extreme Programming).
Vedremo inoltre che per raggiungere quelli che sono i principi descritti nella metodologia Agile, è necessaria una nuova etica denominata DevOps.

DevOps risulta essere necessaria per dare vita alla nuova logica implementativa. Si passerà dunque a descrivere i princìpi di realizzazione di questo nuovo asset. Dopo aver trattato i princìpi di base, nel quarto capitolo verrà descritta e approfondita la struttura di una pipeline di rilascio del software, la quale si propone di automatizzare l'intero processo di produzione. La Continuous Integration non sarà più limitata al versioning del codice, ma diventerà il motore di una architettura, ben più complessa, in cui il testing ed il feedback continuo assumono un ruolo fondamentale.

Infine, nell'ultimo capitolo, verrà descritto il caso di studio. In particolare si vuole analizzare l'approccio adottato all'interno di una realtà aziendale, molto legata alla cultura tradizionale, ma che spinta dalle esigenze di mercato ed ancor più accelerata dagli impatti del COVID-19, sta progressivamente cercando di orientarsi verso l'adozione di metodologie Agile, con l'obiettivo di ottenere una maggiore qualità e affidabilità del software da rilasciare, riducendo il time to market (Metodologia Agile).

II. APPROCCI ALLO SVILUPPO DI UN SOFTWARE

1 L'APPROCCIO WATERFALL

Il Project Management Body of Knowledge (**PMBOK** ® – giunto alla sesta edizione) descrive l'insieme delle prassi standard per la gestione di progetti.
PMBOK ® prevede un approccio "**Waterfall**".

Il metodo di gestione del progetto a cascata, è stato utilizzato con successo in molti progetti strutturali e di costruzione. Quando l'ingegneria e lo sviluppo del software hanno iniziato ad applicare metodi di gestione dei progetti, il *modello a cascata* (*Waterfall*) è stato tra i primi metodi utilizzati.
È un approccio sequenziale, a fasi, in cui l'avanzamento del lavoro scorre da una fase all'altra (come una cascata). Il progresso dipende dal completamento dei risultati della fase precedente per l'inizio della fase successiva.
Lo sviluppo del classico modello a cascata, viene attribuito allo scienziato informatico Winston W. Royce.
Nella pratica vengono utilizzate diverse versioni del modello a cascata. Attualmente sono in uso modelli che suddividono i processi di sviluppo in cinque fasi. Le fasi 1, 2 e 3, sono riunite in un'unica fase di progetto, chiamata analisi dei requisiti (*Requirements*).

- **Analisi:** pianificazione, analisi e specificazione dei requisiti
- **Progettazione**: progettazione e specificazione del sistema
- **Implementazione**: programmazione e test di modulo
- **Test:** integrazione di sistema, test di sistema e test di integrazione
- **Esecuzione**: rilascio, manutenzione, miglioramento

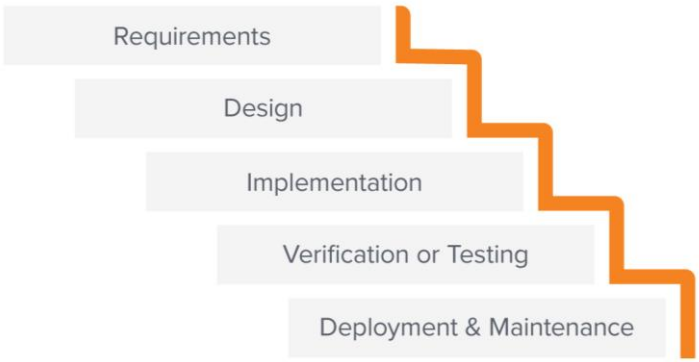


Figura 1 *- Schema approccio Waterfall*

Il metodo di gestione del progetto a cascata può essere utilizzato quando il team di progetto può raccogliere completamente i requisiti prima dell'inizio del progetto. I requisiti dovrebbero essere ben definiti e l'ambito del progetto dovrebbe essere semplice e compreso da tutti i soggetti coinvolti.

(Modello a cascata, 2019)

1.1 LE FASI DEL MODELLO A CASCATA

- **Analisi**

Ogni progetto di sviluppo di un software, inizia con una fase di analisi che consiste in uno studio della fattibilità e una definizione dei requisiti (*Raccolta dei Business Requirements*).
Durante lo studio di fattibilità, il progetto software viene valutato in base ai costi, ai profitti e alla realizzabilità. Dallo studio della fattibilità risultano un disciplinare (una descrizione generale dei requisiti), un piano di progetto e il calcolo di progetto come anche un'offerta per il committente.
Infine segue una dettagliata definizione dei requisiti, che comprende l'analisi della situazione effettiva e il progetto teorico. Mentre l'analisi della situazione effettiva delinea l'area problematica, nel progetto teorico si definiscono quali funzioni e caratteristiche deve offrire il software per poter soddisfare i requisiti. I risultati della definizione dei requisiti includono, ad esempio, delle specifiche tecniche, una descrizione dettagliata di come devono essere soddisfatti i requisiti del progetto e un piano per il test di accettazione.
La prima fase del modello a cascata si conclude con un'analisi della definizione dei requisiti, in cui i problemi complessi vengono scomposti in piccoli incarichi per i quali vengono elaborate le relative strategie di soluzione.

- **Progettazione**

La fase di progettazione prevede la preparazione di un concreto piano di soluzione sulla base di requisiti espressi dal business (il cliente), incarichi e strategie già individuati. In questa fase gli sviluppatori software elaborano l'architettura del software come anche un progetto di costruzione del software, concentrandosi su componenti concreti come interfaccia, framework e librerie. Il risultato della fase di progettazione sarà un documento di bozza con il progetto di costruzione del software e la pianificazione dei test per i singoli componenti.

- **Implementazione**

L'architettura del software, concepita nella fase di progettazione, viene realizzata nella fase di implementazione, che comprende la programmazione del software, la ricerca degli errori e i test modulari. Durante la fase di implementazione il software viene tradotto nella lingua di programmazione desiderata. I singoli componenti vengono sviluppati separatamente, testati durante i test modulari e integrati passo dopo passo al prodotto. Il risultato della fase di implementazione è un prodotto software che nella fase successiva verrà testato per la prima volta come prodotto completo.

- **Test**

La fase di test comprende l'integrazione del software nell'ambiente di destinazione desiderato. È possibile verificare se il software soddisfa i requisiti definiti in precedenza attraverso i test di accettazione sviluppati durante la fase dell'analisi. Un software che supera gli User Acceptance Test è pronto per il rilascio.

- **Esecuzione**

Nel momento in cui termina la fase di test, il software passa alla fase di esecuzione per l'uso produttivo. L'ultima fase del modello a cascata prevede rilascio, manutenzione e miglioramento del software.

(Modello a cascata, 2019)

1.2 ANALISI DEL MODELLO A CASCATA

Il modello a cascata offre una struttura organizzativa chiara per i progetti di sviluppo, nella quale le singole fasi del progetto sono chiaramente distinte. Poiché ogni fase si chiude al raggiungimento di un traguardo, il processo di sviluppo è facile da comprendere. Il fulcro del modello è rappresentato dalla documentazione delle fasi del processo. Le conoscenze acquisite sono registrate nei documenti di requisiti e progettazione.
In teoria il modello a cascata dovrebbe creare le condizioni per una realizzazione rapida ed economica dei progetti attraverso una pianificazione attentamente elaborata in anticipo.
Tuttavia l'utilizzo del modello a cascata per l'uso pratico è controverso. Da una parte, le fasi del progetto per lo sviluppo del software sono chiaramente distinte solo in rari casi. Soprattutto nei complessi progetti di software gli sviluppatori spesso trovano diverse componenti di un'applicazione presenti allo stesso tempo in diverse fasi dello sviluppo. Dall'altra parte, il decorso lineare del modello a cascata spesso non corrisponde alle condizioni reali.

Secondo il modello a cascata le modifiche non sono previste nel corso del progetto. In tal caso un progetto di software, i cui dettagli di sviluppo sono già stati stabiliti all'inizio, si porta a termine senza problemi solo se già nella fase iniziale si investe tempo e denaro nell'analisi e nella progettazione.

Il modello a cascata in cinque fasi basato su quello di Winston W. Royce mostra già un'estensione del modello: la verifica dei risultati di ogni fase, prendendo in considerazione i requisiti e le specifiche precedentemente elaborati.

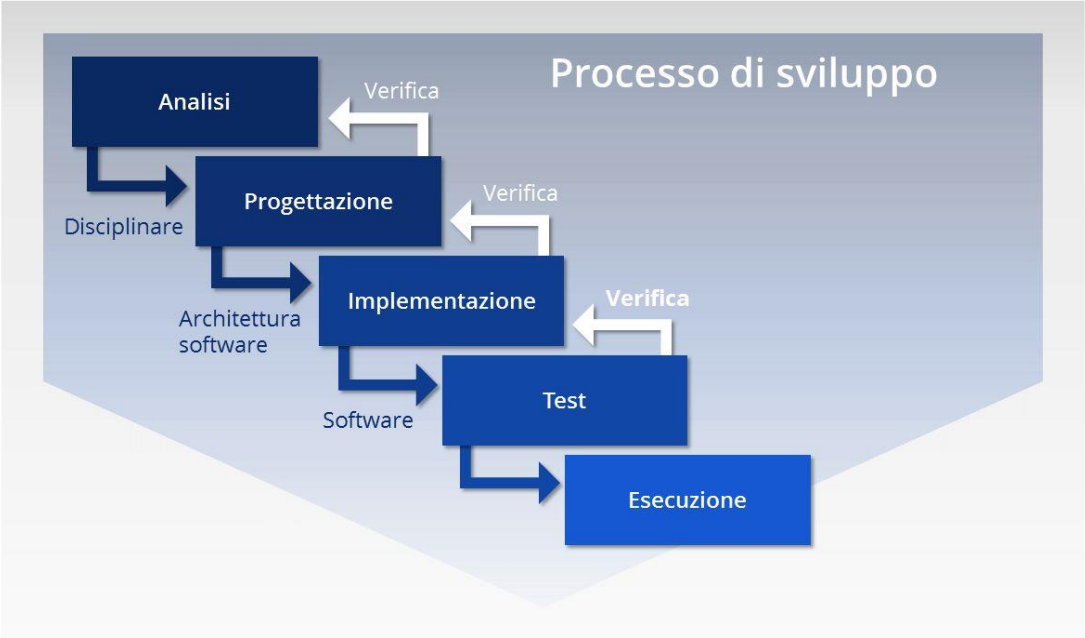


***Figura 2** - Waterfall - Processo di sviluppo*

(Modello a cascata, 2019)

Le estensioni del modello a cascata semplice completano il modello base con funzioni iterative, ad esempio confrontando e verificando i risultati di ciascuna fase con quelli della fase precedente.
Questo è quello che affermava Royce già negli anni '70. L'alternativa al ciclo di vita lineare da lui proposto, che è stato in seguito denominato modello a cascata, conteneva quindi tre estensioni essenziali:

- I risultati di ogni fase di progetto dovevano essere confrontati e verificati con i documenti elaborati in precedenza: ad esempio, bisognava assicurarsi che un modulo rispecchiasse i requisiti predefiniti già dopo il suo sviluppo e non alla fine del processo di sviluppo.
- Il modello a cascata doveva essere seguito per almeno due volte: prima per lo **sviluppo di un prototipo** e poi per lo **sviluppo del prodotto software vero e proprio**.

- L’inclusione dell’utente finale nel processo di produzione. Royce proponeva di includere l’utente in tre occasioni nel processo di sviluppo del software: durante la pianificazione del software nella fase dell’analisi, tra la progettazione del software e l’implementazione e durante la fase di test prima del rilascio del software.

Vantaggi del modello Waterfall

- ✓ Si tratta di una struttura semplice, attraverso fasi di progetto chiaramente distinte
- ✓ E’ possibile generare documentazione del processo di sviluppo attraverso traguardi ben definiti
- ✓ Già all’inizio del progetto è possibile stimare il costo, la stima per la realizzazione dello stesso
- ✓ Il progetto si può ben rappresentare lungo un’asse temporale

Svantaggi del modello Waterfall

- ✓ I progetti più complessi difficilmente si possono suddividere in fasi di progetto chiaramente distinte
- ✓ Poca possibilità di modificare in fase di avanzamento del progetto i requisiti
- ✓ L’utente finale è coinvolto solo nel processo di produzione, dopo la programmazione
- ✓ Gli errori si individuano solo alla fine del processo di sviluppo

(Modello a cascata, 2019)

2 L'APPROCCIO AGILE

2.1 INTRODUZIONE

"Agilità è la capacità di creare e rispondere ai cambiamenti nell'ottica di migliorare il profitto [o produttività] in un ambiente aziendale turbolento. Agilità è l'abilità di trovare un equilibrio tra flessibilità e stabilità"

(Highsmith, 2002)

(Corbucci, Seconda Edizione)

L'espressione Metodologia Agile (o sviluppo agile del software, in inglese Agile Software Development) si riferisce a un insieme di metodi di sviluppo del software emersi a partire dai primi anni 2000 e fondati su un insieme di principi comuni, direttamente o indirettamente derivati dai principi del "Manifesto per lo sviluppo agile del software" (Manifesto for Agile Software Development).
Tra l'11 e il 13 Febbraio del 2001, in una stazione sciistica sulle montagne dello Utah, diciassette persone si sono incontrate per parlare, sciare, rilassarsi.
Il risultato è stato il **Manifesto per lo Sviluppo Agile di Software** (Agile Software Development Manifesto).
L'Agile Project Management comprende molte metodologie per progetti dell'Information Technology (in seguito IT) e non solo, come: XP (eXtreme Programming), Scrum, DSDM (Dynamics System Development Method), Crystal Clear, EVO (Evolutionary Development), Lean Software Development, ed altri metodi/approcci.

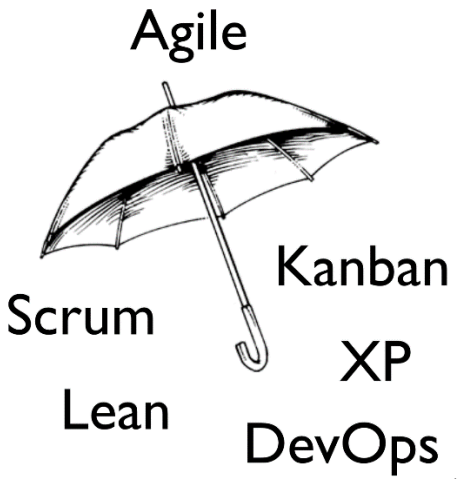


Figura 3 *– Metodologie Agile*

I rappresentanti di queste metodologie, erano accomunati dalla necessità di trovare un'alternativa ai pesanti processi di sviluppo software e alla stesura della relativa documentazione.

I metodi agili si contrappongono al modello a cascata (Waterfall model) e altri modelli di sviluppo tradizionali, proponendo un approccio meno strutturato e focalizzato sull'obiettivo di consegnare al cliente, in tempi brevi e frequentemente (*early delivery/frequent delivery*) software funzionante e di qualità.

La formalizzazione dei principi, su cui si basano tali metodologie è stato oggetto di lavoro di gruppo di progettisti software che si sono riuniti nella cosiddetta **Agile Alliance**.

2.2 SIGNIFICATO DI AGILITÀ

Agilità è la capacità sia di rispondere che di creare cambiamenti ("*abbracciare il cambiamento*" rappresenta uno slogan dell'Agile Project Management). Cambiamenti nelle esigenze, nei requisiti e nei bisogni del cliente finale fanno dell'Agile una metodologia dinamica e non prescrittiva o *plan-driven*, dove deve guidare un piano realizzato a priori.

Mentre in un approccio più tradizionale di Project Management il progetto terminava con il rilascio del deliverable finale, gli agilisti ritengono importante non solo quello che succede durante il progetto, ma anche quello che accadrà dopo che il progetto finisce.
Il concetto di beneficio (**outcome**) è qualcosa di più ampio rispetto a quello di **output**. L'output è legato al delivery, mentre l'outcome è relativo ai benefici attesi, alle aspettative collegate a quel particolare output. Ecco quindi che lo sguardo va esteso oltre la fine del progetto.

Il contesto nel quale si deve essere agili è un ambiente di business turbolento, dove la turbolenza può avere diverse origini, tra le quali:

- Turbolenza legata ai requisiti della soluzione, e questa turbolenza si traduce in continui cambiamenti di ambito, in ritardi, in maggiori costi che il progetto deve sostenere;
- Turbolenza legata alla tecnologia che si è scelto di adottare per realizzare l'oggetto di un progetto;
- Turbolenza legata alla numerosità del team di progetto:
- Turbolenza legata proprio al mercato di riferimento, alle scelte strategiche dell'azienda o al contesto territoriale e politico nel quale l'azienda si muove.

L'agilità è un'abilità capace di far convivere simultaneamente la flessibilità (il caos) con la stabilità (ordine). La flessibilità nel sapere rispondere ai mutamenti del mercato, del contesto e dei requisiti del cliente può essere raggiunta

acquisendo quei valori, principi e pratiche dell'Agile e costruendo un ambiente sano nel quale il team possa lavorare proficuamente aumentando così anche la sua produttività (Corbucci, Seconda Edizione).

2.3 I REQUISITI FONDAMENTALI DELLA METODOLOGIA AGILE

Affinché una metodologia possa definirsi Agile, questa deve possedere quattro requisiti fondamentali.
Deve essere:

- **Iterativa:** lo sviluppo è un processo iterativo, che costruisce versioni parziali di prodotto, esplodendole attraverso successivi e brevissimi periodi di tempo nei quali il prodotto stesso è oggetto di revisioni ed adattamento sia al contesto, in continua evoluzione, sia alle nuove richieste del cliente. Ogni iterazione deve essere **timeboxed**, ossia le iterazioni devono avere una durata fissa e breve (dalle due alle quattro settimane) e concordata anticipatamente tra tutto il team di progetto.

- **Incrementale:** il prodotto rilasciato con approccio Agile ha la caratteristica di essere un prodotto fatto di incrementi che il team rilascerà al termine di ogni iterazione.

- **Team Cross-Functional**: se il team si impegna a rilasciare micro incrementi di prodotto potenzialmente il team deve possedere tutti gli skill necessari e sufficienti per poterli rilasciare.

- **Team Empowered:** saper creare e rispondere ai cambiamenti. Il saper rispondere ai cambiamenti è la tipica gestione delle change request (richiesta di cambiamento rispetto a quanto previsto dal piano) argomento ampiamente dibattuto in un approccio tradizionale.

(Corbucci, Seconda Edizione)

Nell'Agile, il lavoro viene scomposto in una lista di piccoli e concreti deliverable e l'elenco viene ordinato in base alla priorità dettata dalle esigenze di business di quel momento. Subito dopo viene stimato lo sforzo rispetto a ciascuno di essi. Sebbene vengano stimati con maggior dettaglio gli elementi più in alto, generalmente le stime di effort e di costo non possono che essere top-down, considerato che il Backlog, questa coda prioritizzata delle funzionalità, è qualcosa in divenire.

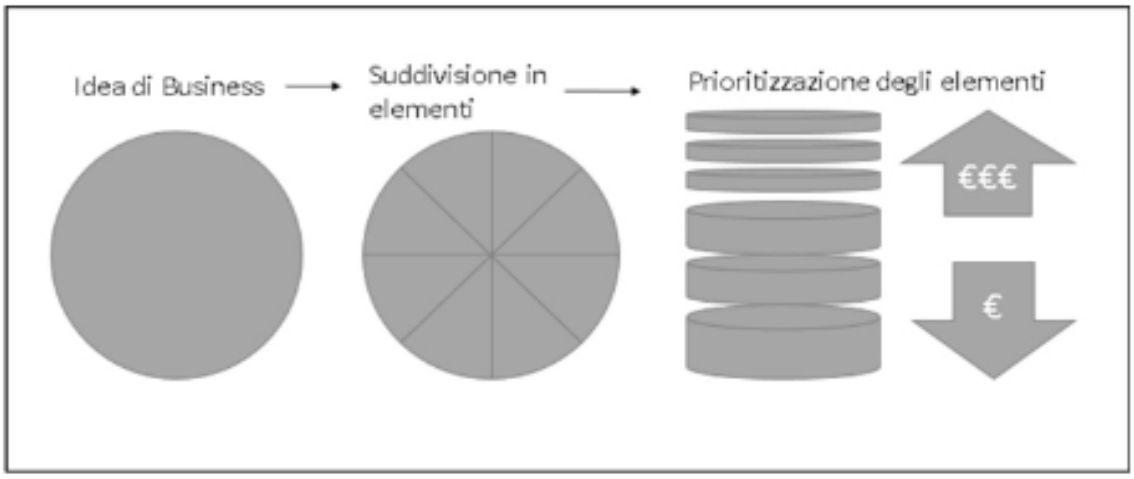


Figura 4 *- Scomposizione dell'Agile*

Il tempo viene frazionato in iterazioni corte e di durata fissa (normalmente 1-4 settimane), con codice potenzialmente consegnabile, dimostrato dopo ogni iterazione fino a ricomporre, iterazione dopo iterazione, il prodotto finale.
Il piano di rilascio viene ottimizzato e le priorità vengono aggiornate alla fine di ogni iterazione in collaborazione con il cliente. Essendo la metodologia Agile una metodologia empirica e non prescrittiva, i risultati ottenuti serviranno da appoggio per pianificare la prossima iterazione.
Il processo viene ottimizzato effettuando un processo di retrospettiva dopo ogni iterazione allo scopo di migliorare la produttività del team di progetto.

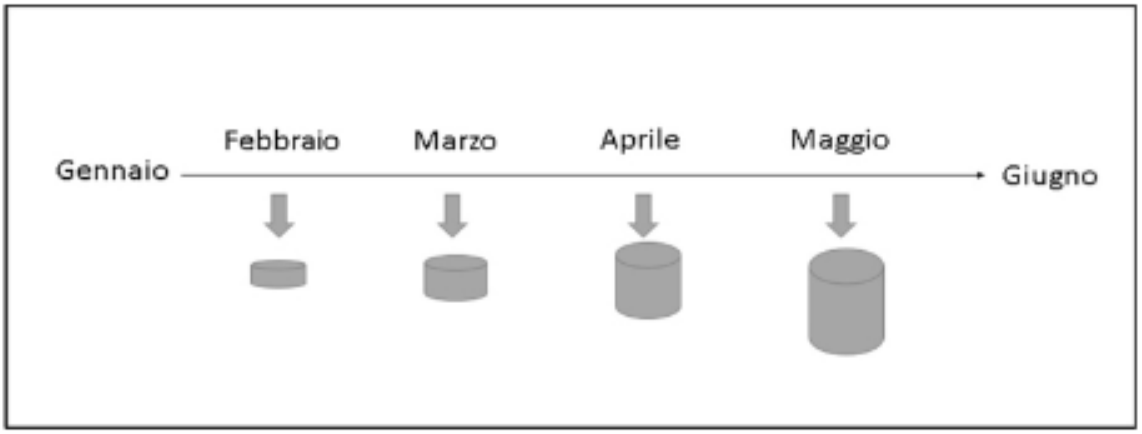


Figura 5 *- Ricomposizione nell'Agile*

(Corbucci, Seconda Edizione)

2.4 APPROCCIO TRADIZIONALE VERSO UN APPROCCIO AGILE

Avere un approccio Agile non significa non fare Project Management, ma farlo con un diverso paradigma.
Nell'approccio tradizionale, uno degli elementi chiave che consentiva una pianificazione molto dettagliata all'inizio, era proprio l'ambito progettuale (lo *scope del progetto*) che doveva essere concertato con tutti gli stakeholder chiave per poi andare a derivare le altre variabili come la schedulazione, il budget e la qualità.
Nell'approccio tradizionale l'ambito è considerato un elemento fisso e, in qualche maniera, da preservare da possibili richieste di cambiamento.
La possibilità di poter apportare cambiamenti al progetto è una eventualità che è resa possibile anche in un approccio più tradizionale e questo, naturalmente, per venire incontro a nuove esigenze del cliente o per una migliore comprensione dello stesso ambito progettuale.
Ma essendo la metodologia tradizionale di Project Management una metodologia predittiva (tipo la Waterfall, o plan-driven), la gestione dei cambiamenti viene trattata attraverso dei processi formali. Pertanto, la change diviene un cambiamento ufficiale della baseline di progetto solo a fronte di un processo di approvazione formalizzato al termine del quale si redige una nuova pianificazione, che diventerà la nuova baseline di progetto e che ufficializzerà non solo l'accettazione e validazione della change ma verranno formalizzati anche tutti gli impatti sulle altre variabili di progetto che sono state coinvolte dal cambiamento.

In un approccio Agile, al contrario, le variabili considerate fisse sono tempi, costi e qualità, quest'ultima considerata una variabile non negoziabile nel contesto Agile.

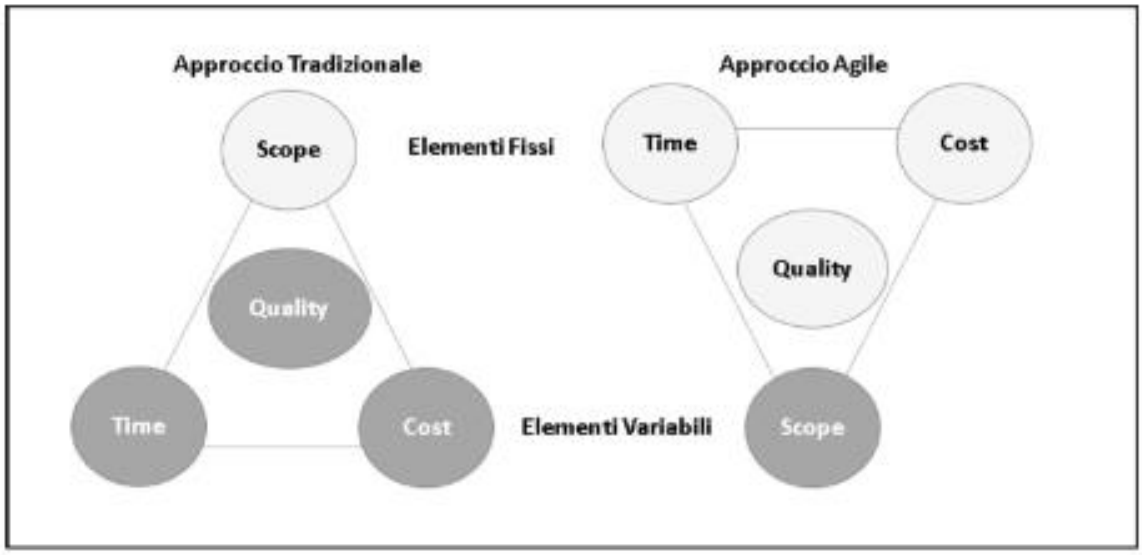


Figura 6 *- Approccio Tradizione e Agile*

Dalla *corrective action* si sposa il principio *dell'adaptive action* Un cambiamento all'ambito progettuale può comportare o una nuova riprioritizzazione dei requisiti o una nuova versione del Backlog.
In questo secondo caso, se l'ambito rientra in una gestione fissa (ossia il cliente vuole il prodotto con quelle funzionalità), il team di sviluppo sarebbe costretto ad incrementare la variabile tempo con nuove iterazioni e un conseguente aggravio anche lato costi. Se il cliente, invece, definisse fissa la variabile tempo, per l'esistenza di una deadline molto stringente, tutto il team dovrebbe collaborare in maniera molto più coesa per rivedere le priorità delle funzionalità, con la probabilità che alcune di queste non verranno rilasciate o, al massimo, andare ad inserire nel team nuove risorse ma sempre con la conseguenza di un incremento dei costi. Quindi, a ben vedere, anche nell'Agile il cambiamento non è gratuito.
Una volta definita la pianificazione e congelata la baseline di progetto, si entra nella fase di realizzazione ed il project manager acquisisce dall'organizzazione, o dall'esterno, le figure per espletare le attività di progetto precedentemente pianificate. Viene costituito il team allargato di progetto, ma il cliente continuerà a collaborare solo con il project manager e con il core team. In approccio Agile, il cliente verrebbe e dovrebbe essere correttamente coinvolto durante tutto il progetto e non solo in alcune fasi o momenti. Se voglio rispondere più velocemente al cambiamento e con maggior produttività e maggior feedback, il cliente non deve essere un elemento esogeno del team, ma deve essere parte integrante del team di progetto rivestendo il ruolo di *Product Owner*, responsabile del prodotto lato business (Corbucci, Seconda Edizione).

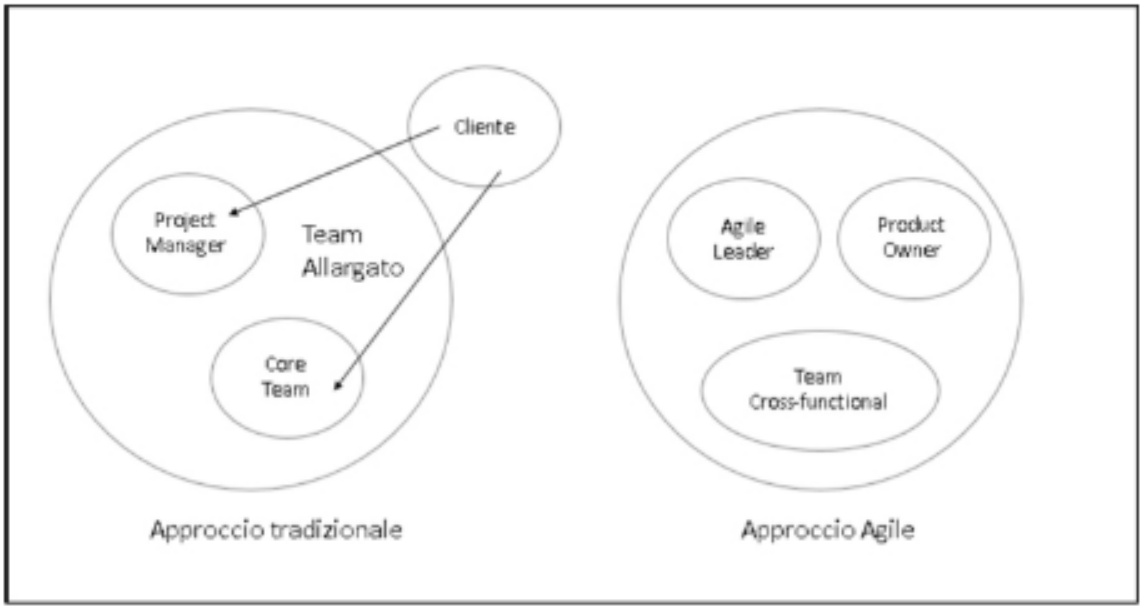


Figura 7 *- Evoluzione del team*

(Corbucci, Seconda Edizione)

2.5 MANIFESTO PER LO SVILUPPO AGILE DEL SOFTWARE

In riferimento anche a quanto descritto nel paragrafo precedente, possiamo riassumere che il **Manifesto** dell'Agile, al quale tutte le metodologie Agile, considera importanti i seguenti macro punti:

- Gli *individui e le interazioni* più che i processi e gli strumenti

  All'aumentare della complessità di un progetto, i processi e gli strumenti aiutano le persone a lavorare con maggior efficienza. Il primo valore del Manifesto, è incentrato sugli individui e il modo in cui interagiscono. Le relazioni sono molto importanti, non sono all'interno del team di progetto, ma anche tra team di sviluppo e il business. Sono le persone che creano il valore e non i processi. Sono le persone che aiutano a rimuovere ostacoli e problemi, collaborando.

- Il *software funzionante* più che la documentazione esaustiva

  Bisogna rilasciare nuove versioni funzionanti del software ad intervalli frequenti e trovare un giusto livello di documentazione realizzata. L'assenza di documentazione non va bene. La troppa documentazione neanche. Ad esempio, si può verificare il caso in cui il team potrebbe aver sviluppato un software che segue alla lettera tutte le specifiche riassunte nella documentazione, ma pieno di bug.

- La *collaborazione con il cliente* più che la negoziazione dei contratti

  La collaborazione diretta, offre i risultati migliori. Questo è il miglior contratto che si possa prevedere, ovvero quello che governa e regola il modo in cui il team e il cliente lavoreranno insieme.

- *Rispondere al cambiamento* più che seguire un piano

  Il team di sviluppo dovrebbe essere autorizzato a suggerire modifiche al progetto considerando il contesto mutevole.

Fermo restando il valore e l'importanza e il valore delle voci a destra, vengono considerate più importanti le voci a sinistra.

(Metodologia Agile, 2021)

2.6 I PRINCIPI DEL MANIFESTO AGILE

I principi dell'Agile Manifesto, possono essere così sintetizzati.

- *La nostra massima priorità è soddisfare i cliente, rilasciando software di valore fin da subito, e in maniera continua.*

Questi rilasci di funzionalità sono chiamati "*done-done*". Il primo "done" attesta la corretta analisi e interpretazione dei requisiti espressi dal cliente, il secondo "done" valida il requisito attraverso l'accettazione formale.

- *Accogliamo i cambiamenti nei requisiti, anche a stadi avanzati dello sviluppo. I processi agili sfruttano il cambiamento a favore del vantaggio competitivo del cliente*.

  Sono benvenuti i cambiamenti anche in fase di sviluppo. Non sono considerati come un rischio ma bensì come un'opportunità per ottenere un vantaggio competitivo.

- *Consegniamo frequentemente software funzionante, con cadenza variabile da un paio di settimane ad un paio di mesi, preferendo i periodi brevi.*

  La realizzazione del prodotto avviene per iterazioni che hanno la caratteristica di essere *timeboxes,* ovvero di durata fissa.

- *Committenti e sviluppatori devono lavorare insieme quotidianamente per tutta la durata del progetto.*

  Le persone del business e gli sviluppatori devono lavorare insieme, di giorno in giorno, durante il progetto. Questo favorisce la comunicazione, il feedback, l'iterazione e la gestione dei cambiamenti.

- *Fondiamo i progetti su individui motivati. Diamo loro l'ambiente e il supporto di cui hanno bisogno e confidiamo nella loro capacità di portare il lavoro a termine.*

  Costruire i progetti intorno ad individui motivati, fornendo loro l'ambiente adatto. L'Agile è lavoro di squadra (teamwork); sono le persone che creano valore. Per fare questo, il project manager deve sostenere e supportare il team di sviluppo e rimuovere qualsiasi ostacolo possa comprometterne la produttività, la fiducia e la comunicazione.

- *Una conversazione faccia a faccia è il modo più efficiente e più efficace per comunicare con il team e all'interno del team.*

  Una delle pratiche Agile è la cura del cosiddetto **Agile Space**: il team di progetto deve lavorare insieme nella stessa stanza. Spesso in azienda viene utilizzato il concetto di *war room.* Questo metodo aiuta anche ad evitare il numeroso volume di email, che vengono spesso scambiate e che comportano una continua interruzione e blocco del flusso.

Interessante è il concetto di Crystal Clear (comunicazione osmotica) che fa riferimento alla informazione catturata origliando quello che accade nella stanza.

- *Il software funzionante è il principale metro di misura di progresso.*

- *I processi agili promuovono uno sviluppo sostenibile. Gli sponsor, gli sviluppatori e gli utenti dovrebbero essere in grado di mantenere indefinitamente un ritmo costante.*

  Il team deve mantenere per tutta la durata del progetto, un ritmo costante ed essere allocato non più di 40 ore settimanali. L'obiettivo è quello di ottenere una produttività costante.
  Un team ben allocato è un team in grado di gestire anche eventuali emergenze.

- *La continua attenzione all'eccellenza tecnica e alla buona progettazione esaltano l'agilità.*

  Il team Agile deve tendere verso l'eccellenza tecnica perché è questo che fornisce valore e vantaggio competitivo al cliente.

- *La semplicità – l'arte di massimizzare la quantità di lavoro non svolto – è essenziale.*

  La semplicità consiste nell'assenza di Design non specificato o di funzionalità aggiuntive non richieste (*Scope Creep*).

- *Le architetture, i requisiti e la progettazione migliori emergono da team che si auto-organizzano.*

  In un team con questa caratteristica, ogni membro si assume la responsabilità del risultato gestendo il proprio carico di lavoro (workload). Un team che si auto-organizza non è caratterizzato da una mancanza di leadership ma da un diverso stile di leadership.

- *A intervalli regolari, il team riflette su come diventare più efficace, dopodiché regola e adatta il proprio comportamento di conseguenza.*

  Affinché il cambiamento possa essere gestito, affinché la collaborazione e la comunicazione possano essere più efficaci, serve un elemento che faccia da collante: il feedback. Nell'Agile esistono due pratiche caldamente consigliate a questo scopo: gli **Stand-up Meeting** ed il **processo di Retrospettiva**. Lo Stand-up Meeting viene tenuto tutti i giorni alla stessa ora per un massimo di

dieci minuti dove il team racconta brevemente le attività sulle quali lavorerà. Momento importante anche per capire se esistono ostacoli da rimuovere.

La **Retrospettiva** significa fare tesoro delle *Lesson learned* acquisite nella iterazione appena conclusa. Il processo di retrospettiva non si limita solo a capire quello che è andato bene o male ma definisce le azioni da implementare durante la prossima iterazione.

(Corbucci, Seconda Edizione)

I vantaggi della metodologia Agile

- ✓ Il ciclo di vita dello sviluppo del software è più rapido
- ✓ Il programma è prevedibile analizzando ciascuno sprint
- ✓ Focus sul cliente
- ✓ Flessibile nell'accettare i cambiamenti
- ✓ Promuove comunicazioni efficienti
- ✓ Ideale per progetti con finanziamento non fisso

Gli svantaggi delle metodologia Agile

- ✓ Agile richiede un alto grado di coinvolgimento del cliente, che non tutti i clienti sono disposti a dare
- ✓ Agile prevede che ogni membro del team sia completamente dedicato al progetto
- ✓ Comporta un aumento dei costi al fine di gestire cambiamenti di priorità e sprint aggiuntivi

(PM - Project Management, 2020)

PARTE III FRAMEWORK AGILE

3 TIPOLOGIE DI FRAMEWORK

I framework Agile più popolari sono i seguenti:

- *SCRUM* (Sutherland)
- *eXtreme Programming (Kent Beck 1999)*
- *Feature Driven Development* (De Luca & Coad)
- *Crystal* (Cockburn)
- *DSDM* (Dynamic System Development Method)
- *Lean Software Development* (Poppendieck)

Ogni metodo propone un diverso processo, ma tutti condividono gli stessi principi. Nei prossimi paragrafi, andremo ad approfondire tre framework (Scrum, eXtreme Programming e Lean Software Development).

4 SCRUM

Scrum propone un vero e proprio framework per la realizzazione di un prodotto software o, generalizzando, di un prodotto pensato, realizzato ed implementato in contesti turbolenti, tecnologicamente variabili e dai requisiti mutevoli.

Rispetto al framework di eXtreme Programming, descritto nel successivo capitolo, che viene utilizzato massicciamente in progetti di sviluppo software, Scrum può essere utilizzato anche in contesti di sviluppo prodotto non prettamente legati allo sviluppo software.

4.1 SIGNIFICATO DI SCRUM

Ideato e sviluppato da *Ken Schwaber* e *Jeff Sutherland* (cofirmatari dell'Agile Manifesto) si basa su tre semplici punti:

- Sprint
- Backlog
- Scrum Meeting

Trova radici nel Rugby ed indica proprio il pacchetto di mischia.

Questo termine vuole fotografare un team che, proprio come nel Rugby, si muove come un'unica entità sul campo (il progetto). Il team deve lavorare insieme in modo che tutti gli attori spingano verso la stessa direzione, agendo come un unico elemento coordinato (Corbucci, Seconda Edizione).

4.2 I VALORI DEL MANIFESTO IN SCRUM

Scrum rappresenta un framework ed una metodologia di Agile Project Management e, per essere tale, si ispira ai valori e ai principi contenuti nell'Agile Manifesto.
Ogni singola metodologia Agile propone un approccio leggermente diverso da questi valori.

- *Individuals and interactions over processes and tools*
- *Working software over comprehensive documentation*

Per facilitare la comunicazione e l'interazione, Scrum si basa su cicli frequenti di controllo e adattamento come lo *Sprint* al termine del quale il team rilascia porzioni di lavoro funzionante (quello che viene definito *Working Software*). Alla fine del singolo *Sprint* il team di sviluppo ottiene un feedback da parte del Customer, feedback prezioso per mantenere alto il livello di adattamento al contenuto e al contesto.

Un'altra pratica in Scrum è rappresentata da *Daily Scrum Meeting*, incontro giornaliero per verificare se il team sta andando nella giusta direzione. In questo frangente, il feedback è più orientato verso l'interno e sebbene il team lavori nella stessa stanza, la war room, questa pratica è consigliata per migliorare la comunicazione all'interno del gruppo di lavoro.

- *Customer collaboration over contract negotiation*

La collaborazione con il cliente è essenziale nelle metodologie Agile e in Scrum il cliente riveste il ruolo fondamentale di Product Owner. Questa è parte integrante dello Scrum Team.

- *Responding to change over following plan*

Scrum risponde ai cambiamenti attraverso continui cicli di pianificazione-esecuzione-monitoraggio di breve durata. La risposta al cambiamento è essenziale per creare un prodotto in grado di soddisfare il cliente e fornire un valore all'azienda.

I valori di questo framework possono essere così riassunti:

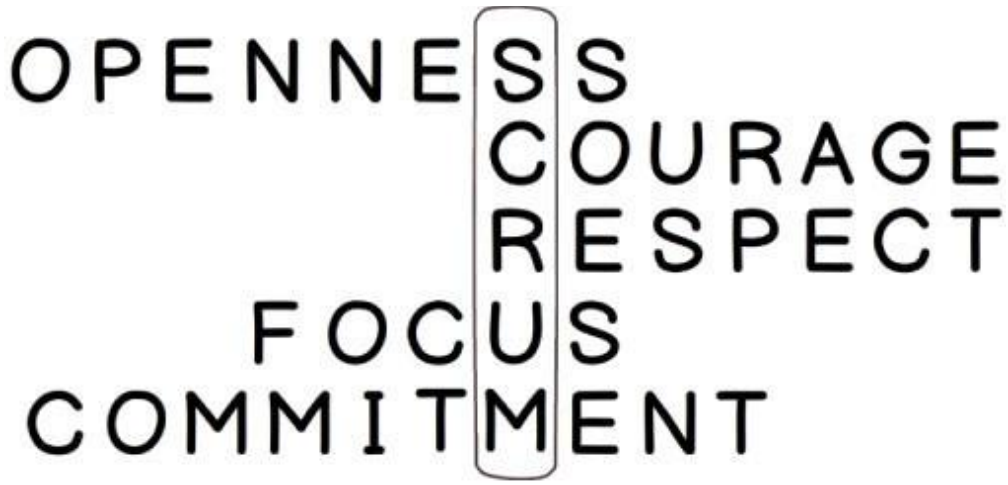


Figura 8 *- Valori di SCRUM*

- **Openness:** mentre si lavora insieme, si dichiara quello che si sta facendo, si esprimono emozioni e preoccupazioni affinché possano essere affrontate**.**
- **Courage:** poiché si lavora come team, ci si sente maggiormente supportati e si dispone di più risorse. Questa condizione dà il coraggio di intraprendere maggiori sfide.
- **Respect**: mentre si lavora insieme, si condividono sia successi che insuccessi e tutto questo porta le persone a rispettarsi e ad aiutarsi a vicenda.
- **Focus:** poiché ci si concentra su poche cose alla volta, si lavora bene insieme e si produce un lavoro eccellente. Si rimane concentrati sempre sugli elementi che hanno il valore più alto.
- **Commitment:** poiché si possiede un grande controllo sul proprio destino, questo consente di essere maggiormente impegnati, concentrati verso il successo.

(Corbucci, Seconda Edizione)

4.3 I PRINCIPI DI SCRUM

In Scrum, i principi (le regole) che devono essere seguiti dal team di progetto sono i seguenti:

- **Empirismo:** la conoscenza deriva dall'esperienza e le decisioni sono prese solo sulla base di quello che si conosce.
- **Trasparenza**: gli aspetti significativi del processo devono essere visibili ai responsabili del risultato finale.

- **Ispezione**: chi utilizza Scrum deve ispezionare frequentemente gli artefatti di Scrum e l'avanzamento verso un obiettivo con lo scopo di rilevare le eventuali deviazioni indesiderate.
- **Adattamento**: l'adattamento, al contesto e al contenuto, deve essere portato a termine il più rapidamente possibile per ridurre al minimo ulteriori deviazioni.

(Corbucci, Seconda Edizione)

4.4 IL CICLO DI VITA DELLO SCRUM

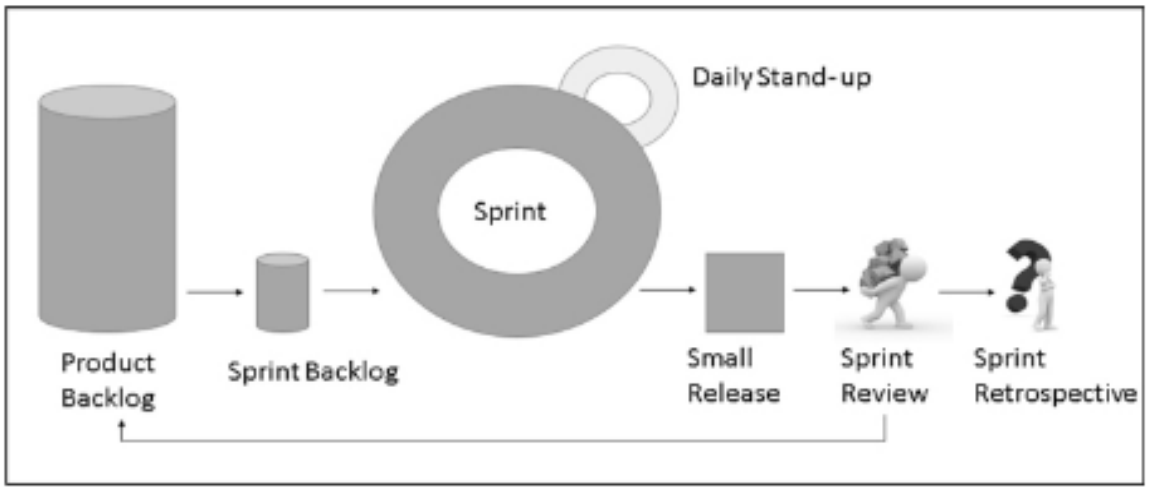


Figura 9 *- Il ciclo di vita dello SCRUM*

1. Primo passo: Product Backlog

Il primo passo è quello di identificare le *funzionalità* che il cliente vorrebbe vedere nel prodotto. Questo consente di andare a definire lo scope del progetto. Rappresenta la coda prioritizzata di una serie di elementi che devono essere realizzati.
Il committente, che in Scrum è identificato nel *Product Owner*, è l'unico responsabile della gestione, armonizzazione e prioritizzazione di questi elementi. Il *Product Backlog* è dinamico e cambia continuamente per identificare ciò che serve al prodotto per essere adeguato, competitivo e utile. Cambiamenti nei requisiti di business, nelle condizioni di mercato o nella tecnologia possono causare cambiamenti nel Product Backlog. A meno che il cliente non sappia veramente quello che vuole, un primo elenco di elementi deve andare a popolare il Backlog.

Attraverso cicli di rilasci (gli *Sprints*) e feedback continuo, il Product Backlog acquisirà uno spessore e con lui anche il prodotto che ne rifletterà la consistenza e congruità. Una particolarità apprezzata nello sviluppo Agile consiste di potersi fermare sia nella raccolta dei requisiti che nella realizzazione degli elementi presenti nel Backlog quando il Customer ritiene sia stato raggiunto un determinato valore per il business o raggiunti i limiti relativi a vincoli temporali ed

economici (si rammenti che l'ambito è una variabile negoziabile). Questo significa che il Backlog può essere congelato "sine die" o che alcuni elementi presenti nel Backlog non verranno mai implementati nel prodotto finale. Non tutto quello che esiste nel Backlog deve essere acquisito nel prodotto finale.

Il Backlog non solo rappresenta una serie di funzionalità e requisiti richiesti dal Customer, ma, più generalmente, è un elenco di tutto quello che potrebbe essere necessario per confezionare un prodotto che abbia valore ed essere in linea con le aspettative di chi lo dovrebbe, poi, anche utilizzare

Il team di sviluppo ha una corresponsabilità nel coadiuvare il Product Owner nell'elencare gli elementi del Backlog stesso. Non sarebbe corretto dare per scontato che il cliente sappia tutto sulla realizzazione di un sito web o sul miglior linguaggio di programmazione da utilizzare o sulla sequenza di passi necessari per un design architetturale.
Gli elementi che compongono il Backlog hanno una descrizione, un ordine e una stima. Il Product Backlog è spesso ordinato per valore, per rischio, per priorità e necessità. L'ordinamento dall'alto verso il basso degli elementi del Product Backlog guida le attività di sviluppo. Più alto è l'ordine di un elemento del Product Backlog, maggiore è il consenso su di esso e maggiore è il livello d'importanza ad esso assegnato (Corbucci, Seconda Edizione).

2. Secondo passo: Sprint Backlog

La fase successiva avrà l'obiettivo di identificare le funzionalità contenute nel Product Backlog che saranno realizzate nel prossimo Sprint e di creare quello che in Scrum prende il nome di **Sprint Backlog**. Per fare questo è previsto un incontro per la pianificazione dello Sprint (**Sprint Planning Meeting)**, che rappresenta un evento tipico in Scrum.

A questo meeting partecipa tutto lo Scrum Team con un duplice obiettivo:

- Identificare le funzionalità che saranno realizzate al termine del prossimo Sprint;
- Definire la metrica che aiuterà il team nel rilascio "Done" delle funzionalità (Definition of Done).

Il meeting è un incontro della durata di otto ore per uno Sprint di un mese. Per Sprints più brevi, l'evento è proporzionalmente più rapido

Nel primo, verrà indicato l'elenco delle funzionalità (o *User Stories*) che saranno rilasciate e nel secondo, il solo team di sviluppo elaborerà un elenco di task da terminare per produrre il micro-scope definito precedentemente insieme al Product Owner. Alla base di tutto questo, si instaura tra le due parti una negoziazione per definire la corretta capacity per il prossimo Sprint. Questa capacity è legata soprattutto alla produttività del team di sviluppo e a quanto

tempo (effort) i singoli componenti del team potranno impegnarsi nel rilasciare gli item del Backlog.

Prima di entrare nella vera e propria fase di esecuzione del progetto, nello Sprint, il team di sviluppo elabora altri due artefatti di Scrum: la **Definition of Done** ed il **Burndown Chart.**

- **Definition of Done:** si intende il "software funzionante" detto *Working software*
- **Burndown Chart:** rappresenta l'ammontare di lavoro rimanente che il team dovrà ancora completare per tutto il tempo allocato nello Sprint. Sull'asse verticale, il team posiziona il numero totale di Story Points relativi alle User Stories che sono state collegialmente scelte durante la fase dello Scrum Planning Meeting insieme al Product Owner.

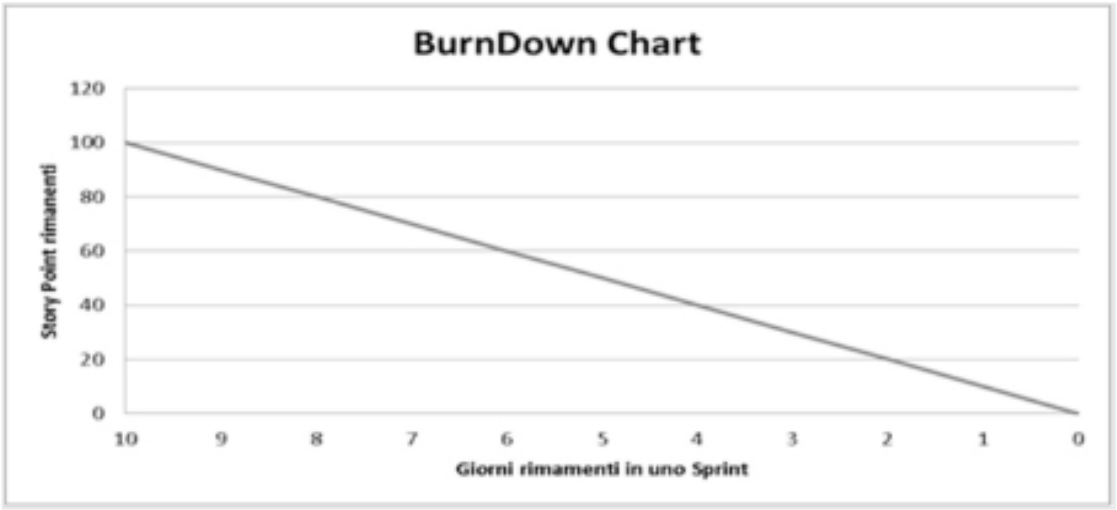


Figura 10 *- Burn Down Chart*

Lo Story Point è una metrica astratta che rappresenta la complessità di una *User Story* rispetto ad un'altra. Sull'asse orizzontale, invece, sono posizionati, in ordine decrescente, il numero di giorni rimanenti per uno Sprint.
Lo Sprint è sempre della durata fissa *timeboxed e* che vige la regola di massimo 30 giorni.

(Corbucci, Seconda Edizione)

3. Terzo passo: Lo Sprint

Definito un *Burndown Chart*, il team entra nella fase di realizzazione e parte lo *Sprint*.
Una delle regole che impone Scrum è che a Sprint iniziato, il Product Backlog viene congelato:

- Non sarà possibile aggiungere o rimuovere degli item durante il processo di realizzazione
- La composizione del team non deve cambiare.

Ogni Sprint può essere considerato un progetto con un orizzonte non più lungo di un mese. Come i progetti, gli Sprints devono realizzare qualcosa e quindi deve essere definito ciò che si va a progettare, codificare, implementare, testare e rilasciare in produzione.

Il team lavora per il tempo allocato sullo Sprint senza interruzioni. È vero che il team nella metodologia Agile deve essere un team disponibile, ma non si può non considerare la reale disponibilità delle risorse in un contesto aziendale standard nel quale possono convivere più progetti contemporaneamente, dove i progetti possono convivere con le operation aziendali, il tutto con risorse umane limitate.

La capacity di uno Sprint deve considerare i seguenti elementi:

- la capacità del team di lavorare su specifici item del Backlog;
- possibili assenze da parte dei suoi membri;
- attività che potrebbero esulare dal contesto progettuale.

Interessante è l'inserimento di un buffer che si comporta come un cuscinetto nel caso in cui qualcosa dovesse andare storto.

Durante il ciclo, il team si incontra ogni giorno per ispezionare e adattare il processo di sviluppo attraverso i *Daily Scrum Meeting*, importante evento topico in Scrum. Il Daily Scrum è un evento della durata di 15 minuti che serve al team di sviluppo per sincronizzare le attività e creare un piano per le prossime 24 ore.

Durante l'incontro, ogni membro del team di sviluppo spiega:

- che cosa è stato fatto dopo l'ultima riunione (che cosa si è fatto ieri);
- che cosa si farà prima della prossima riunione (che cosa farà oggi);
- quali sono gli ostacoli incontrati.

Il team utilizza il *Daily Scrum* per valutare i progressi verso l'obiettivo dello Sprint e per valutare come il progresso sia o meno in linea con il completamento del lavoro dello Sprint Backlog.

Il Daily Scrum incrementa la probabilità che il team di sviluppo raggiunga l'obiettivo dello Sprint. Il team di sviluppo s'incontra spesso subito dopo il Daily Scrum per ripianificare il lavoro rimanente. Alla fine dello Sprint il team rilascia le funzionalità che vengono riviste e accettate dal committente durante lo *Scrum Review Meeting*, altro momento topico di Scrum (Corbucci, Seconda Edizione).

4. Quarto passo: Lo Sprint Review

Alla fine dello Sprint si tiene l'incontro di *Sprint Review* per ispezionare l'incremento e adattare, se necessario, il Product Backlog. Durante la riunione di Sprint Review il team di sviluppo e gli stakeholder condividono e discutono su ciò che è stato fatto durante lo Sprint. Si tratta di un incontro informale e la presentazione dell'incremento ha lo scopo di suscitare commenti e promuovere la collaborazione. Il risultato dello Sprint Review è un Product Backlog rivisto che definisce quali probabili voci sono selezionabili per il prossimo Sprint. Il Product Backlog può anche essere modificato nel suo complesso per venire incontro a nuove opportunità di business (Corbucci, Seconda Edizione).

5. Quinto passo: lo Sprint Retrospective

E' l'occasione per il team di ispezionare se stesso e creare un piano di miglioramento da attuare durante il prossimo *Sprint.*

Lo scopo del processo di Sprint Retrospective è di:

- Analizzare l'andamento dell'ultimo Sprint per quanto riguarda le persone, le relazioni, i processi e gli strumenti;
- Identificare e ordinare i maggiori elementi che non sono andati bene e il potenziale di miglioramento;
- Creare un piano per apportare i miglioramenti al modo di lavorare del team.

Al termine del processo di Retrospective, parte un ciclo successivo.

(Corbucci, Seconda Edizione)

Un modo per raccogliere e classificare i feedback consiste nell'utilizzo del seguente template: (what-is-sprint-retrospective)

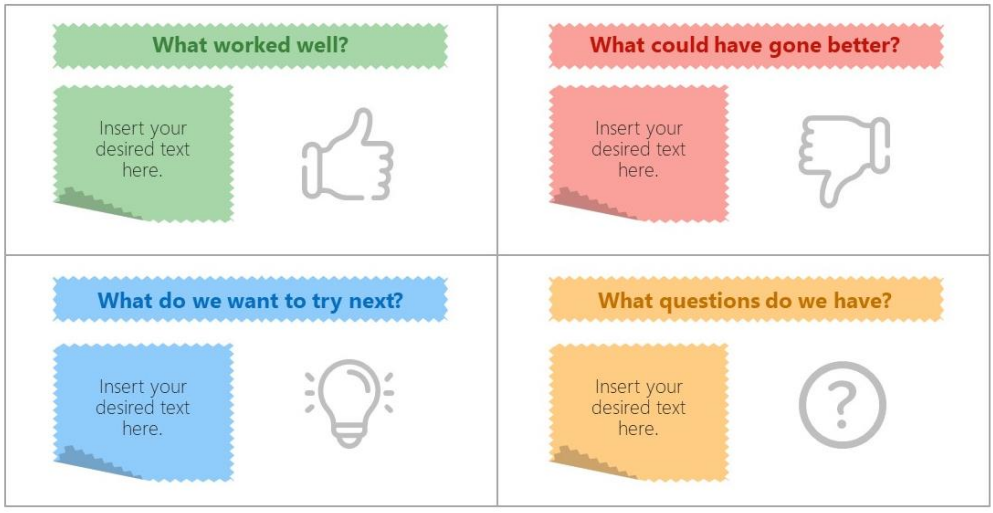


Figura 11 *- Agile Retrospective*

Il template consente di sintetizzare e riassumere quanto segue:

- Che cosa è andato bene: quali sono stati i successi del progetto
- Che cosa è andato male
- Cosa si desidera cercare di fare la volta successiva
- Quali sono i dubbi emersi

Indipendentemente dallo specifico framework, il lavoro di analisi a fine progetto è essenziale per migliorare e per perseguire successi futuri.

4.5 ATTORI DEL FRAMEWORK SCRUM

Lo *Scrum Team* è composto dalle seguenti figure:

- *Team di sviluppo*
- Il *Product Owner*
- Lo *Scrum Master*

Il Manager, il Project Manager, il PMO non appaiono tra i ruoli. Il motivo è che il framework di Scrum definisce i ruoli che sono specifici in Scrum e non i ruoli che possono esistere in una organizzazione che usa Scrum.

Il Team di sviluppo

Il team di sviluppo è un team che si auto-organizza e si struttura in maniera tale da essere indipendente nella gestione del proprio lavoro.

Il team di sviluppo in Scrum, deve essere così caratterizzato:

- Il team deve *auto-organizzarsi*. Nessuno (neanche lo Scrum Master) può dire al team di sviluppo come trasformare il Product Backlog in incrementi di funzionalità potenzialmente rilasciabili;
- Il team di sviluppo è *cross-Functional*, ossia il team deve possedere tutte le competenze necessarie affinché si possa impegnare a rilasciare un incremento di prodotto.
- I singoli membri dei team di sviluppo hanno competenze specialistiche e aree di focus, ma la responsabilità è del team nel suo complesso;
- Il team non contiene sotto-team dedicati a particolari domini come il testing o la business analysis.

Il concetto di agilità si riversa anche in termini di *numerosità* del team di sviluppo. La dimensione ottimale del team è abbastanza piccola da rimanere agile e grande abbastanza da completare il lavoro. Si può sostenere in forma generale che la dimensione dovrebbe essere compresa fra 3 e 9 membri. Avere team di sviluppo con meno di tre membri comporta una ridotta interazione e un minore guadagno in termini di produttività, oltre ad incontrare vincoli di competenza

durante lo Sprint. Questo renderebbe impossibile al team di realizzare un incremento potenzialmente rilasciabile. Avere più di nove membri rende il coordinamento molto oneroso. Team di sviluppo di grandi dimensioni generano troppa complessità, non facilmente gestibile nemmeno con un processo empirico quale è Scrum.
Le dimensioni ridotte del team rappresentano un aspetto comune alle diverse metodologie Agile. Scrum favorisce team di piccole dimensioni e questo permette di evitare che nel gruppo ci siano persone che oziano.
Inoltre, è più facile costruire interazioni tra i membri, si spende meno tempo per coordinare gli sforzi, si evita che qualcuno dei membri passi in secondo piano, aumenta la soddisfazione delle persone che appartengono a piccoli team e viene ridotta la probabilità che si verifichino delle super specializzazioni (nocive in Scrum).

Il product Owner

Il *Product Owner* dello Scrum Team, ha la responsabilità di massimizzare il valore del prodotto e del lavoro svolto dal team di sviluppo.
Il Product Owner è il Product Leader. Controlla il progetto da due punti di vista simultaneamente.
Da una parte cerca di capire i bisogni e le priorità degli stakeholder (cliente ed utenti in primis) e dall'altro comunicando al team di sviluppo cosa realizzare e in quale sequenza. Compito del Product Owner è quello di verificare che i criteri di accettazione del prodotto siano soddisfatti. È sua la responsabilità di validare l'ambito che il team ha prodotto alla fine di ogni Sprint. Suo è il compito esclusivo di gestire il Product Backlog.

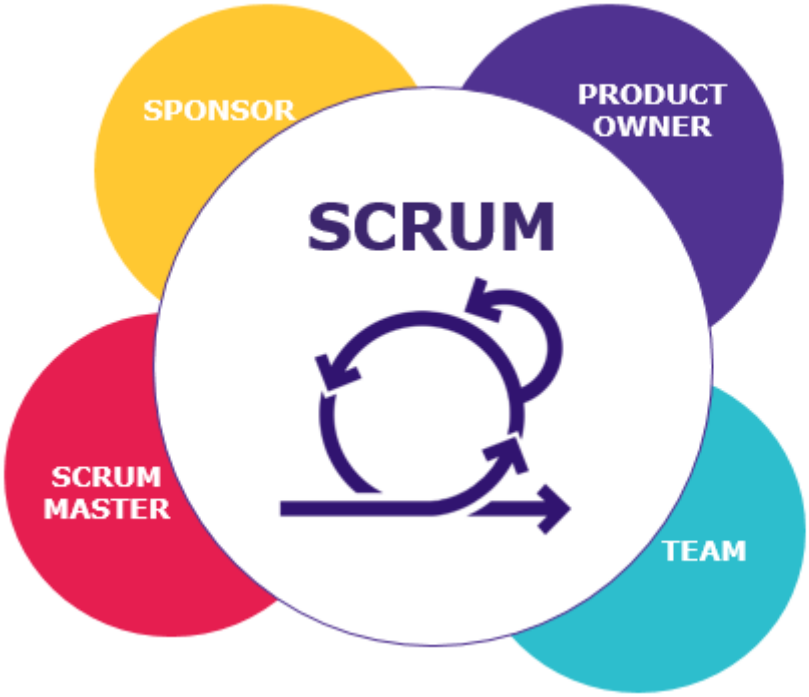


Figura 12 *- Scrum Agile Team*

Lo Scrum Master

Mentre il Product Owner è orientato alla realizzazione del giusto prodotto ed il team di sviluppo è orientato alla corretta realizzazione del prodotto, lo Scrum Master è orientato nell'aiutare chiunque voglia comprendere ed abbracciare i valori, i principi e le pratiche di Scrum.
Lo Scrum Master è il responsabile del successo di Scrum all'interno dell'organizzazione ed ha il compito di assicurare che i principi, le pratiche ed i valori di Scrum siano compresi, approvati e seguiti.
È l'interfaccia reciproca tra il Management (il Product Owner) e lo Scrum Team. In una organizzazione che inizia ad avere un approccio con Scrum, lo Scrum Master riveste il ruolo di coach. Lo Scrum Master è un leader e non un manager. Se il problema è un impedimento che il team non riesce a risolvere, lo Scrum Master ne acquisisce in toto l'ownership cercando di risolverlo. (Corbucci, Seconda Edizione)

Tale ruolo è chiave nell'agile, cosi come sono fondamentali le sue qualità:

Figura 13 *- Qualità di uno Scrum Master*

(Scrum-master-qualities)

5 LEAN SOFTWARE DEVELOPMENT

La metodologia Lean, al contrario delle altre metodologie nate nel contesto dello sviluppo software, trae origine dalla filosofia e dai principi del Lean Manufacturing del sistema di produzione della Toyota (Toyota Production System) che ha rivoluzionato il modo di costruire le automobili a partire dagli anni ’80.

Il Lean Manufacturing ha come obiettivi:

- Eliminare gli sprechi
- Semplificare e razionalizzare la catena del valore
- Produrre su richiesta (just in time)
- Concentrarsi sulle persone che aggiungono valore

Il *Lean Thinking* sfrutta l’intelligenza e l’esperienza dei lavoratori front line, con la convinzione che siano proprio questi a determinare e continuamente migliorare il loro modo lavorare, stando loro stessi direttamente in contatto o con il cliente o con il puro macchinario.
Le origini La filosofia Toyota (The Toyota Way) può essere riassunta in due pilastri fondamentali: “**Continuous Improvement and Respect for People**”.
Il *Continuous Improvement* è assimilabile a tutto quello che viene racchiuso nel termine Kaizen che non significa solo migliorare continuamente ma rimanda a concetti più ampi volti a standardizzare un paradigma lavorativo. L’aspetto più importante del Kaizen è il processo di miglioramento continuo che c’è alla base.
Si deve a Mary and Tom Poppendieck (Lean Software Development: An Agile Toolkit – 2003, Mary Poppendieck & Tom Poppendieck) il merito di aver contestualizzato la filosofia Toyota importandone principi e pratiche da un ambiente manifatturiero ad un ambiente di sviluppo software per dare vita alla metodologia Lean nell’Agile Project Management.

La metodologia Lean si articola in sette principi e ventidue pratiche.

1. *Primo Principio: eliminare gli sprechi (Eliminate Waste)*

Eliminare gli sprechi e investire più tempo solo su ciò che aggiunge e apporta valore reale al cliente.
Pertanto, il primo passo per l’attuazione dello sviluppo Lean sarà quello di imparare a riconoscere gli sprechi (Seeing Waste, la pratica) per poi scoprire quali possano essere le loro maggiori fonti ed eliminarle di conseguenza.

2. Secondo Principio: amplificare l'apprendimento (Amplify Learning)

Il secondo principio Lean vuole sottolineare l'importanza di una corretta informazione e comunicazione tra le parti in gioco. Amplify Learning significa aumentare il feedback e condividere la conoscenza quando si è di fronte a problemi complessi o mai riscontrati prima piuttosto che continuare a cambiare idea procrastinando, sine die, una possibile soluzione.
La metodologia Lean propone una serie di pratiche per amplificare l'apprendimento, tra le quali:

- **Feedback:** quando il team si trova alle prese con un problema o quando sorge un nuovo problema durante lo sviluppo software la prima azione da mettere in campo è quella di verificare ed assicurare che i cicli di feedback siano stati identificati correttamente per poi incrementarne la frequenza nelle aree maggiormente problematiche. Anziché raccogliere requisiti da parte di più utenti, un primo approccio potrebbe essere quello di mostrare loro potenziali soluzioni ottenendo un prezioso input. Invece di lasciare che i bug si accumulino, eseguire il test non appena il codice è scritto.

- **Iterazioni**: uno degli elementi caratteristici di tutta la metodologia ed approccio Agile è l'iterazione. Cicli di produzione software che coprono un orizzonte temporale breve migliorano il controllo ed il feedback del cliente. L'iterazione deve essere interpretata come punto di sincronizzazione nel momento in cui l'iterazione termina ed il team di sviluppo mostra al cliente un prodotto parziale ma funzionante. L'iterazione costringe a prendere decisioni in termini di valutazione del lavoro realizzato dal team, in termini di rilascio in produzione della nuova funzionalità sviluppata, in termini di nuova valutazione delle priorità dei requisiti.

- **Sincronizzazione (Synchronitation):** ogni qualvolta diverse persone lavorano sulla stessa cosa, sorge il bisogno di verificare se il lavoro è sincronizzato.
  - Una verifica quotidiana all'interno del team
  - Una integrazione settimanale tra più team di sviluppo

- **Sviluppo Set-Based (Set-based developments):** nello sviluppo Set-based, la comunicazione è orientata a diffondere i vincoli,
  Di fronte ad una problematica di difficile soluzione un buon approccio potrebbe essere quello di sviluppare una serie di soluzioni alternative al problema e vedere quanto bene queste funzionino veramente per poi scegliere tra una delle alternative o unire le caratteristiche migliori delle varie alternative.

3. *Terzo principio: decidere il più tardi possibile (Decide as Late as Possible)*

   Mantenere aperte diverse opzioni, purché praticabili, sino al momento di prendere una determinata decisione.

4. *Quarto principio: consegnare il più velocemente possibile (Deliver as Fast as Possible)*

   Consegnare il più velocemente possibile, non significa sbrigarsi e fare un lavoro approssimativo e ricco di difetti, ma rilasciare e trasferire valore al cliente non appena parte una richiesta. La possibilità di consegnare rapidamente un artefatto ha un duplice vantaggio:
   - Un vantaggio per il cliente che spesso si traduce in maggiore opportunità di business nel rilasciare sul mercato, ad esempio, un elemento che contraddistingue il proprio prodotto;
   - Un' opportunità per il fornitore in quanto i clienti potrebbero cambiare idea.

Molte organizzazioni utilizzano un sistema *Pull* utilizzando le previsioni di lavoro per determinarne un piano. Questo però creerebbe degli sprechi. Un approccio Lean tende di attivare un processo di produzione solo in base a quelle che sono le esigenze del cliente. Il modello Kanban aiuta a trasformare il processo in continuo valore.

Questo modello applicato allo sviluppo di software può essere così rappresentato.

***Figura 14** - Il Kanban nello sviluppo software*

5. *Quinto principio: dare forza al team (Empower the Team)*

Il quinto principio Lean non significa abbandonare la leadership ma permettere alle persone che aggiungono valore, quelle che di solito il lavoro lo svolgono, di

usare il loro pieno potenziale. Per garantire ciò, devono aumentare i fattori che contribuiscono alla soddisfazione sul lavoro come l'autorealizzazione, il riconoscimento e la responsabilità.

6. Sesto principio: costruire nell'integrità (Build Integrity In)

Il sesto principio Lean non è orientato a perorare la causa che debba essere concepito e realizzato un design articolato e completo in anticipo, ma piuttosto propone di mantenere il design più semplice possibile lasciando aperta la possibilità di poter apportare modifiche, anche in corso d'opera, a patto di non perdere di vista l'integrità dell'intero sistema.

7. Settimo principio: guarda l'Intero (See the Whole)

Non significa ignorare i dettagli ma porre attenzione nel non cadere nella tentazione di ottimizzare alcune parti a discapito del tutto.
Una delle tecniche che viene utilizzata è quella chiamata come Five Whys: per affrontare la causa principale si continua a chiedere il perché e non ci si ferma quando si trova la prima ragionevole risposta dal momento che può essere stato ravvisato solo un sintomo.

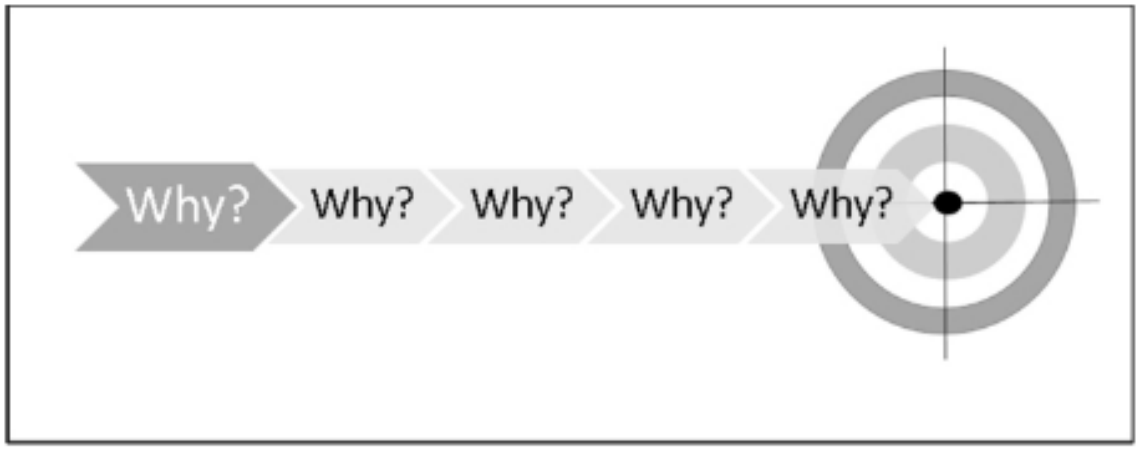


Figura 15 *- Five Whys Approach*

(Corbucci, Seconda Edizione)

6 EXTREME PROGRAMMING (KENT BECK 1999)

Agile *Extreme Programming*, propone un vero e proprio framework per la realizzazione di un prodotto software, concepito esclusivamente in questo contesto.
Attraverso il suo framework e le sue pratiche, eXtreme Programming consente di coinvolgere direttamente il cliente, di rispondere ragionevolmente ai cambiamenti nei requisiti, anche a quelli tardivi e di sostenere e supportare il lavoro di gruppo.
Il primo progetto su cui molte delle tecniche dell'XP si sono forgiate, è stato un progetto per Chrysler nel quale una dozzina di sviluppatori hanno riscritto in due anni tutto il sistema paghe e stipendi per impiegati, dirigenti e consulenti esterni della casa automobilistica americana (circa 90.000 persone, con un'enorme diversificazione di profili stipendiali).
Si tratta di un progetto il cui costo è stato inferiore di 1-2 ordini di grandezza rispetto a quello prevedibile usando un approccio tradizionale.
La programmazione estrema è stata ideata da *Kent Beck e Ward Cunningham* (entrambi cofirmatari del Manifesto dell'Agile).
Molto simile a Scrum, in Extreme Programming il progetto si divide in blocchi rapidi di lavoro (le iterazioni).
Al completamento di ciascuno di essi, viene consegnata al cliente una versione funzionante del prodotto.
Questi blocchi vengono scelti ed indicati direttamente dall'unico responsabile del prodotto (Un *Product Owner* o responsabile del business) con l'aiuto del team di sviluppo e rappresenta tutto il lavoro che deve essere svolto dal team (lo scope nell'approccio tradizionale di *Project Management*).
Inoltre, vengono organizzate riunioni giornaliere dal team di sviluppo (*Daily Stand Up Meeting*) per verificare che cosa si sia fatto (obiettivo di consuntivazione) e cosa si farà (obiettivo di previsione).
Obiettivi di questi incontri giornalieri è verificare se il team sta andando verso la giusta direzione.
XP, come metodologia di Agile Project Management, acquisisce i commenti del cliente nel corso dell'intero progetto in modo da poter incorporare suggerimenti e nuove informazioni durante lo sviluppo del prodotto.
Tutta la metodologia XP, ed in modo particolare le pratiche di XP, si basa sulla teoria del processo di apprendimento detta **SHU-HA-RI,** tre stadi che devono essere applicati in questo ordine:

- **SHU:** in questa fase iniziale lo studente segue gli insegnamenti di un maestro, con precisione.
- **HA:** lo studente inizia ad espandersi, inizia ad imparare i principi fondamentali e la teoria dietro la tecnica
- **RI:** lo studente imparerà solo dalla sua stessa pratica

Extreme Programming si ispira a questa teoria, applicandola nel seguente modo:

- Livello 0: applicare i principi e le pratiche - SHU
- Livello 1: integrare pratiche con altre pratiche - HA
- Livello 2: creare nuove pratiche e nuovi principi purché si ispirino a certi valori – RI
  -

(Corbucci, Seconda Edizione)

6.1 LE PRATICHE

Le pratiche, suddivise in pratiche primarie e secondarie (corollari), servono per implementare i valori attraverso i principi di XP.
Queste rappresentano delle vere e proprie Best Practice che i team Agile utilizzano nella gestione progettuale.
Le dodici pratiche primarie sono le seguenti:

- **The Planning Process (o Planning Game):** La pianificazione del processo in XP consiste nella definizione del business Values di ogni caratteristica desiderata e nella stima dei costi per decidere in che ordine le caratteristiche debbano essere realizzate. Si tratta quindi di una strategia estremamente duttile, che può essere ripensata durante lo svolgimento del progetto.

- **Small Release:** La vita e lo sviluppo dell'applicazione sono scanditi dai rilasci di versioni del prodotto funzionanti. Ogni rilascio rappresenta il punto conclusivo di un'iterazione di sviluppo e l'inizio di una nuova pianificazione. Per poter tener conto di cambi di prospettiva, errori di valutazione, nuovi requisiti, restrizioni di bilancio, ogni iterazione dovrebbe durare non più di qualche settimana (in genere, da due a quattro

- **Metaphor:** I team di lavoro XP condividono un "sistema dei nomi", nel quale siano compresi I concetti fondamentali relativi al software sotto sviluppo (in pratica il dominio dei dati)

- **Simple Design**: Un programma costruito con XP dovrebbe essere il programma più semplice in grado di soddisfare i requisiti. Non si cercano soluzioni che possano portare benefici in un prossimo futuro. La qualità del progetto è comunque tenuta alta, grazie anche alle attività di re factoring.

- **Testing**: è un elemento fondamentale dell'XP. I programmatori devono scrivere i test contemporaneamente alla scrittura del programma stesso (strumenti come XUnit sono particolarmente utili). Il testing di accettazione invece è demandato al cliente finale

- **Re factoring**: durante le fasi di integrazione del codice appena prodotto con il resto del sistema, bisogna operare in maniera da tenere alta la qualità della progettazione, evitando ridondanze dei dati e duplicazioni del codice. Fondamentale rimane ancora la comunicazione tra I vari gruppi.

- **Pair Programming: i** programmatori XP lavorano in coppia: due programmatori sulla stessa macchina. Uno dei programmatori è addetto alla scrittura fisica del codice, mentre l'altro ne controlla la correttezza, salvo scambiarsi i ruoli di tanto in tanto. Esperimenti dimostrano come la produttività sia superiore rispetto a quella di due programmatori separati.

- **Collective Ownership**: ogni programmatore deve essere in grado di avere un'idea generale del progetto e di poter toccare qualunque punto del codice, anche di quello che non ha scritto lui personalmente. Quest'obiettivo può essere raggiunto seguendo il principio di Simple Design

- **Continuous Integration**: il processo di integrazione del software con i nuovi moduli avviene più volte al giorno. Ciò aiuta I programmatori ad essere sempre aggiornati sullo stato del progetto e riduce la maggior parte dei gravi problemi di integrazione che si verificano quando questa fase non è eseguita così spesso.

- **40-hour Week**: i programmatori stanchi commettono più errori. Essi non dovrebbero fare troppo straordinario, in modo da essere freschi, in salute e produttivi.

- **On-site Customer**: il committente deve essere coinvolto nello sviluppo perché è l'unica fondamentale fonte di convalida del sistema. Partecipa perciò alla stesura dei test di sistema e verifica, periodicamente, che il sistema realizzato corrisponda effettivamente alle proprie esigenze. Inoltre, è la principale fonte di informazione per la conoscenza sul dominio di applicazione.

- **Coding Standard:** i programmatori dovrebbero condividere uno stesso stile di scrittura del codice, in modo da rendere più semplice la comunicazione

(Corbucci, Seconda Edizione)

6.2 CICLO DI VITA XP

Il ciclo di vita tecnico di XP è il seguente:

- *Primo passo: Architectural Spike*

Il primo passo consiste nel mettere insieme cliente e sviluppatori per capire non solo quello che deve essere fatto ma anche come farlo. *L'Architectural Spike* si ispira al valore della semplicità e l'architettura del sistema sulla base dei requisiti utente. Il team non darà tutta la soluzione architetturale ma fornirà al cliente una prima idea sulla quale lavorare aspettando che l'informazione con il tempo, come un iceberg, emerga iterazione dopo iterazione.

La pratica che il team Agile sfrutta è quella del Simple Design, ossia si cerca di trovare un design nell'architettura che sia il più flessibile e aggiornabile possibile. Parallelamente alla definizione del picco architetturale, il team ha bisogno di comprendere l'ambito del progetto in termini di requisiti e funzionalità: vengono infatti prodotte le User Stories (artefatto proprietario della metodologia XP).

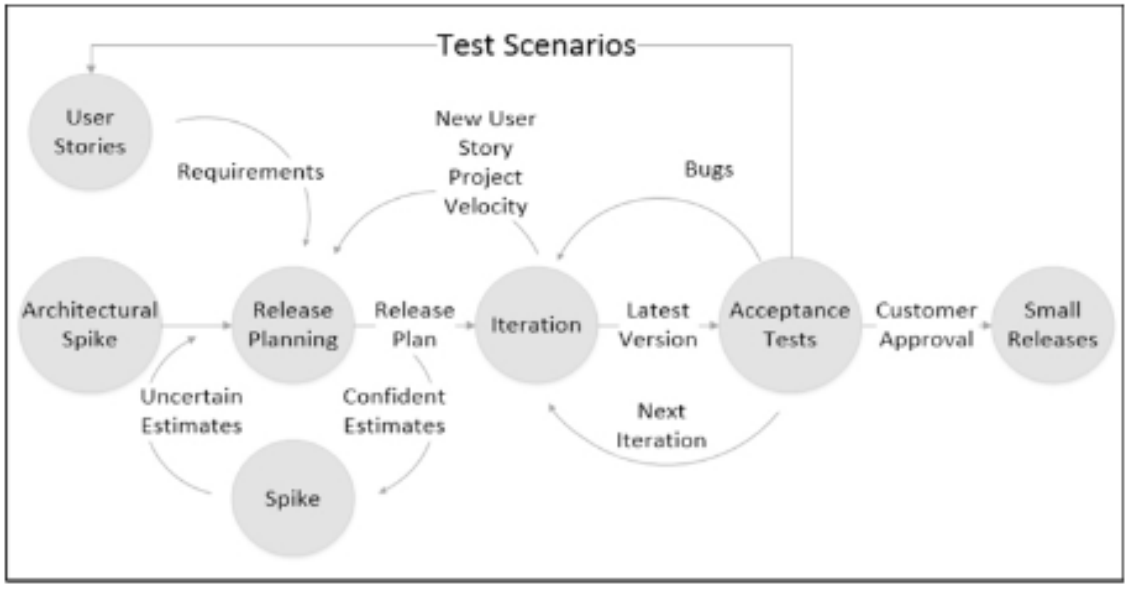


Figura 16 *- Ciclo di vita tecnico XP*

- *User Story*

La User Story rappresenta una funzionalità, o parte di essa, scritta dall'utente in un linguaggio naturale (quindi discorsivo) che consente agli sviluppatori di comprendere i bisogni del cliente.
Una User Story è generalmente composta da:

- Una descrizione scritta che verrà utilizzata per la pianificazione o come promemoria (*Card*);
- Una conversazione tra il team di sviluppo ed il cliente che servirà a far emergere ulteriori dettagli (*Conversation*);
- Il test di accettazione che valida la funzionalità e determina la completezza della User Story (*Confirmation*).

La User Story cerca di colmare quel naturale divario, spesso un baratro, tra i desiderata del cliente e quanto lo sviluppatore percepisce. Da un lato il cliente ha in mente un prodotto con specifiche funzionalità mentre dall'altro lo sviluppatore interpreta il messaggio del cliente proponendo soluzioni tecniche incomprensibili al cliente stesso. L'analisi dei requisiti diviene una crociata ed il documento finale che viene redatto viene a volte disconosciuto da una delle due parti in causa.

- *Story Point*

Lo Story Point Man mano che le User Stories vengono redatte, il team di sviluppo è chiamato a valutarle singolarmente. Lo Story Point rappresenta l'unità di misura per esprimere la grandezza di una User Story. La grandezza deve comprendere le attività di analisi, design, codifica e test (la User Story deve essere completata "*done done*" al termine dell'iterazione). Lo Story Point non indica la quantità di tempo necessaria per completare una funzionalità ma una misurazione (astratta) della grandezza relativa di una funzionalità rispetto alle altre.

- *Project Velocity*

La Project Velocity La velocity rappresenta una metrica utilizzata nell'Agile per valutare le performance del team di sviluppo. Questa rappresenta la quantità di Story Points che viene rilasciata al termine di ogni iterazione. Traslando il concetto di velocità nell'Agile, lo spazio è rappresentato dalla distanza che il team deve ricoprire per raggiungere un obiettivo e questo parametro può essere tradotto come sommatoria degli Story Points che il team deve rilasciare al termine della release.

- *Secondo passo: Release Plan*

Dopo aver definito un picco architetturale e dopo aver acquisito un numero congruo di User Stories, il team elabora un piano di rilascio delle funzionalità previste per la release sulla base delle priorità concertate con il Product Owner. Il Release Planning Meeting è un meeting che ha l'obiettivo di definire un piano di rilascio delle funzionalità che verrà schematizzato ed ufficializzato con un Release Plan.

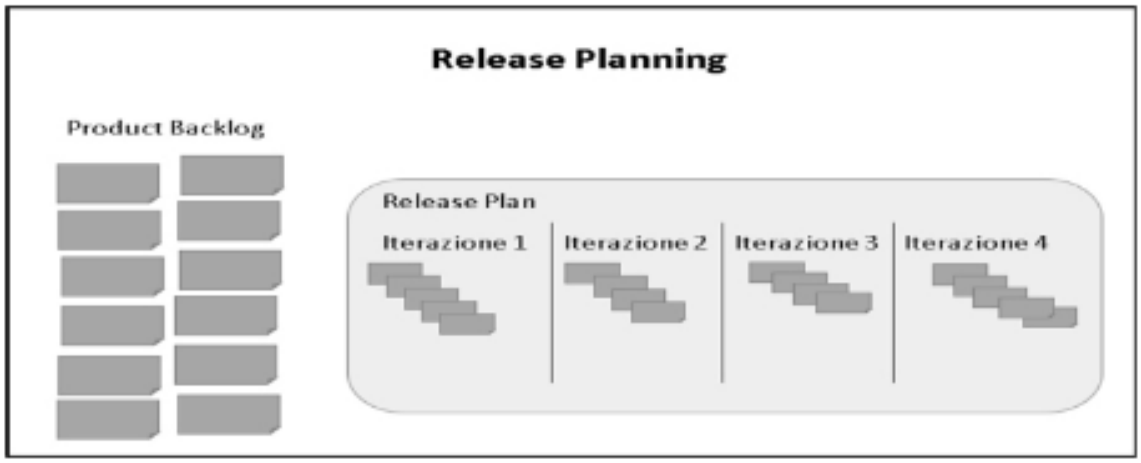


Figura 17 *- Release Plan*

Due sono i momenti principali per lo sviluppo di un piano di rilascio:

- **Planning Game:** che può essere definito come la tecnica che consente al team, partendo dal Backlog, di definire il Release Plan.
- **Planning Poker**: rappresenta una tecnica Agile per la stima e la pianificazione basata fortemente sul consenso dei partecipanti. Mentre nel planning game il Product Owner, la persona di business, è fortemente coinvolta definendo le priorità per ciascuna delle User Stories, nel planning poker i fari sono puntati quasi esclusivamente sul team di sviluppo, anche se è consentita la presenza del Product Owner.

➢ *Terzo passo: L'iterazione*

In XP si consigliano iterazioni che abbiano una durata compresa dalle due alle quattro settimane. Come i progetti, le iterazioni devono realizzare qualcosa e quindi deve essere definito ciò che si va a progettare, codificare, implementare, testare e rilasciare in produzione. Il team lavora per il tempo allocato sull'iterazione senza interruzioni. Alla fine dell'iterazione il team rilascia funzionalità testate, secondo gli scenari di test definiti nelle User Stories, che vengono riviste e accettate dal committente. È' possibile inoltre determinare il numero di User Stories completate e di conseguenza la velocità.
Anche se XP è una metodologia incentrata sul codice anche la pianificazione risulta essere molto importante.
La pianificazione però non è il piano.
SE i lunghi cicli di sviluppo, previsti nei metodi tradizionali, non sono in grado di far fronte al cambiamento, XP cerca di introdurre questo miglioramento rendendo cicli di sviluppi più corti.

(Corbucci, Seconda Edizione)

7 DEVOPS

7.1 DEFINIZIONE

Per raggiungere quelli che sono i principi descritti nella metodologia Agile è necessaria una nuova etica denominata DevOps.
Il termine DevOps è composto da due parti: *developments*, tutte le attività di sviluppo di un prodotto o servizio, e *operation*, tutte le attività di messa in opera e mantenimento in vita degli stessi.
L'obiettivo su cui la maggioranza dei filoni di interpretazione DevOps concorda, è l'ambizione di *aumentare la qualità* e *ridurre il time-to-market*.

La definizione che più rispecchia l'intento iniziale è quella del ThoughtWorks Radar 2010:

DevOps è un nuovo movimento che cerca di realizzare le esigenze di business per rapidità di consegna dei prodotti software, mantenendo stabili gli ambienti già in vita. Usa due approcci: il primo è la promozione di una stretta collaborazione tra programmatori e sistemisti; il secondo applica le pratiche dello sviluppo Agili di software (collaborazione, automazione, semplicità e così via) ai processi di operation, come l'approvvigionamento, la gestione dei cambiamenti e il monitoraggio della produzione. Comprende cultura, processi e strumenti - tutti al supporto di migliorare la comunicazione con il feedback e ottenere tempi di consegna e risultati più prevedibili.

(DevOps - Guida per integrare Development e Operations e produrre software di qualità)

Per DevOps si intende una cultura, un movimento, una filosofia, un insieme di pratiche e l'atto di utilizzare strumenti (software) per automatizzare e migliorare i metodi di gestione di sistemi complessi.

L'esigenza di applicare DevOps esiste perché lo sviluppo di software è un Asset strategico per le organizzazioni e le imprese di tutte le dimensioni nella moderna economia globale.
Progettando, costruendo, testando, implementando, gestendo applicazioni e sistemi in modo più rapido e affidabile, si cerca di creare valore per i clienti e promuovere un flusso di lavoro gestibile rispetto al prodotto.
Infine, gli obiettivi specificati nella definizione non limitano l'ambito delle pratiche DevOps ai test e alla distribuzione. Al fine di raggiungere tali obiettivi, è importante includere un modello operativo nella raccolta dei requisiti, ovvero molto prima dello sviluppo. Analogamente, la definizione non significa che le pratiche DevOps terminano con il rilascio in produzione; l'obiettivo è garantire l'alta qualità del sistema distribuito durante tutto il suo ciclo di vita. Pertanto, devono essere incluse anche le pratiche di monitoraggio che aiutano a raggiungere tali obiettivi.

7.2 METODOLOGIA DI RILASCIO TRADIZIONALE

Soffermandoci sulla definizione di DevOps, in origine il team di sviluppo e il team operazionale lavoravano in modo completamente distaccato. Test e implementazione erano attività isolate che venivano eseguite solo dopo la progettazione e quindi consumavano più tempo rispetto ai cicli attuali di *build*.

Lo scenario progettuale, senza l'impiego di DevOps, è il seguente:

- I *team di sviluppo* e il *team operazionale* hanno tempistiche distinte e non sincronizzate causando ulteriori ritardi. Nello specifico Developer e Operations hanno i loro processi, strumenti e basi di conoscenza indipendenti. L'interfaccia tra loro è di solito un sistema di ticket.
  I team di sviluppo richiedono l'implementazione di nuove versioni del software e il personale operativo gestisce manualmente il ticket. In questo accordo però, i team di sviluppo cercano continuamente di spingere le nuove versioni in produzione, mentre il personale operativo tenta di bloccare queste modifiche per mantenere la stabilità del software. Teoricamente, questo modus operandi offre una maggiore stabilità per il software in produzione però in pratica comporta lunghi ritardi tra gli aggiornamenti del codice e la distribuzione, oltre a processi di risoluzione dei problemi inefficaci.

- I membri del team impiegano molto tempo a testare, distribuire e progettare invece di costruire il progetto.

- Nessuna pratica di programmazione estrema comprende la distribuzione. Il passaggio alla produzione tende ad essere un processo stressante nelle organizzazioni, infatti vengono svolte attività manuali soggette a errori e, anche, correzioni dell'ultimo minuto.

7.3 SOFTWARE DEVELOPMENT LIFE CYCLE

Prima di entrare nel dettaglio della metodologia, soffermiamoci su cosa si intende per ciclo di vita del rilascio.
Per *Software Development Life Cycle* si intende un processo continuo, che inizia nel momento in cui viene presa una decisione per iniziare un progetto, e finisce nel momento in cui il progetto è stato realizzato.

Figura 18 *- The Software Development Life Cycle*

Le principali fasi sono le seguenti:

1. **Concepimento - pianificazione e analisi dei requisiti**

La prima fase fa riferimento alla Raccolta dei requisiti ovvero alla raccolta e analisi degli input, i desiderata del cliente.
Queste informazioni vengono utilizzate per pianificare l'approccio del progetto e per condurre studi di fattibilità del prodotto.

2. **Progettazione – Design**

In questa fase viene progettata l'architettura. Si tratta di una fase con focus su aspetti tecnici, discusse con il cliente. Vengono definite le tecnologie utilizzate, il carico di lavoro, i limiti di tempi e il budget che vengono calcolati ed adottati in funzione del progetto. Inoltre, tale fase, consiste nel definire l'esperienza utente dell'app (Ad esempio il layout e il funzionamento) e nel trasformarla nella progettazione di una interfaccia utente, in genere con l'aiuto di un Graphic Design.

2. **Implementazione**

I developers seguono le linee guida date dall'organizzazione, programmando e produco il software richiesto attraverso appositi strumenti.
Questa fase al suo interno include 4 fasi: la scelta dell'algoritmo di sviluppo, la scrittura, la compilazione e il debug del codice prodotto.

4. **Test**

Viene testato il funzionamento del prodotto generato allo step precedente e vengono applicate delle eventuali correzioni affinché il prodotto soddisfi i requisiti richiesti.

5. **Rilascio e manutenzione**:

Una volta che il prodotto è stato testato, viene rilasciato. Eventuali bug non emersi nella fase di test verranno poi risolti a posteriori.

(Software Development Life Cycle (SDLC))

7.4 DEVOPS E IL CICLO DI VITA DELL'APPLICAZIONE

DevOps influenza il ciclo di vita dell'applicazione nelle fasi di *pianificazione, sviluppo, distribuzione e operatività.*

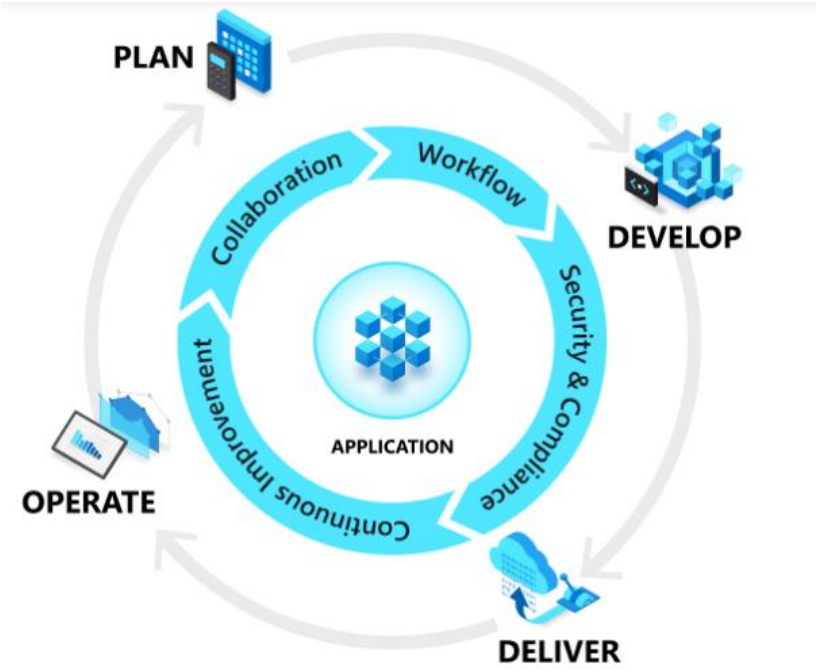


Figura 19 *- DevOps e il ciclo di vita dell'applicazione*

1. **Piano:** durante la fase di pianificazione, i team DevOps concepiscono, definiscono e descrivono le funzionalità delle applicazioni e dei sistemi da creare.

2. **Sviluppo:** la fase di sviluppo include tutti gli aspetti della codifica, tra cui scrittura, test, revisione e integrazione del codice da parte dei membri del team, oltre all'inserimento del codice in artefatti della compilazione che possono essere distribuiti in diversi ambienti. I team DevOps si impegnano per innovare rapidamente senza sacrificare la qualità, la stabilità e la produttività. A tale scopo usano strumenti a produttività elevata, automatizzano i passaggi ripetitivi e manuali ed eseguono l'iterazione in piccoli incrementi tramite test automatizzati e integrazione continua.

3. **Distribuzione**: Il recapito è il processo di distribuzione di applicazioni negli ambienti di produzione in modo coerente e affidabile. La fase di recapito include anche la distribuzione e la configurazione dell'infrastruttura di base completamente regolamentata che costituisce tali ambienti.

4. **Operazioni:** La fase operativa prevede la manutenzione, il monitoraggio e la risoluzione dei problemi delle operazioni negli ambienti di produzione. Durante l'adozione delle procedure DevOps i team si impegnano per assicurare l'affidabilità del sistema e la disponibilità elevata e cercano di ridurre a zero il tempo di inattività, rafforzando al tempo stesso la sicurezza e la Governance. I team DevOps cercano di identificare i problemi prima che influiscano sull'esperienza dei clienti e di attenuare rapidamente i problemi quando si verificano. Questo livello di vigilanza richiede telemetria avanzata, avvisi di utilità pratica e visibilità completa delle applicazioni e del sistema sottostante.

   (Microsoft Azure)

7.5 CULTURA DEVOPS

Benché l'adozione delle procedure DevOps automatizzi e ottimizzi i processi tramite la tecnologia, tutto è basato sulla cultura interna dell'organizzazione e sulle persone che contribuiscono al lavoro. Per stabilire una cultura DevOps sono necessarie profonde modifiche al modo in cui per persone lavorano e collaborano. Quando le organizzazioni scelgono di adottare una cultura DevOps possono tuttavia creare l'ambiente per lo sviluppo di team a prestazioni elevate.

- **Collaborazione, visibilità e allineamento**

Uno degli aspetti fondamentali di una cultura DevOps ottimale è costituito dalla collaborazione tra team, che inizia dalla visibilità. Diversi team, ad esempio quelli addetti a sviluppo e operazioni IT, devono condividere i rispettivi processi DevOps, le priorità e le preoccupazioni. I team devono inoltre pianificare insieme il lavoro e allineare obiettivi e indicatori di successo correlati al business.

- **Cambiamenti a livello di ambito e responsabilità**

Grazie all'allineamento, i team acquisiscono la proprietà e vengono coinvolti in fasi aggiuntive del ciclo di vita, non solo nelle fasi centrali per i rispettivi ruoli. Gli sviluppatori diventano ad esempio responsabili non solo dell'innovazione e della qualità della fase di sviluppo, ma anche delle prestazioni e della stabilità offerte dalle loro modifiche nella fase operativa. Al tempo stesso i responsabili delle

operazioni IT devono includere Governance, sicurezza e conformità nella fase di pianificazione e sviluppo.

- **Cicli di rilascio più brevi**

I team DevOps rimangono flessibili rilasciando software in brevi cicli. I cicli di rilascio più brevi semplificano la pianificazione e la gestione dei rischi perché il progresso è incrementale e ciò riduce anche l'impatto sulla stabilità del sistema. La riduzione del ciclo di rilascio permette inoltre alle organizzazioni di adattarsi e reagire all'evoluzione delle esigenze dei clienti e alla pressione della competizione.

- **Apprendimento continuo**

I team DevOps a prestazioni elevate si concentrano sulla crescita. Falliscono e rispondono immediatamente agli errori e incorporano le lezioni apprese nei processi, migliorando continuamente, incrementando la soddisfazione dei clienti e accelerando l'innovazione e l'adattabilità al mercato. DevOps è un percorso e favorisce quindi una crescita continua.

(DevOps - Guida per integrare Development e Operations e produrre software di qualità)

7.6 DEVOPS: MICRO SERVIZI E L'EVOLUZIONE DELLA INFRASTRUTTURA

L'implementazione efficace di strumenti e procedure DevOps per testare, distribuire, monitorare e modificare il codice e i sistemi complessi di aziende, spesso richiede un'architettura software nota come *micro servizi*.

I *micro servizi* sono un approccio a livello di architettura per la creazione di applicazioni in cui ogni funzione di base o servizio viene creato e distribuito in modo indipendente. L'architettura di micro servizi è distribuita e ad accoppiamento debole, in modo che l'errore di un componente non comprometta l'intera app. I componenti indipendenti interagiscono e comunicano con contratti API ben definiti.
L'infrastruttura basata su Cloud consente l'architettura dei micro servizi

L'infrastruttura è tutto ciò che è necessario per gestire il ciclo di vita di un'applicazione software. Fino a circa vent'anni fa in questo campo il ruolo dell'automazione era molto ridotto. Oggi l'approccio DevOps è dell'avviso opposto: eliminare le operazioni manuali, puntare all'automazione in primo luogo, ma anche poter creare qualcosa di autonomo grazie alla tecnologia.

Nel 2000 e negli anni precedenti si parlava di *bare metal* (i server fisici). Il mondo non era ancora virtuale e automatico, anzi era quasi esclusivamente manuale. Quando un manager decideva di dare il via a un nuovo progetto, iniziava uno studio di requisiti di capacità. Ottenuta la copertura economica, si doveva scegliere un

fornitore per l'hardware e acquistare delle macchine fisiche, che venivano consegnate nel giro di qualche giorno o settimana e che dovevano essere installate e configurate. Per questo motivo sovente la potenza di calcolo acquistata era quasi sempre sovrastimata e in eccesso, perché doveva corrispondere ai picchi di utilizzo stimati in fase di analisi; questo sottoutilizzo in stato normale rappresentava già di per sé uno spreco di capitale.

Il 2001 è l'anno in cui sul mercato arriva la *virtualizzazione*. VMWare cerca di far presa proprio su quella disponibilità di capacità di calcolo inutilizzata; inoltre con la virtualizzazione è possibile rendere efficiente l'utilizzo dell'infrastruttura dividendo in unità logiche la potenza di calcolo di una singola macchina. In un server fisico vengono simulati (emulati) più server logici, che sono virtuali. Non è più necessario aspettare quindi che il corriere consegni un nuovo server fisico per ottenerne uno.

Nel 2006 Amazon presenta *Amazon Web Services*, il primo esempio su larga scala di *Infrastructure-as-a-Service* (IaaS, infrastruttura come servizio), che successivamente sarà definito dal marketing come *Cloud*. La novità in questo approccio riguarda l'opportunità di poter beneficiare della virtualizzazione e l'automazione insieme: ora con comandi e protocolli si possono creare e distruggere macchine virtuali, elementi di storage e intere reti. Non è più necessario passare per richieste asincrone al gestore dell'infrastruttura. A consuntivo, un algoritmo calcola il costo di utilizzo delle risorse.

Il 2009 è l'anno della rivoluzione successiva, quando si diffonde il termine *Platform-as-a-Service* (PaaS, piattaforma come servizio). La novità del PaaS è legata alla magia di poter inviare il proprio codice al sistema con un solo comando e vederlo in produzione pochi minuti dopo, senza doversi occuparsi di complicati strumenti di rilascio.

(Sviluppo Software, 2020)

Le tendenze per il 2021 e oltre

L'emergenza associata al lockdown ha massimizzato l'apertura delle aziende al cloud in ogni sua forma. Nel momento in cui lo Smart Working è diventato un imperativo categorico, tutte le organizzazioni hanno rivisto le politiche di gestione. Per garantire la continuità operativa dei telelavoratori da casa è stato necessario riorganizzare infrastrutture e applicazioni. E questo ha portato a un'accelerazione delle strategie in cloud e di nuove soluzioni a supporto del lavoro da remoto (Zanotti, 2020)

7.7 APPLICAZIONE DI DEVOPS

Oltre a stabilire una cultura DevOps, i team applicano l'approccio DevOps implementando alcune procedure nell'intero ciclo di vita dell'applicazione. Alcune procedure contribuiscono all'accelerazione, all'automazione e al miglioramento di una fase specifica. Altre procedure sono relative a più fasi e aiutano i team a creare processi semplici che contribuiscono al miglioramento della produttività.

Continuous Software Development

Con il termine Continuous Software Development (CSD) ci si riferisce ad una serie di tecniche che consentono di sviluppare un applicativo software in maniera iterativa. Le funzionalità sviluppate, così come le modifiche, diventano immediatamente parte di un prodotto software.

Figura 20 *- Continuos Software Development*

Continuous Integration (CI)

La *Continuous Integration* (definita in italiano anche "integrazione continua") è una tecnica di *sviluppo agile di software*. Con questo metodo di integrazione gli sviluppatori integrano porzioni di codice finiti nell'applicazione anche più volte al giorno, piuttosto che integrarle tutte soltanto alla fine del progetto (Digital Guide IONOS, 2019).
Uno dei principali vantaggi dell'adozione della CI è che consente di risparmiare tempo, durante il ciclo di sviluppo, identificando e risolvendo tempestivamente i conflitti.
È anche un ottimo modo per ridurre la quantità di tempo speso per la correzione di bug e regressione mettendo più enfasi sull'avere una buona suite di test. Infine, aiuta a condividere una migliore comprensione della code base e delle funzionalità che stai sviluppando per i clienti.
L'obiettivo di questo moderno metodo è quello dunque di suddividere il lavoro in porzioni più piccole, per rendere il processo stesso di sviluppo più efficiente e poter reagire con maggiore flessibilità alle modifiche.

Gli sviluppatori caricano il proprio codice ultimato una o più volte al giorno sulla *mainline*, ovvero il codice sorgente a cui hanno accesso tutti i programmatori. Dal momento che si tratta di porzioni di codice relativamente ridotte, anche l'integrazione richiede meno tempo. Nel caso in cui venga individuato un errore è possibile identificarlo e risolverlo in modo relativamente veloce.
Bisogna ricordare, tuttavia, che lo sviluppatore non lavora da solo sul programma. Mentre lui ha apportato le sue modifiche, molto probabilmente i suoi colleghi avranno svolto altri compiti, quindi ciascun sviluppatore del team possiede una versione differente sul proprio computer. Anche la versione della *mainline* sarà sicuramente cambiata nel frattempo, fatto che costringe il programmatore a

integrare tutte le modifiche innanzitutto nella sua *Working Copy*. Se a questo punto emerge un errore, dovrà risolverlo.
Solo allora il programmatore potrà aggiungere le sue modifiche alla mainline e testare nuovamente il programma. Se non emergono altri errori, che possono presentarsi nel caso in cui non abbia aggiornato correttamente la sua Working Copy, la procedura è conclusa e lo sviluppatore può dedicarsi al compito successivo.

Una sola sorgente

Tutti i membri del team devono utilizzare la stessa sorgente (lo stesso repository) quando lavorano sul codice. Lo stesso per quanto riguarda la banca dati.

Build automatizzati

Per creare un programma funzionante dal codice sorgente è necessario compilarlo, attualizzare le banche dati e spostare i file nelle posizioni corrette. Questo processo può essere automatizzato. Idealmente dovrebbe essere possibile eseguire un processo di build con un solo comando.

Sistemi di testing automatico

L'integrazione di meccanismi di testing nel processo di build consente al team di automatizzare, e dunque velocizzare ulteriormente, la Continuous Integration. Proprio come il processo di build, anche i test devono poter essere eseguiti nel modo più efficiente possibile (la cosa migliore è implementare una serie di meccanismi di controllo diversi), per ogni modifica apportata al repository principale.
Questo consente di rilevare eventuali errori in anticipo, riducendo al minimo le interruzioni del team. Il testing è l'elemento centrale del metodo agile **Test Driven Development (TDD).**

Integrazione giornaliera

L'integrazione continua può funzionare solo se tutti i membri del team condividono il sistema. È sufficiente che un membro del team non integri costantemente il suo codice nella mainline perché tutti gli altri partano da presupposti sbagliati. La comunicazione gioca un ruolo fondamentale, infatti se tutti gli sviluppatori restano aggiornati sul lavoro degli altri, molte difficoltà possono essere prevenute.

Mainline funzionante

Il codice di programmazione contenuto nella *mainline* deve essere sempre testato e funzionante. In tal senso ogni sviluppatore deve inoltre verificare che il suo codice sia funzionante quindi, dopo l'integrazione, dovrà testare nuovamente il codice insieme al build. Se vengono riscontrati degli errori dovrà risolverli. In questo modo viene garantito che il codice è privo di difetti.

Tempi brevi di integrazione

La vera e propria integrazione del codice (incluso il testing) deve avvenire piuttosto velocemente. L'Extreme Programming (XP) prevede appena 10 minuti per questo compito. Considerando che uno sviluppatore effettua in genere più di un'integrazione al giorno, tempi di build più lunghi provocherebbero delle perdite di tempo consistenti.

Ambiente di prova copiato

Si consiglia generalmente di eseguire i test in un ambiente esterno e sicuro, che deve comunque essere configurato esattamente come l'ambiente di produzione. Questo può talvolta diventare abbastanza dispendioso, dal momento che la macchina deve essere configurata in maniera identica. La virtualizzazione di interi computer consente tuttavia di ridurre sempre di più questi costi.

Accesso semplice

Tutti i collaboratori devono poter accedere facilmente all'ultima versione ed essere in grado di eseguire il programma.

Comunicazione chiara e comprensibile

Sapere chi ha introdotto quali modifiche è tanto importante quanto garantire l'accesso di tutti i collaboratori al codice sorgente e al file eseguibile. Come parte integrante della comunicazione, è fondamentale che gli sviluppatori si informino a vicenda su chi sta lavorando su quale processo di build. A tale scopo alcuni team sfruttano display separati o raffigurazioni visive che evidenzino che si è attualmente impegnati con un’integrazione.

Suddivisione automatizzata

Infine è fondamentale automatizzare la suddivisione del software. I file devono essere trasferiti da un ambiente all'altro, passaggio che può richiedere molto tempo. Per questo è consigliato l'utilizzo degli script che automatizzano e velocizzano questo processo.

Vantaggi e svantaggi della Continuous Integration

Nonostante le sue caratteristiche positive, nel lavoro quotidiano la Continuous Integration non offre solo dei vantaggi. Infatti, è vero che consente di evitare un'ampia e prolungata fase di integrazione alla fine del processo e permette di risolvere i problemi tempestivamente, ma per un team ben coordinato l'adattamento alla Continuous Integration può risultare difficile. In questi casi il metodo può persino provocare l'effetto contrario e risultare più dispendioso in termini di tempo.

(Digital Guide IONOS, 2019)

Continuous Delivery (CI)

La Continuous Delivery (in italiano “consegna continua”) è un concetto innovativo dello sviluppo dei software sempre più popolare. Secondo la Continuous Delivery le fasi di produzione, sviluppo, controllo qualità e rilascio non sono fini a sé stesse, ma si ripetono continuamente e in automatico nel processo di sviluppo all’interno della cosiddetta pipeline di Continuous Delivery. Il vantaggio è che i prodotti software vengono sottoposti al controllo qualità a intervalli regolari e possono già essere rilasciati mentre il prodotto è ancora in lavorazione. Nella pipeline si ricevono quindi costantemente dei feedback grazie ai quali è possibile apportare qualsiasi modifica al codice sorgente e migliorare così il software in maniera mirata. (Digital Guide IONOS, 2019)

La Continuous Delivery è un modello introdotto nello sviluppo di software per eseguire contemporaneamente le fasi di sviluppo, rilascio, feedback e gestione della qualità in brevi intervalli e in un ciclo continuo. Lo sviluppo in questo modo diventa più efficiente e il cliente riceve prima il prodotto, anche quando questo non è ancora pronto. La Continuous Delivery fornisce feedback allo sviluppatore sulla base di test automatizzati che controllano principalmente il build ogni qualvolta viene modificato il codice sorgente.
Secondo lo sviluppo software classico, il prodotto finale viene rilasciato non appena contiene tutte le caratteristiche richieste, funziona bene e non risultano gravi difetti dal controllo qualità. Lo sviluppatore di solito prevede patch e aggiornamenti del software a intervalli più o meno regolari. Grazie alla Continuous Delivery il prodotto viene rilasciato già nelle prime fasi di sviluppo, mentre è ancora in elaborazione. La bozza prevede spesso solo le funzioni di base del software, che vengono poi testate dal cliente in un ambiente reale. Il cliente (e chi testa il software per lui) gioca dunque un ruolo fondamentale nel controllo qualità.
Il feedback che ne deriva aiuta lo sviluppatore a migliorare le caratteristiche del programma durante lo sviluppo, ricevendo possibilmente importanti indicazioni su quale deve essere l’aspetto successivo da sviluppare. In assenza della Continuous Delivery, questo processo risulta piuttosto faticoso e può portare

malumori da entrambe le parti: il cliente aspetta un prodotto finito che soddisfi le sue esigenze e i suoi desideri; lo sviluppatore dall'altra parte non sa ancora in cosa esso consista esattamente. Se la comunicazione sullo stato dello sviluppo del prodotto avviene già durante le prime fasi di lavoro, è più facile tenere in considerazione i desideri del cliente ed evitare errori. Tale principio può essere rappresentato come un ciclo:

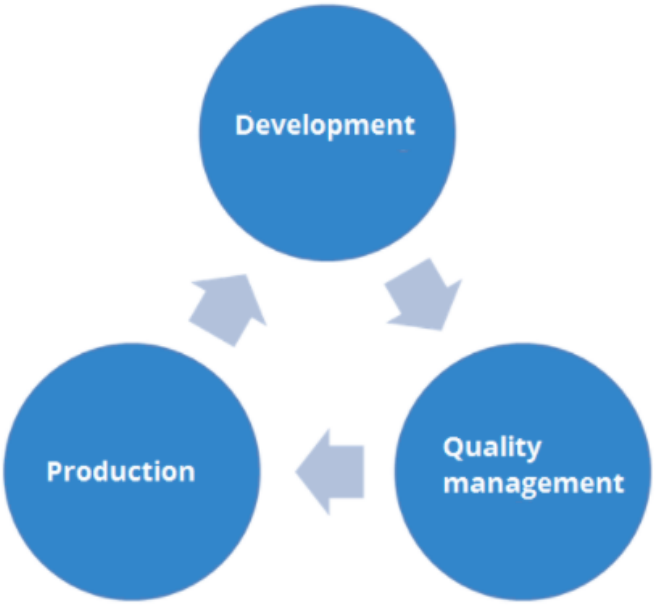


Figura 21 *– Ciclo di sviluppo*

(Digital Guide IONOS, 2019)

Le tre fasi di sviluppo, controllo e produzione non si estinguono in un unico processo, ma sono costantemente interconnesse tra loro: il prodotto quindi attraversa continuamente queste singole fasi e viene sempre migliorato. Con un gran numero di clienti non si può più fare a meno dell'automatizzazione: ed è qui che si inserisce la Continuous Delivery, che automatizza l'intero processo.
Grazie alla Continuous Delivery è possibile testare in tempo reale ogni elaborazione ed estensione del software (praticamente ogni modifica apportata al codice sorgente), raccogliendo così feedback costanti, utili a individuare i problemi e risolverli già nelle prime fasi di sviluppo. Un sistema molto pratico per stabilire in modo tempestivo quale parte del codice ha causato un bug: senza la Continuous Delivery la ricerca del problema potrebbe diventare davvero dispendiosa.
Prima del rilascio al cliente, il software attraversa nel suo stadio intermedio la pipeline di Continuous Delivery: in questa fase vengono effettuati test sia automatici che manuali.
Solo quando tutti i test sono stati superati con un feedback positivo viene creata una versione "stabile" ("stable") e il prodotto viene ufficialmente pubblicato (questo processo come anche la stessa applicazione rilasciata viene chiamato

“release”). La probabilità che il cliente riceva un prodotto senza errori è molto alta (Digital Guide IONOS, 2019).

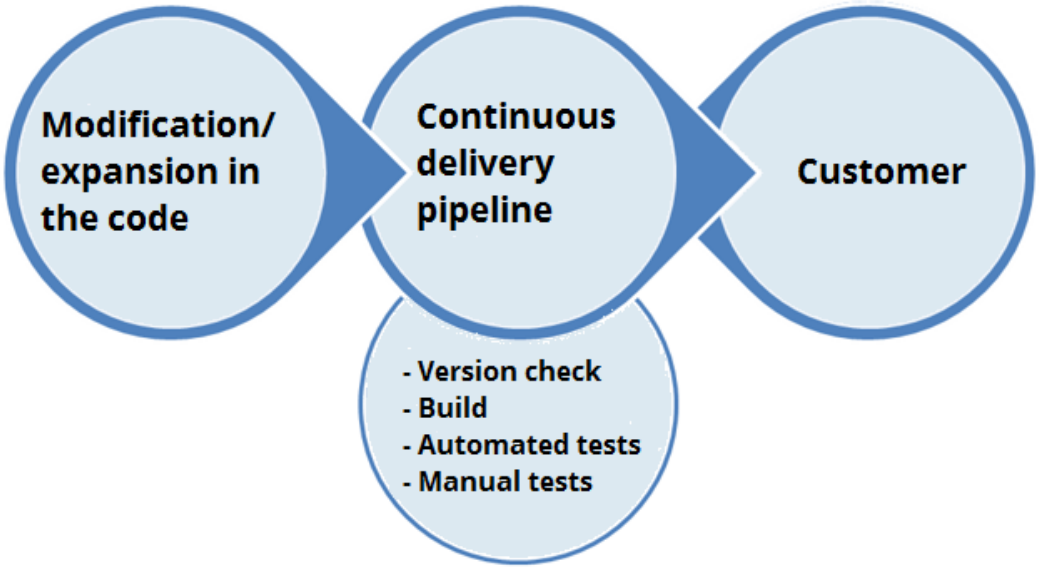


Figura 22 *- Continuos Delivery*

(Digital Guide IONOS, 2019)

Il grosso vantaggio della Continuous Delivery è che conviene sia al cliente che allo sviluppatore: il cliente riceve il prodotto più in fretta e di regola senza errori, lo sviluppatore effettua dei “test sul campo” che sono molto più attendibili dei test beta effettuati in azienda, in quanto forniscono dati e indicazioni preziosi. Il completo processo di sviluppo diventa molto più flessibile e il rischio di pubblicare un software con degli errori viene ridotto al minimo (Digital Guide IONOS, 2019).

Le fasi della pipeline di Continuous Delivery

Con la modifica del codice si attiva la pipeline di Continuous Delivery e viene eseguito il processo di test. Di seguito le fasi che percorre la pipeline di Continuous Delivery:

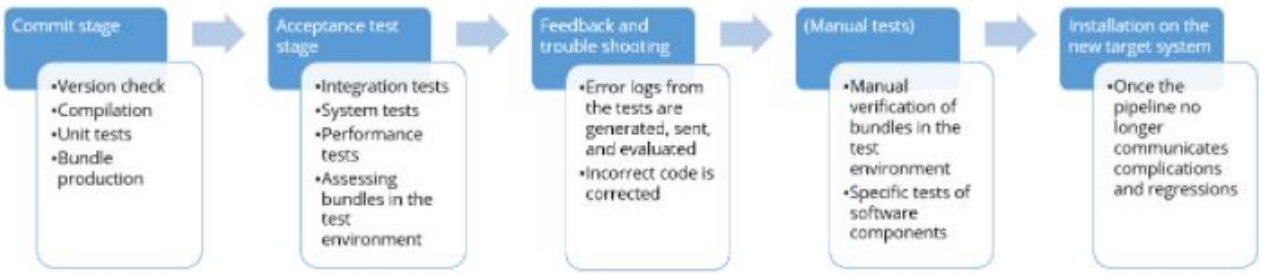


Figura 23 *- Pipeline Continuos Delivery*

- **Commit Stage:** in questa prima fase di test la versione del software viene provata, le componenti di software compilate e necessari test di unità eseguiti. Dopo che i test si sono conclusi con buon esito, questa fase è terminata. Gli artefatti binari delle componenti di software vengono raggruppati in bundle e archiviati in repository. Questo pacchetto è determinante per la funzionalità della pipeline perché indica lo stato del software: il pacchetto contiene la quantità di dati che in seguito sarà installata sul sistema finale. I risultati dei test nel Commit Stage possono quindi essere assegnati alle modifiche specifiche del codice sorgente: ciò rappresenta il vantaggio significativo della Continuous Delivery.

- **Acceptance Test Stage:** in questa seconda fase di test avvengono i test di accettazione, tra cui i test di integrazione (funziona l'interazione tra le componenti?) e gli importanti test di sistema (funziona il software sul sito utente?). Inoltre ci sono test opzionali che possono essere integrati all'Acceptance Test Stage, come i test di prestazione e altri test che controllano i requisiti non funzionali del software. Per l'Acceptance Test Stage si utilizzano ancora i bundle della fase precedente installati in un ambiente test adatto.
  Gli eventuali errori o le complicazioni che si registrano in queste fasi vengono documentate e, se necessario, inviate allo sviluppatore come feedback, via e-mail, programmi di messaggistica o tramite Tool speciali. Le segnalazioni di errore o le regressioni si riferiscono sempre all'ultima modifica, in quanto la pipeline si attiva ad ogni modifica del codice. Quindi lo sviluppatore può reagire velocemente per correggere errori o per ritoccare i codici errati.
  All'occorrenza vengono eseguiti anche test manuali. Anche per questi test la pipeline si serve del bundle della prima fase e lo installa in un ambiente test adatto.
  Se i test si concludono tutti con esito positivo, il pacchetto può essere installato manualmente sul sistema di destinazione: per farlo di solito c'è bisogno semplicemente di "premere un pulsante". Se anche questa fase è automatizzata si parla di Continuous Deployment.

  (Digital Guide IONOS, 2019)

Continuous Deployment (CD)

Continuous Deployment (CD) è un processo di rilascio del software che utilizza test automatizzati per convalidare se le modifiche a una base di codice sono corrette e stabili per la distribuzione autonoma immediata in un ambiente di produzione.

Il ciclo di rilascio del software si è evoluto nel tempo. Il processo legacy di spostare il codice da una macchina a un'altra e verificare se funziona come previsto era un processo soggetto a errori e pesante in termini di risorse. Ora, gli

strumenti possono automatizzare l'intero processo di distribuzione, consentendo alle organizzazioni di ingegneria di concentrarsi sulle esigenze aziendali principali anziché sul sovraccarico dell'infrastruttura.

Nella fase di consegna, gli sviluppatori esamineranno e uniranno le modifiche al codice che verranno poi impacchettate in un artefatto. Questo pacchetto viene quindi spostato in un ambiente di produzione dove attende l'approvazione per essere aperto per la distribuzione. In fase di deployment, il pacchetto viene aperto e rivisto con un sistema di controlli automatizzati. Se i controlli falliscono il pacco viene rifiutato.

Quando i controlli passano, il pacchetto viene distribuito automaticamente in produzione. La distribuzione continua è la pipeline di distribuzione software automatizzata completa end-to-end.

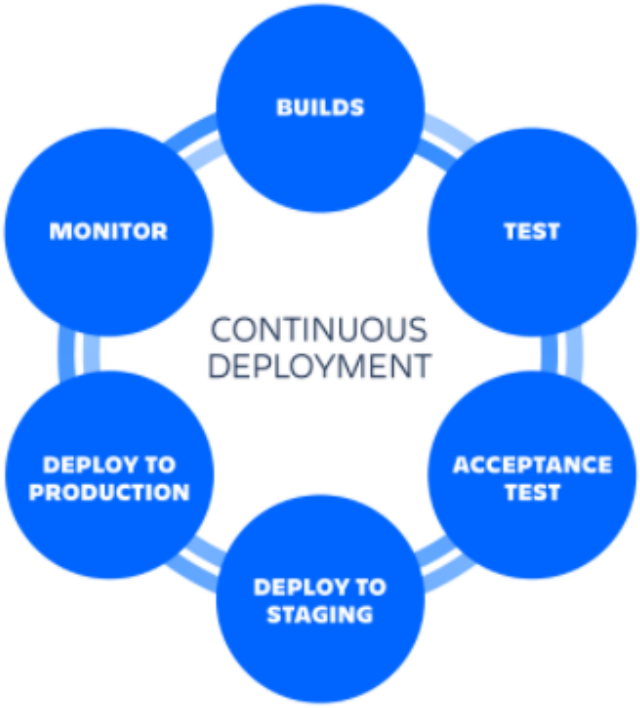


Figura 24 *- Continuos Deployment*

La distribuzione continua offre incredibili vantaggi in termini di produttività per le moderne aziende di software. Consente alle aziende di rispondere alle mutevoli esigenze del mercato e ai team di implementare e convalidare rapidamente nuove idee e funzionalità.

Con una pipeline di distribuzione continua in atto, i team possono reagire al feedback dei clienti in tempo reale. Se i clienti inviano segnalazioni di bug o richieste di funzionalità, i team possono rispondere immediatamente e distribuire le risposte. Se il team ha un'idea per un nuovo prodotto o funzionalità, può essere nelle mani dei clienti non appena il codice è stato inserito (Pittet, 2021).

Continuous Testing

Il prerequisito per il Continuous testing è automatizzare ogni test case. Non è pensabile un approccio in cui ogni nuova funzionalità viene provata attraverso un test eseguito manualmente dal programmatore e mai ripetuto. I test devono essere definiti a priori, ed eseguiti ogni volta che viene implementata una nuova funzionalità, in questo modo, il programmatore può continuare a programmare, in quanto i test, essendo automatici e quindi senza una gestione manuale, saranno veloci garantendo un rapido rilascio del codice (Pittet, 2021).

Continuous Monitoring

I test possono essere applicati anche all'ambiente di produzione, ciò permette di monitorare continuamente i vari ambiente e poter agire di conseguenza, oltre ad avere anche una panoramica sulle prestazioni.

Attraverso appositi strumenti, è possibile analizzare in maniera automatica i dati, ottenere visualizzazioni end-to-end delle applicazioni e usare informazioni dettagliate basate su Machine Learning per identificare e risolvere rapidamente i problemi. È importante che il team riceva una notifica quando le infrastrutture e i servizi correlati non sono raggiungibili(down) (Pittet, 2021).

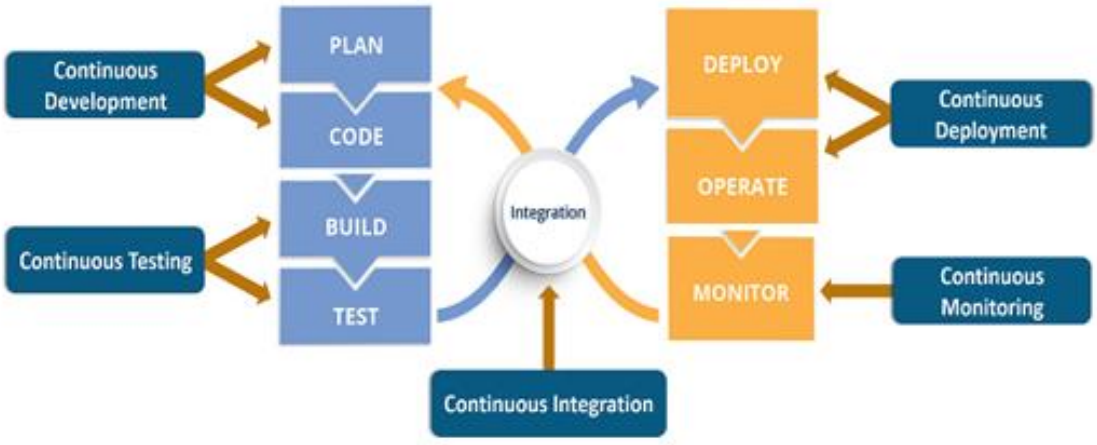


Figura 25 *- Overview DevOps*

7.8 STRUMENTI DEVOPS

Al fine di favorire e applicare la cultura DevOps, i team hanno a disposizione molti strumenti, utili per le proprie attività e per ogni fase del ciclo di vita dell'applicazione.
E' importante sottolineare che l'adozione di uno strumento specifico o una tecnologia non corrisponda all'adozione di DevOps nella sua interezza.
Questo non vieta di poter implementare alcuni strumenti DevOps al fine di semplificare le procedure.
I principali strumenti per l'implementazione dell'approccio DevOps, sono i seguenti:

Continuos Integration

- *Mantenere un repository del codice sorgente*

Senza avere un repository del codice è impossibile automatizzare il build ed i test. Per questo motivo va utilizzato un repository Git, come ad esempio: **Github.**

Github è un sistema di controllo di che consente di gestire le directory contenenti i file dei progetti.

I vantaggi offerti da un sistema distribuito come Git sono molteplici:

- Permette di eliminare gran parte delle operazioni manuali sui file di progetto e la duplicazione.
- Traccia tutte le modifiche ai file, rendendo possibile l'accesso immediato a una versione del codice in un momento precedente.
- Associa le modifiche all'utente che le ha introdotte, offrendo un sistema di tracciamento della contribuzione.
- Permette la progettazione di una varietà di flussi di lavoro e di integrazione delle modifica per scenari di complessità elevata.
- E' persistente e robusto, poiché qualsiasi copia del repository è indipendente e conserva la storia del progetto fin dall'inizio.

Gli sviluppatori lavorano normalmente sui file di progetto, con in più la possibilità offerta dal sistema di controllo di versione di salvare le modifiche in qualsiasi momento, con un'operazione chiamata *Commit*, simile a una fotografia o istantanea dei file nel momento in cui decidiamo di salvare la lavorazione.

I Commit sono salvati nel database, possono essere percorsi avanti e indietro e generano lo storico di tutte le lavorazioni effettuate.

- *Automatizzare il build*

Per build si intende il processo che trasforma il codice sorgente in un artefatto. Deve essere possibile eseguire la compilazione e la creazione dei pacchetti da mettere sui server in modo completamente automatizzato, senza alcun intervento umano. Per questo si possono utilizzare: **Maven, Apache Ant, Gradle**

- *Rendere il build auto-testante*

Ogni volta che il codice sorgente viene compilato ed impacchettato, è importante che vengano eseguiti dei test sul sorgente affinché la qualità del codice venga tenuta sotto controllo ed eventuali bug vengano scoperti il prima possibile. Uno strumento che viene utilizzato ad esempio è **Jenkins**.

Tutti eseguono Commit alla baseline tutti i giorni: è inutile compilare e testare il sorgente se gli sviluppatori sviluppano codice sui propri computer senza sincronizzarsi spesso col codice scritto da altri. È importante però che il codice sia compilabile ed abbia superato tutti i test. Per poter effettuare dei Commit si utilizza **Git**.

Ogni Commit fa partire una build. Ogni modifica al codice sorgente condiviso potrebbe generare dei bug. Compilare e testare subito dà la possibilità di intervenire immediatamente su eventuali bug. Per questa tipologia di attività vengono utilizzate le funzionalità di **Jenkins** e di **Github.**

Continuos Delivery/Deployment

Nel seguito una sintesi di tools che vengono maggiormente utilizzati per eseguire la *Continuous Integration e il Continuous Delivery/Deployment* applicati ad un semplice progetto IT con Web Services, già precedentemente descritti.

- *GIT*
- *Github*
- *Jenkins*
- *MySql*: è un relational database management system (RDBMS) composto da un client a riga di comando e un server.

7.9 IL CLOUD NELLA DIGITAL TRANSFORMATION

Nell'economia digitale odierna, le imprese dipendono dall'uso efficace della tecnologia non solo per supportare i processi aziendali in corso, ma anche per guidare nuove fonti di differenziazione competitiva.
Per molte aziende, il successo o il fallimento si collega direttamente all'efficacia dei loro ambienti di distribuzione dei servizi IT.
Un pilastro fondamentale dell'ecosistema della tecnologia della digital transformation è senza dubbio il cloud computing, ed è in questo contesto, quello dell'innovazione aziendale, che deve essere visto il futuro del cloud.
Proprio come un altro ecosistema di tecnologie di trasformazione digitale di cui si parla molto quando si tratta di innovazione, IoT, il cloud computing è di un termine generico, con diverse tecnologie, componenti, approcci e tipi di applicazioni.
Con una raffica di nuove tecnologie digital tra cui gli smartphone e il mobile, programmi di analisi e business intelligence, social media e sensori per rilevare dati ovunque, le aziende sono alla ricerca di un punto di svolta. Ma con budget IT che non crescono e una gamma di applicazioni e tecnologie in continua espansione, l'innovazione può essere elusiva, difficile da individuare, attivare e riconoscere. Molti reparti IT fanno gli straordinari solo per stare al passo con le operazioni quotidiane, lasciando poco tempo per concentrarsi sulle iniziative strategiche che aiutano le imprese a prosperare a lungo termine.
La ricerca delle migliori soluzioni tecnologiche in cloud può aiutare le aziende a concentrarsi sull'innovazione, supportando un provisioning dei servizi più rapido, riducendo i costi IT e, cosa forse più importante, aumentando l'agilità aziendale.
Infatti, il cloud computing può aiutare le aziende a soddisfare le esigenze del business, riducendo i tempi dei cicli di implementazione e sviluppo, aumentando l'efficienza, trovando nuovi clienti ed espandendo i vantaggi strategici.
Grazie alla capacità del cloud di adattarsi rapidamente alle mutevoli condizioni del business, le aziende possono puntare all'ottimizzare del marketing, delle operation, dell'amministrazione, della logistica e del sales.

Esistono tre modelli generali di implementazione del cloud: pubblico, privato e ibrido.

- Un *cloud pubblico* è il luogo in cui un fornitore indipendente, di terze parti, come Amazon Web Services (AWS) o Microsoft Azure, possiede e gestisce risorse di elaborazione a cui i clienti possono accedere attraverso Internet.

- Quindi, al contrario, un cloud privato viene creato e gestito da una singola azienda. Il cloud privato potrebbe essere basato su risorse e infrastrutture già presenti nel data center locale di un'organizzazione o

creato e sviluppato su una nuova infrastruttura separata. In entrambi i casi, la stessa azienda possiede e gestisce il suo cloud privato.

- Un cloud ibrido è un modello in cui un cloud privato si connette all'infrastruttura cloud pubblica, consentendo a un'organizzazione di orchestrare i carichi di lavoro tra i due ambienti. In questo modello, il cloud pubblico diventa un'efficace estensione del cloud privato, per formare un unico cloud uniforme. Una distribuzione cloud ibrida richiede un alto livello di compatibilità tra il software e i servizi sottostanti utilizzati sia dai cloud pubblici e privati.

(Fracasso, Cloud-nella-digital-transformation, 2018)

8 CASO DI STUDIO

8.1 INTRODUZIONE

L'obiettivo di questo lavoro è analizzare, a partire dalle caratteristiche delle principali metodologie di Project Management descritte nei capitoli precedenti, la modalità di adozione dei vari framework di gestione all'interno di progetti di un reale contesto.

I progetti analizzati sono orientati allo sviluppo di siti web dei Brands, gestiti da un'azienda multinazionale che opera nel settore Automotive con una rilevante quota di mercato.

L'analisi è relativa non sono alla modalità di applicazione delle metodologie, ma vuole porsi come obiettivo, quello di capire eventuali benefici ed aspetti migliorativi, oltre a capire quali barriere non consentono l'applicazione di operare completamente in modalità agile.

Inoltre, a partire da più base dati, è stata sviluppata un'analisi quantitativa volta a verificare gli impatti che la metodologia adottata ha sull'Application Maintenance.

8.2 L'AZIENDA E LA TRASFORMAZIONE DIGITALE

Il mondo in cui opera l'organizzazione sta diventando sempre più complesso, in un contesto di mercato Globale.
In seguito ad una recente fusione con un altro gruppo che opera nel settore dell'Automotive, nasce l'esigenza di ottimizzare le sinergie di 14 marchi-Brands e la necessità di rispondere in maniera sempre più tempestiva ai clienti, nell'ambito di un mercato sempre più "fluido".

Importante considerare il fenomeno della Trasformazione Digitale (*Digital Transformation)* che ha richiesto un cambio della cultura aziendale.
Al fine di essere competitivi e attrarre nuovi clienti, è stato necessario prevedere l'integrazione delle tecnologie digitali in tutti gli aspetti del business.
L'era digitale ha messo al centro il cliente e le sue scelte consapevoli ed autonome. Supportati da numerose informazioni reperibili grazie ad Internet ed ai social network, sono mutate le esigenze dei "nuovi clienti", sempre alla ricerca di qualità e convenienza, sfruttando canali che esulano da quelli della vendita tradizionale.
Prima di recarsi fisicamente presso un Dealer del gruppo aziendale, il cliente è spinto ad effettuare una ricerca online sfruttando i siti web ed effettuando una

configurazione o avviando un processo di acquisto online (per mezzo di piattaforme e-commerce).
E' solo nella fase finale che l'acquirente si lascia orientare maggiormente dal venditore. (Federauto, 2016)
Da qui la necessità di creare esperienze sempre più personalizzate e coinvolgenti per i clienti. La tecnologia si è evoluta per adattarsi a questo scopo.
L'obiettivo è quello di migliorare ed evolvere la *Customer journey,* ovvero il percorso e la relativa esperienza che il cliente finale vive, durante la relazione con l'azienda.
Una storia che inizia nel momento in cui il cliente cerca un bene o servizio di un'azienda per soddisfare un proprio bisogno, e finisce con l'acquisto.
Oggi questa ricerca, più che mai, viene effettuata sfruttando i siti web.
Ed è per questo motivo che è importante investire e migliorare sempre più l'esperienza utente. Internet è diventato uno strumento: lo si usa se conviene, altrimenti no. Con dieci volte più siti e cento volte più pagine fra cui scegliere, gli utenti hanno sempre meno pazienza per siti difficili da usare; ogni errore nel progettare un sito significa affari mancati. L'usabilità è più importante che mai (Customer Journey).

Un altro fattore che ritengo sia importatane considerare è l'impatto che il Covid-19 ha avuto nel settore Automotive.
L'impatto del Covid-19 sull'Automotive è stato un fattore che ha portato a delle criticità ma anche a delle opportunità.
Il comparto auto, è stato colpito dal Covid-19 fin dalla fase iniziale dell'epidemia quando gli effetti, sulla supply chain con origine in Cina, hanno cominciato a trasmettersi a livello globale. Ma oltre al blocco della produzione, il settore è stato colpito anche sul fronte della domanda. Molti consumatori, infatti, di fronte all'incertezza dei mesi a venire, hanno rimandato o annullato l'acquisto di una nuova auto, registrando cali fino al 98% (L'impatto del Covid-19 sull'Automotive).

In ultimo, un altro fattore significativo riguarda il fatto che, negli ultimi anni, molte case automobilistiche hanno deciso di investire sulla mobilità elettrica. La sensibilità dei consumatori verso forme di mobilità ecologica è sempre più sviluppata: nell'ultimo anno, prima dello scoppio dell'epidemia Covid-19, l'interesse dei consumatori italiani per i veicoli ibridi/elettrici era salito al 71%, dal 58% dell'anno precedente. Nasce dunque la necessità di investire, oltre che sulla produzione anche sull'area *Digital Marketing* al fine di attrarre e coinvolgere il cliente mediante quelli che vengono chiamati siti vetrina.
Nasce la necessità di sfruttare delle tecnologie evolute, come ad esempio il chat bot, la voice interaction che, unita ad una User Experience d'effetto, attraggono il cliente sul web (L'impatto del Covid-19 sull'Automotive).

8.3 L'IMPORTANZA DELL'ESPERIENZA UTENTE

Parlando di realizzazione di siti web, spesso citerò termini come "esperienza utente", "usabilità", "User Experience", "UX", "comunicazione".
Prima di entrare nel merito del caso d'uso, ritengo sia importante spiegare cosa si intende nello specifico e quanto sia importante questo aspetto per un'azienda che investe nell'esperienza Digitale.

In ambiti di siti web con i termini esperienza utente, usabilità e User Experience, ci si riferisce alla facilità con il quale un utente riesce ad utilizzare (navigare) una pagina web.
E' importante dunque capire che gli utenti abbandonano siti web che non reputano gradevoli o semplici da navigare. In internet è tutto a portata di clic. Se un sito non è gradito basta un clic per tornare indietro e visitarne un altro.

L'azienda oggetto del caso di studio, utilizzando reportistiche generate da tagging di Analytics inseriti nelle pagine dei siti (Google Analytics o Adobe Analytics), si focalizza nel capire come l'utente naviga il sito, quanto tempo rimane su una pagina, quante pagine visualizza e quali, quelle che sono le percentuali di abbandono e così via.
Prevede inoltre di effettuare delle attività A/B test, che consentono di capire quale User Experience sia maggiormente preferibile dall'utente, proponendo ad esso due o più versioni diverse della stessa pagina.

Fattori importanti che condizionano l'usabilità di siti web sono:

- **Velocità**. Mai come oggi gli utenti esigono la velocità. Velocità di caricamento delle pagine, velocità nel trovare le informazioni che stanno cercando e così via. Pagine web lente nel caricarsi (e parliamo di secondi) è una delle maggiori cause di abbandono. Gli utenti sono estremamente impazienti.

- **Architettura dei contenuti.** Le informazioni dovrebbero essere organizzate secondo una scala gerarchica e tutte (sicuramente le più importanti) dovrebbero poter essere raggiunte in pochi secondi tramite due o al massimo tre clic. Uno dei problemi più critici, riscontrati dagli studi di usabilità, riguarda proprio la ricerca delle informazioni. Il caso di studio in esame sarà molto focalizzato su questo ultimo aspetto.

- **Responsive Design.** Le pagine web dovrebbero adattarsi perfettamente alle diverse risoluzioni quali desktop, tablet e smartphone. Gli utenti detestano contenuti che fuoriescono dal display e che comportano ad esempio lo scorrimento orizzontale. Inoltre, il traffico più alto si registra ormai su device mobile. Anche in questo caso vedremo come la nuova

UX, studiata per la tipologia di progetto oggetto di studio, è stata orientata ad una esperienza Mobile First.

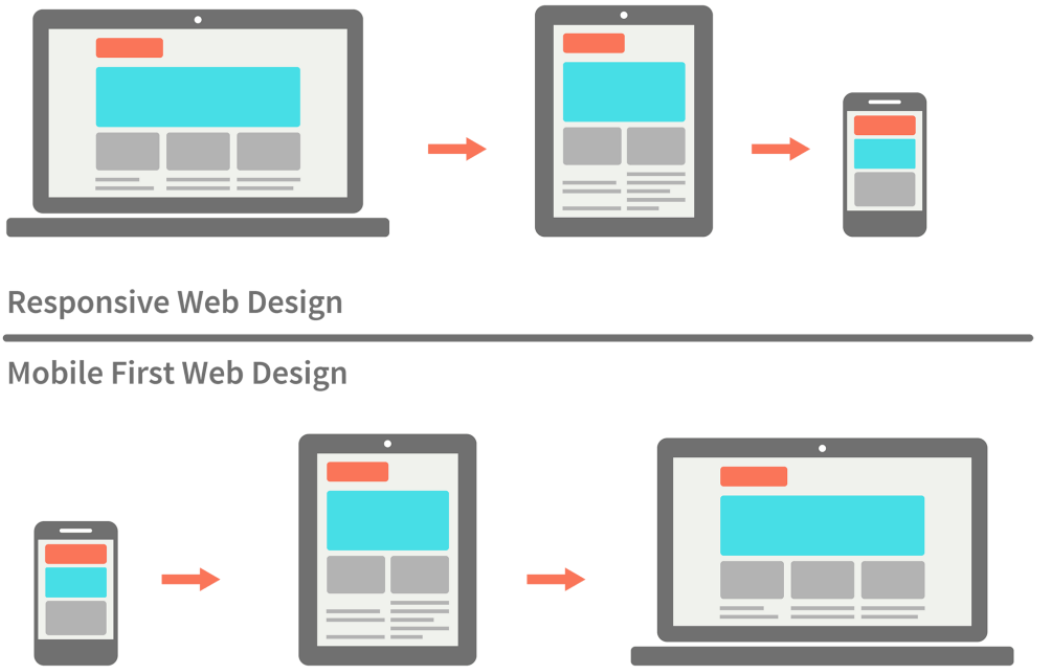


Figura 26 *- Mobile First Design*

(Gonzalo, 2017)

- **Qualità dei contenuti**. Gli utenti si aspettano di ottenere informazioni originali o quantomeno complete ed esaustive circa quello che stanno cercando. Offrire contenuti incompleti o "visti e rivisti" non gioverà all'esperienza complessiva dell'utente.

- **Contrasti colore e leggibilità del testo.** Anche se strano, ci sono ancora tanti siti web che fanno un utilizzo errato dei contrasti colore creando molta confusione. I contrasti colore dovrebbero essere usati con criterio per focalizzare l'attenzione dell'utente su determinati elementi della pagina web. Sono errori critici di usabilità anche quelli legati al testo e alla sua formattazione. Testo giustificato, dimensioni troppo piccole e caratteri poco leggibili sono da evitare.
-

(Quarto, Esperienza Utente)

8.4 DA UN APPROCCIO "WATERFALL" AD UN APPROCCIO "IBRIDO"

Prendendo in considerazione dunque il contesto in cui opera l'organizzazione, un ambiente sempre più competitivo, dinamico, globale dove le esigenze dei clienti aumentano in funzione anche dell'avanzamento della tecnologia, negli anni è stato necessario adottare in azienda, a livello di Project Management, delle metodologie meno "pesanti" come l'approccio "Waterfall" e più orientate al raggiungimento dei risultati.

Vale la pena specificare che, in questo specifico contesto, il raggiungimento dei risultati consiste nella soddisfazione non solo del cliente finale ovvero "l'end user". L'end user è colui che accede e naviga il sito web in ottica di esplorare i dettagli dei modelli di auto, configurare dei veicoli, compilare delle form di contatto o consultare l'elenco delle macchine usate o kilometro zero.
Il raggiungimento dei risultati consiste anche nel rispettare tempi, costi e requisiti di business che, nel caso d'uso descritto, coincide con l'ente aziendale Brand/Team Digital.

L'obiettivo che si sta percorrendo, è dunque quello di affiancare al tradizionale approccio Waterfall, molto utilizzato in azienda in passato, un approccio più Agile, prevedendo in specifiche fasi progettuali, l'adozione di metodologie più iterative volte al maggior coinvolgimento del cliente sin dalle prime fasi di analisi, durante alcune fasi di rilascio/condivisione dei risultati e nella fase di Maintenance.
Vedremo nel paragrafo successivo che, tale approccio, verrà impiegato principalmente in due fasi:

- Fase di studio e analisi del Design e delle Nuove Funzionalità di un sito web
- Fase di sviluppo/integrazione, facendo riferimento ad alcuni principi DevOps.
- Fase di "Application Maintenance".

8.5 IL PROGETTO DIGITALE (SITO WEB)

Di seguito sono illustrate le fasi previste nel modello ibrido, basato su Waterfall ed Agile, applicato negli ultimi anni, a progetti di sviluppo di prodotto digitali come la realizzazione di siti web.

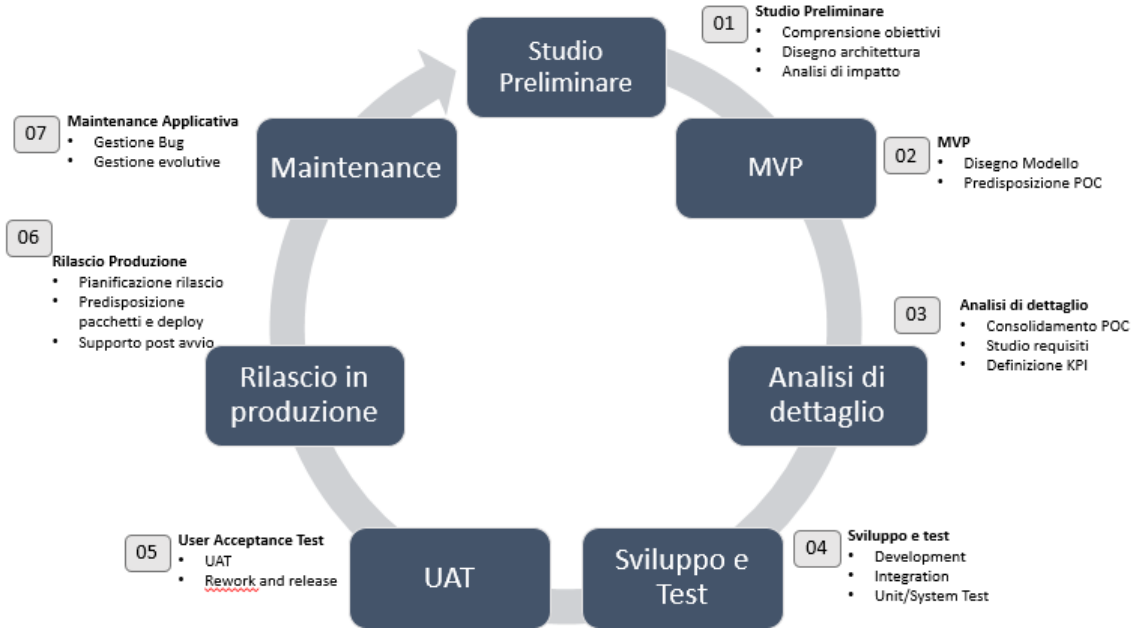


Figura 27 *- Fasi del progetto di sviluppo sito web*

Come si può evincere dalla figura, rispetto ad un approccio Waterfall tradizionale, il modello applicato ha previsto l'esecuzione di una fase di modellazione aggiuntiva tra lo studio preliminare e l'analisi di dettaglio, denominata "**MVP"**.
MVP sta per *Minimum Viable Product* ed è la versione del prodotto con caratteristiche appena sufficienti per essere utilizzabile. Questa fase consente di ricevere i primi feedback da parte del cliente prima di procedere con l'analisi di dettaglio e lo sviluppo futuro del sito. (Minimum_Viable_Product, 2021)
Il metodo ha previsto dunque l'interazione con il business fin dalla fase di idea, al fine di validare il binomio problema/cliente prima di realizzare una qualsiasi forma di prototipo.
Si è trattato di un processo iterativo di generazione di idee, prototipazione, presentazione, raccolta dati, analisi ed apprendimento.
Tale fase, coerentemente con quanto previsto da Agile, è stata inserita all'interno delle attività progettuali. Lo scopo è stato quello di anticipare i feedback del Business in merito alla soluzione da adottare, attraverso l'implementazione di Mock-up o POC che hanno consentito di condividere, in maniera chiara ed immediata, la proposta che si desiderava implementare. In questo modo è stato possibile raccogliere subito un riscontro sulla bontà della soluzione prescelta, minimizzando i ricicli e le attività di rework successive che, soprattutto nei progetti digitali legati a sviluppi di siti web, per loro natura molto visuali, risultano verificarsi molto di frequente in fase di UAT (User Acceptance Test).
La metodologia *iterativa* è dunque stata utilizzata per gestire le fasi di *MVP* e *Detailed Design* del nuovo sito web.

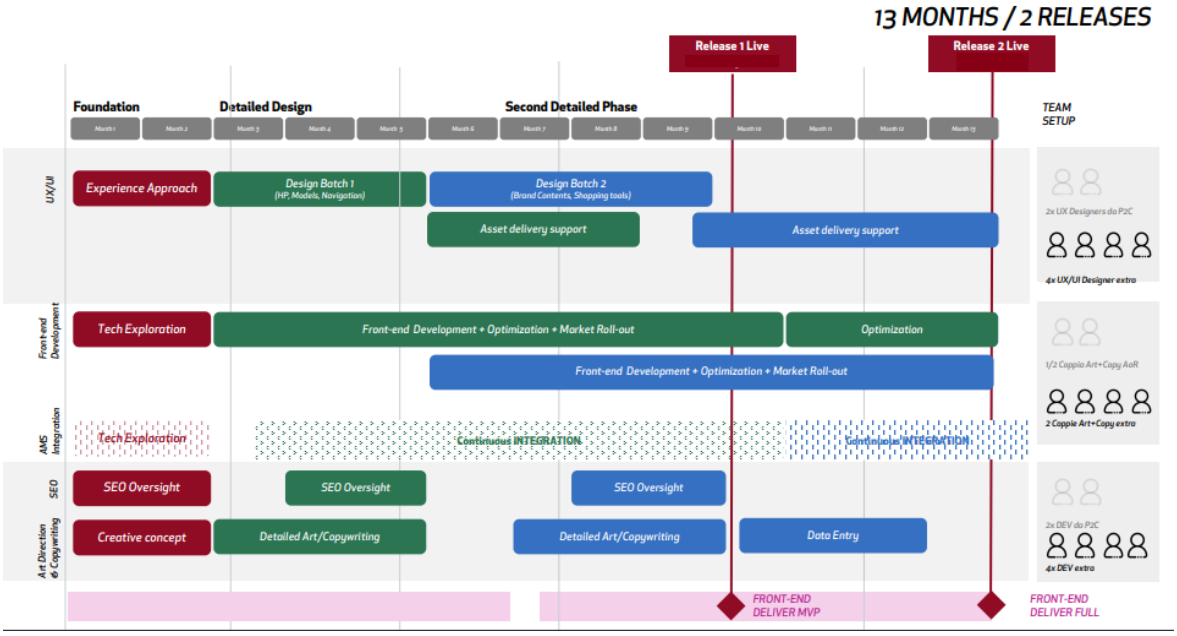


Figura 28 *- Pianificazione progetto caso di studio*

8.5.1 Studio preliminare

Il principale driver di business, per questa tipologia di progetto che prendo come riferimento per il caso di studio in esame, è stato quello di prevedere un risultato d'impatto per lo sviluppo di un nuovo sito di Brand, Mobile-native, che includesse anche un'esperienza digitale guidata alla conversazione (Voice Interaction).

8.5.2 MVP - *Minimum Viable Product*

Gli MVP condivisi in questa fase, sono stati orientati al raggiungimento dei seguenti obiettivi:

- *Design minimale e piccole iterazioni*: oltre alla realizzazione di asset (immagini e video) ad alta risoluzione, l'obiettivo è stato quello di offrire delle interazioni coinvolgenti al cliente.

- *Mobile First*: una soluzione orientata all'esperienza mobile e orientata a migliorare i KPI. L'obiettivo è stato quello di incrementare i volumi di utenti che navigano il sito da device.

  I principali KPI del progetto, sono stati i seguenti:

- *Incrementare il traffico mobile*: analizzando i reports Analytics aziendali, è emerso che oggi la home page del sito ha il 40% di visitatori che accedono per mezzo di device mobile. Una riprogettazione mobile-native avrebbe consentito dunque di aumentare tale numero.

- *Incrementare il traffico dalla pagina di modello verso il configuratore* (Tool che viene utilizzato per configurare il modello delle auto, direttamente online). Analizzando i reports Analytics è emerso che oggi, il 40% degli utenti, per il mercato Italia, accedono al configuratore partendo dalla pagina di modello del veicolo a cui sono interessati. Riprogettando dunque la pagina di modello, rendendola più attraente, sarebbe stato possibile aumentare il numero di visite.

- *Incrementare la qualità (numero di pagine navigate e tempo di consultazione).*

- Attrarre un *target di consumatori più giovani* mediante un sito che offra una esperienza coinvolgente.

- *Aumentare la percentuale di raccolta dati* (Lead) con conseguente aumento percentuale dei volumi di vendita. In generale, quante più persone compilano su un sito web un form di richiesta preventivo, tanto più vi è la possibilità che un processo di acquisto presso il Dealer, sia completato.

In termini di *action*, i driver di business sono stati tradotti nel seguente modo:

- Completo re design UI (font, colori, component visual)
- Architettura/alberatura del sito completamente rivista
- Ridefinizione di tutti i contenuti del sito.
- UX re design di tutte le aree di contenuto.
- UX re design di tutti i tools integrati
- UX re design del 50% dei componenti attualmente utilizzati
- Sviluppo di nuovi tools

8.5.3 Analisi di dettaglio

Anche in questa fase è stata prevista la predisposizione di un set di Mockups che consentisse di garantire l'aderenza delle esigenze di usabilità degli utenti, riducendo sensibilmente il gap tra le attese utente e l'effettiva realizzazione.
Per ottimizzare l'esecuzione di questa fase, l'approccio adottato ha previsto l'esecuzione di sessioni fortemente interattive, il cui scopo è stato quello di

andare ad alimentare una flipchart con tutti gli elementi di cui sopra, così che lato business si è potuto avere la percezione visiva e immediata del tipo di soluzione proposta, mentre lato progetto si sono potuti intercettare subito tutti gli input utili ad assicurare una progettazione lineare e completa in ogni sua parte.
Come previsto in Agile, gli sviluppatori sono stati comunque parte integrante di questa fase analisi, al fine di verificare con anticipo, la fattibilità del prodotto da sviluppare e integrare.
Questo approccio ha evitato di ottenere un Design congelato e approvato, ma non integrabile.

Sulla base di quanto raccolto e analizzato attraverso tali sessioni, sono stati predisposti dei Mockups (attraverso il Tool Figma), condiviso con i key users di riferimento, con l'obiettivo di ricevere feedback per l'affinamento dei requisiti.

Il processo iterativo ha consentito di lavorare immediatamente sui feedback ricevuti dal Business, proponendo in tempi brevi delle soluzioni alternative.

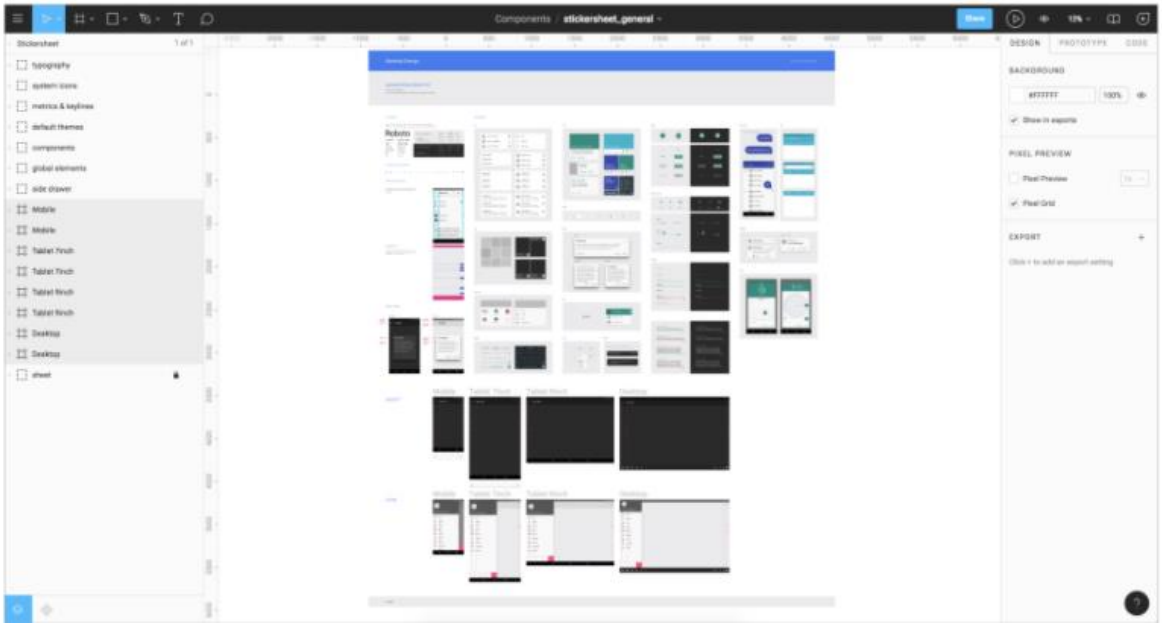


Figura 29 *- Esempio di output Figma*

(Kopf)

8.5.4 Sviluppo e distribuzione–DevOps e strumenti

Come descritto nei precedenti capitoli, DevOps influenza il ciclo di vita dell'applicazione nelle fasi di *pianificazione, sviluppo, distribuzione e operatività.* Non appena la fase di **Foundation** e di **Detailed Design** è stata completata, è stata prevista la consegna al team di sviluppo di un insieme di documenti:

- L’ HTML (codice applicativo di front end, da implementare in piattaforma).

- Wireframe/PSD che ha riportato la struttura e il comportamento dei componenti da sviluppare e che, in generale, costituiscono una parte integrante di pagine già esistenti o di nuove pagine.

- Documenti funzionali che hanno descritto le diverse funzionalità, o parte di esse, al fine di consentire agli sviluppatori di comprendere i bisogni del cliente (concetto simile a User Story visto nei precedenti capitoli dedicati all’approfondimento del framework Agile XP).

Da questo momento in poi, la documentazione consegnata, è stata considerata approvata e si è cercato di limitare quanto più possibile le modifiche ai Requisiti di Business, al fine di evitare ricicli (che avrebbero portato ad un allungamento delle tempistiche di progetto, rischiando il fallimento dello stesso) e change request (che avrebbero alterato il budget approvato).
Da questo punto di vista dunque, la tendenza, è stata quella di adottare uno dei principi descritti nel modello Waterfall.
D’altro canto invece, per quanto riguarda le attività di sviluppo e integrazione, sono stati adottati alcuni approcci e strumenti di DevOps.
Il team di sviluppo ed il team operation nel contesto aziendale specifico, sono enti separati.

- Il *team di sviluppo* è rappresentato da un fornitore esterno, un’azienda leader nei settori Strategy & Consulting, Interactive, Technology e Operations.
- Il *team operation* è un team che appartiene ad un ente aziendale, Technical Competence Center, che si occupa della gestione della infrastruttura aziendale.

Il processo aziendale da sempre adottato, prevede un sistema di ticketing, come interfaccia di comunicazione tra i due team di lavoro: i team di sviluppo richiedono l’implementazione di nuove versioni del software ed il personale operativo gestisce manualmente il ticket.
Negli ultimi anni, la necessità di adottare delle procedure più evolute, ottimizzando i processi tramite la tecnologia, ha portato l’azienda a dover anche modificare la cultura interna dell’organizzazione e delle persone che contribuiscono al lavoro.
L’organizzazione ha puntato alla collaborazione tra i due team, iniziando a condividere i processi, includendo nella pianificazione del progetto anche le tempistiche relative alle attività operazionali di deployment, iniziando a condividere anche gli stessi tools.

Strumenti a supporto DevOps

Come descritto nei precedenti capitoli, l'adozione di uno strumento specifico o una tecnologia, non corrisponde all'adozione di DevOps. Possiamo però dire che le persone possono implementare e semplificare le procedure DevOps se scelgono gli strumenti appropriati.
A tal proposito, un elemento vincente per il risultato di questo progetto, è stata l'adozione di alcuni strumenti DevOps che hanno consentito di accelerare il processo di integrazione e delivery del codice sorgente.
Tali strumenti hanno consentito al team di sviluppo, in collaborazione con il team operation, di essere più autonomi nell'eseguire le operazioni.
Sono stati inoltre molto importanti i corsi ad hoc svolti da ogni gruppo, rispetto le varie soluzioni proposte sul mercato.

CI/CD con Cloud Manager

In collaborazione con la divisione ICT dell'azienda, in fase di progetto è stata adottata una soluzione di *cloud computing*.

Questo ha consentito di ridurre i tempi di cicli di implementazione e sviluppo, aumentando l'efficienza, rendendo i servizi più rapidi, aumentando l'agilità aziendale.

Gli sviluppatori, durante la fase di sviluppo del progetto, hanno utilizzato una metodologia di *Continuos Integration* (CI) e di *Continuos Delivery* (CD) integrando porzioni di codice finite nell'applicazione anche più volte al giorno, con l'obiettivo di risparmiare tempo, durante il ciclo di sviluppo. Questo ha consentito al team di individuare eventuali anomalie nel codice e di intervenire, prima di procedere con i restanti sviluppi e soprattutto prima di consegnare la soluzione al cliente Business.

Uno strumento che inizia ad essere utilizzato all'interno dell'organizzazione e nello specifico contesto descritto, è il prodotto *Adobe Cloud Manager*.
I siti web in questione sono sviluppati su piattaforma *Adobe Experience Manager*.
Adobe Experience Manager Sites, è una piattaforma di gestione dei contenuti Web, che consente di offrire ai clienti esperienze digitali coerenti su vari canali.
Adobe Cloud Manager è un servizio Cloud che consente agli sviluppatori di creare, testare e distribuire applicazioni sviluppate su piattaforma Adobe Experience Manager. Nello specifico contesto, questo strumento ha consentito di gestire le proprie distribuzioni di codice personalizzato sui propri ambienti Cloud gestiti da AEM, con un'automazione gestibile della pipeline e una flessibilità completa per i tempi o la frequenza delle distribuzioni.
Utilizzando l'API di Cloud Manager, gli sviluppatori e coloro che si occupano di operation, hanno avuto la possibilità di integrare processi personalizzati nella pipeline di Cloud Manager.

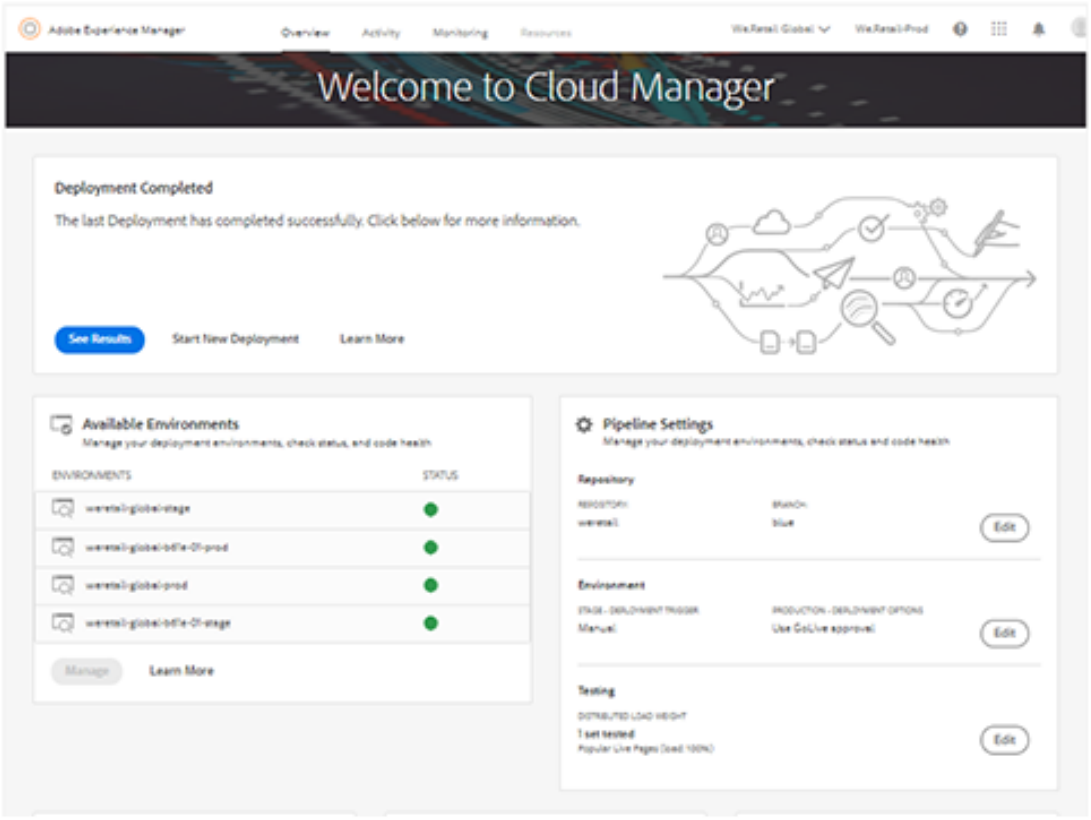


Figura 30 *- Adobe Cloud Manager Console*

(Cloud Manager)

Tale strumento include un framework di integrazione continua e distribuzione continua (CI/CD), che consente ai team IT di implementare e di accelerare la distribuzione di personalizzazioni o aggiornamenti, senza compromettere le prestazioni o la sicurezza.

In linea generale, il prodotto consente di svolgere le seguenti attività:

- **Adottare il processo di Continuous Integration / Continuous Delivery** per ridurre il time to market da mesi/settimane a giorni/ore.

- **Ispezionare il codice, testarlo e convalidarlo in sicurezza**, prima di passare alla produzione per ridurre al minimo i problemi sul sito online.

- **Schedulare una implementazione automatica**, pianificata o manuale, anche al di fuori dell'orario lavorativo per la massima flessibilità e controllo. Questo consente di velocizzare sicuramente le attività, senza attendere che un team dedicato, in base alla schedulazione delle varie attività, prenda in carico un ticket di richiesta di deploy per processarlo.

Nello specifico contesto, poiché questo strumento è stato adottato da poco tempo, è stato utilizzato/sfruttato principalmente per la fase di Continuous

Integration / Continuous Delivery e per la fase di test. L'implementazione automatica non è stata ancora adottata ma sicuramente lo sarà in futuro. Al momento non avrebbe apportato a delle migliorie nel processo di sviluppo e distribuzione del software. Non vi è stata l'esigenza.

L'immagine seguente illustra il flusso di processo CI/CD utilizzato in Cloud Manager:

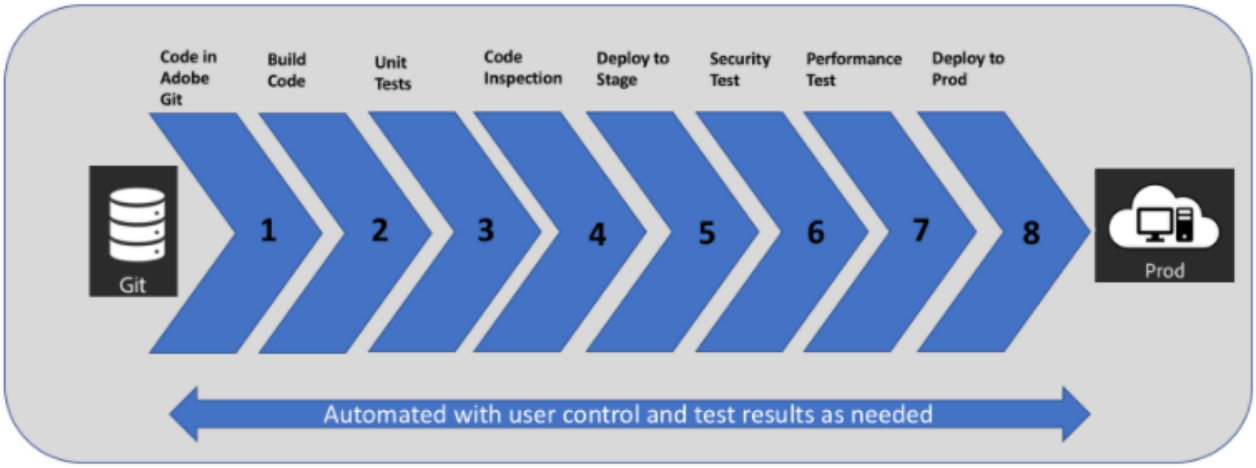


Figura 31 *- Flusso di processo CI/CD*

(Adobe)

Git

Nel contesto aziendale analizzato, gli sviluppatori utilizzano come strumento di controllo del versioning dei file di progetti, depositati in una specifico repository, un prodotto aziendale. Il repository viene utilizzato come archivio ma anche come fonte dunque, per controllare le varie versioni.

Git facilita il lavoro in parallelo di più attori/sviluppatori o anche degli stessi che lavorano su più progetti. Un ulteriore vantaggio nell'adozione di tale strumento in azienda, è dato anche dalla possibilità/facilità di effettuare dei rollback o rework. Grazie al versioning, si ha sempre traccia di un punto di codice "pulito" da cui partire. Facilita anche eventuali Knowledge Transfer essendo presente sul repository, tutto il codice sorgente.

Jenkins

E' un altro strumento adottato in azienda con l'obiettivo di automatizzare il processo di test e rilascio del codice, su ambiente di sviluppo. E' dall'ambiente

di sviluppo che è possibile avviare già dei primi test con l'utente business, prima di procedere sui successivi ambienti previsti da AEM (Certificazione-Stage e Produzione). Anche quest'ultimo rientra tra gli strumenti DevOps che consentono di adottare il modello di Continuos Integration e Continuos Delivery.

Visual Studio

E' una soluzione che rientra nel set completo di soluzioni DevOps. Questo prodotto crea un ambiente di lavoro in cui i membri del team possono compilare, testare e distribuire software in modo rapido, frequente e affidabile. L'impiego di questo strumento, ha consentito di raggiungere più velocemente gli obiettivi grazie a tempi più rapidi nella distribuzione di nuove funzionalità, patch di sicurezza e correzioni di bug.

Tool di monitoraggio/Continuous Monitoring

Trattandosi di sito web, poiché legato a quest'ultimo c'è un concetto di investimenti, immagine, comunicazione, è necessario ed importante che il funzionamento e la raggiungibilità sia sempre garantita.
Immaginiamo ad esempio che un cliente acquisti una macchina presso un dealer e, spinto dalle informazioni presenti nel libretto di uso e manutenzione, decida di conoscere qualche dettaglio in più navigando il sito di Brand. Oppure immaginiamo che grazie ad un banner pubblicitario, un cliente decida di informarsi e recuperare qualche informazione in più sull'auto di interesse, accedendo al sito web.
Questi sono esempi che mi permettono di spiegare quanto possa arrecare comunque un danno, il mal funzionamento di un sito web, la non raggiungibilità dello stesso.
E' importante dunque che sia previsto un continuo sistema di monitoraggio applicato all'ambiente di produzione, che consenta di monitorare continuamente i vari ambiente e poter agire di conseguenza, oltre ad avere anche una panoramica sulle prestazioni.
Attraverso appositi strumenti, è possibile analizzare in maniera automatica i dati, ottenere visualizzazioni end-to-end. È importante che il team riceva una notifica quando le infrastrutture e i servizi correlati non sono raggiungibili (down).
Per svolgere queste attività di monitoraggio, il team operation che si occupa di gestire l'infrastruttura, utilizza degli script, dei Tool ad hoc, sviluppati in azienda.

8.5.5 User Acceptance Test

Indipendentemente dalla fase di test del codice che viene eseguita in fase preliminare dal team di sviluppo e dal team operation, gli User Acceptance Test costituiscono sempre la fase di test di accettazione finale, da parte del Business, prima del deploy della soluzione in ambiente di produzione.
Per questa specifica fase di test, indipendentemente dalle sessioni ad hoc organizzate, che coinvolgono tutti i membri del team di lavoro (ICT, Web Agency, System Integrator e Business), non sono impiegati specifici Tool.
Questo potrebbe essere sicuramente un margine di miglioramento per il futuro.
I test vengono svolti:

- Per i browser più diffusi come Chrome, Firefox, Safari ed Edge.
- Su tre breakpoint diversi Desktop, tablet e mobile.
- Su sistemi operativi più utilizzati dal pubblico di riferimento del sito web: Apple: iOS e OSX, Windows e Android.

8.5.6 Application Maintenance - Analisi quantitativa del processo

Il seguente paragrafo ha come obiettivo quello di analizzare l'impatto dell'introduzione di tale metodologia, meno tradizionale e più Agile, non solo a livello delle prime fasi di sviluppo di un nuovo progetto ma sull'*Application Maintenance*.

L'Application Maintenance è una disciplina che fa parte della famiglia dei *Managed Service.* Specifica per i servizi IT, nello specifico contesto aziendale, consiste nella manutenzione correttiva ed evolutiva delle applicazioni software (i siti web) post Go Live, e ha lo scopo di tenerle sempre efficienti ed aggiornate alle esigenze di business.

In azienda è previsto un sistema di ticketing che consente di gestire le seguenti macro attività previste dal servizio:

- *Manutenzione correttiva*: correzione dei malfunzionamenti delle applicazioni.
- *Manutenzione evolutiva*: interventi mirati ad adeguare le applicazioni alle nuove esigenze di business dell'azienda, alle policy aziendali ed alla normativa.

E' importante specificare che il System Integrator che, in fase progettuale ha provveduto ad adottare alcuni principi della metodologia DevOps, sfruttando strumenti come Cloud Manager, è il medesimo team che a regime, si occupa dell'Application Maintenance.
Gli stessi metodi e principi sopra descritti, impiegati in fase di sviluppo e rilascio del codice applicativo, sono i medesimi metodi e principi adottati anche in fase di Maintenance dell'applicazione.

Per cui, a fronte della necessità di sviluppo di una nuova evolutiva piuttosto che di un intervento finalizzato alla correzione di un bug, il team di sviluppo/Maintenance adotta gli stessi strumenti e principi DevOps già descritti.

Vediamo dunque nel seguito come, l'applicazione di tali metodologie ha portato nell'ultimo periodo, a dei benefici in termini di costi, Lead Time (tempi di evasione dei ticket), qualità.

Scenario

Partendo da una base dati relativa ai volumi complessivi di siti web gestiti dal team di AMS (team che si occupa della Maintenance), nell'arco temporale 2016-2021, si evince un aumento progressivo degli stessi e di conseguenza un aumento dei volumi di siti web applicativi, da gestire. Tale crescita, nel contesto specifico, è giustificata dalla continua necessità di digitalizzare gli output verso il cliente finale (es. i cataloghi delle auto da cartacei a siti informativi).

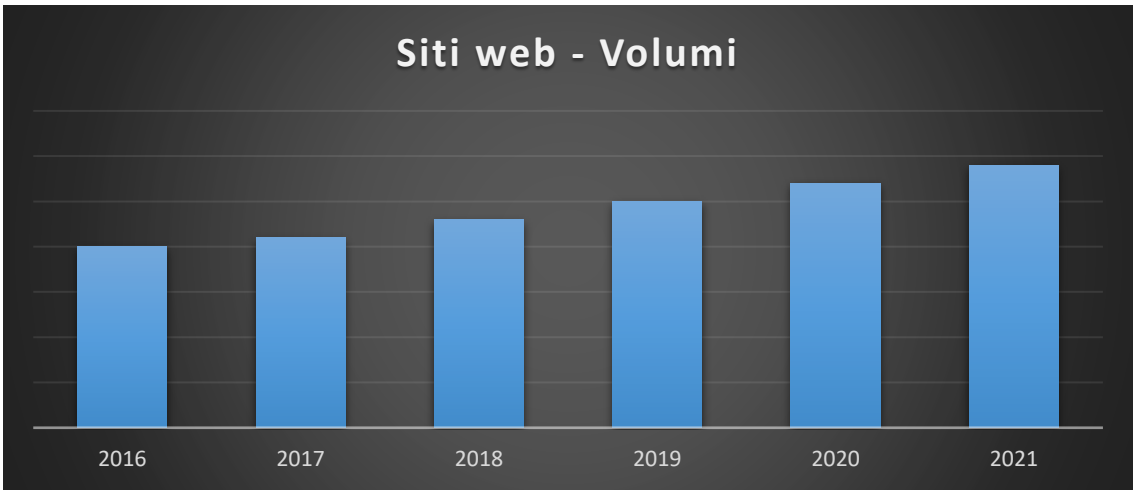


Figura 32 - Volumi Siti Web

Analizzando in parallelo, i dati relativi al *Valore del contratto di Maintenance* stipulato, notiamo che a fronte di un aumento del volume di siti web da gestire, diminuisce l'importo di tale valore.
In particolare si nota un valore contrattuale che diminuisce gradualmente, con una flessione significativa nell'ultimo anno (dal 2020 al 2021).
Sicuramente la flessione negli anni è giustificata dal fatto che, con il trascorrere del tempo, a parità di volumi, gli applicativi online da diversi anni, raggiungono di base una certa stabilità a livello di funzionamento del codice applicativo.
Viene richiesto dunque un minor effort in termini di interventi e di conseguenza si ha un saving sul servizio erogato.
D'altro canto, è importante anche considerare l'efficienza operativa.

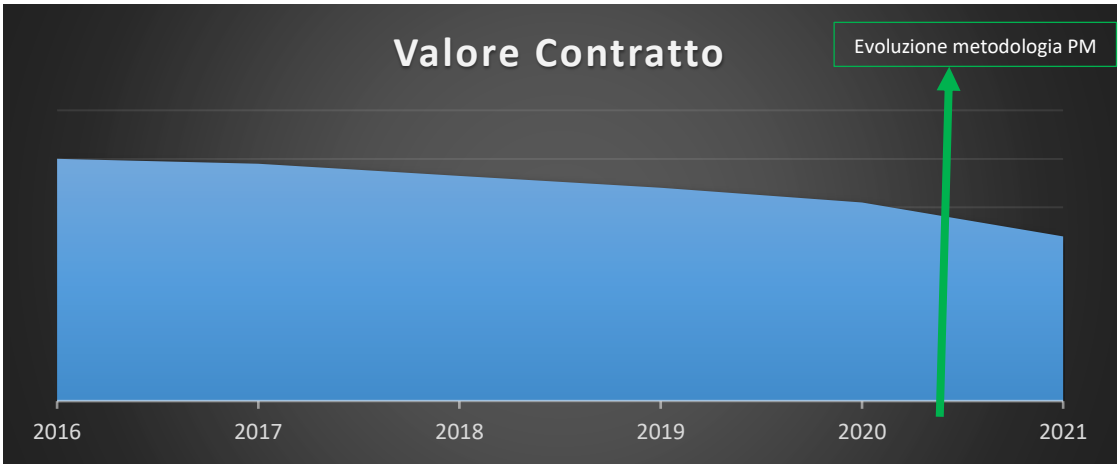


Figura 33 - Andamento valore contratto

Considerando l'ultimo anno come periodo significativo in cui l'azienda ha iniziato ad introdurre alcune tecniche DevOps orientate al Continuos Integration e Continuos Delivery, l'adozione del Cloud Manager e l'autonomia in fase di deploy, possiamo dedurre che la natura della tipologia di tale servizio, ha portato ad una costante ottimizzazione del valore contrattuale, in quanto l'efficienza del team operativo è aumentata.

Il seguente grafico mostra una netta diminuzione del valore contrattualizzato da ricondurre ad una maggior efficienza in termini di tempo e di qualità, portata dall'adozione di approcci meno tradizionali. La stima dell'ottimizzazione si può misurare tra l'intorno del 7% e 9% (evidenziato in giallo).

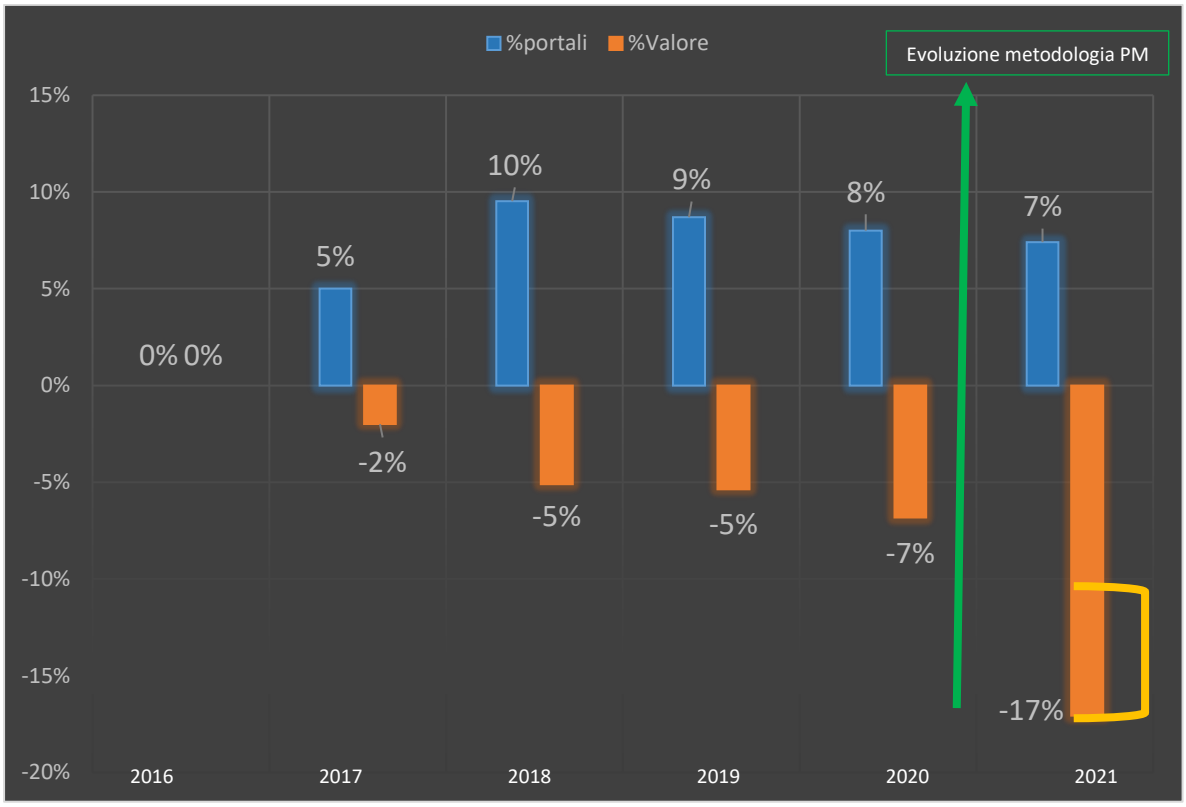


Figura 34 - Trend crescita dei portali vs trend decrescita valore contrattuale

Andando a ricercare in dettaglio le cause di tale efficienza, decido di soffermarmi su due variabili maggiormente impattate dall'adozione di metodologie più distanti da quelle tradizionali e più orientate alla metodologia Agile, ovvero:

- Lead Time
- Qualità

Lead Time e Qualità

Esistono diverse definizione del Lead Time, ma la più adatta per il seguente contesto è "la quantità di tempo che trascorre dal momento in cui il team di sviluppo/Maintenance ha una richiesta al momento che il team completi l'item." Nello specifico scenario è il tempo che intercorre da quando il team di Maintenance riceve a sistema il ticket "Assegnato al gruppo di lavoro" a quando il ticket viene evaso "Riassegnato all'utente che lo ha aperto" oppure "Chiuso".

In termini di Qualità, secondo lo standard ISO / IEC 9126, un sistema software ha sei dimensioni principali che riguardano la qualità: funzionalità, affidabilità, usabilità, efficienza, manutenibilità e portabilità.

In questo caso, ci concentriamo sull'affidabilità, che è importante perché i bug in un sistema applicativo possono portare a risultati indesiderati, come crash del sistema e/o disservizi per il cliente finale.

Dalla raccolta dei dati nel periodo 2016-2021 è significativo analizzare l'andamento delle segnalazioni/richieste evolutive, per mezzo di ticket, e dei tempi medi di risoluzione. Dal grafico seguente si può notare che con l'introduzione di nuove metodologie DevOps/Agile, vi è una proiezione di decrescita dei ticket ed una maggior velocità di risoluzione/gestione delle nuove richieste evolutive.

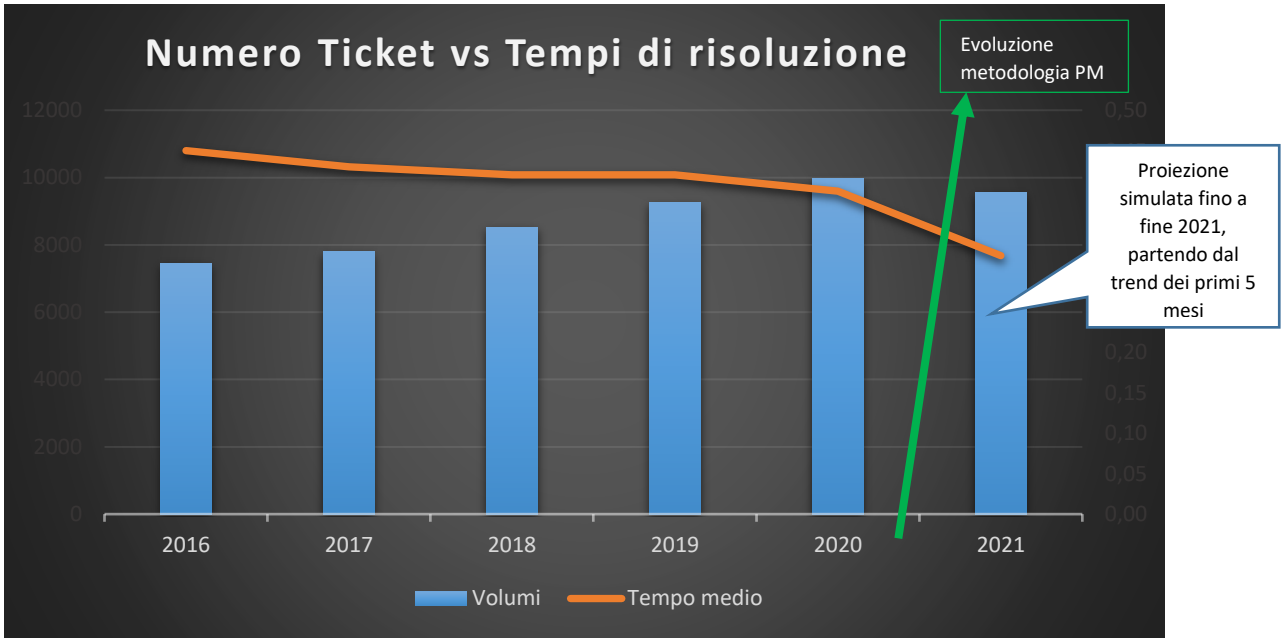


Figura 35 - #Ticket vs Tempo di risoluzione

In conclusione, i seguenti risultati/migliorie trovano riscontro nei seguenti cambiamenti apportati a livello aziendale, in termini di approccio alla gestione dei progetti:

- L'applicazione di una metodologia sempre più iterativa in fase di Design ha portato ad una maggior soddisfazione dei requisiti Business e ad un minor numero di evolutive richieste post Go Live.

- L'adozione di strumenti come il *Cloud Manager* ha consentito al team di sviluppo di essere maggiormente autonomo nella fase di delivery, accorciando i tempi. E' migliorata anche la qualità, considerando che il vendor Adobe, definisce degli "standard" da rispettare affinché gli applicativi possano essere installati.

- L'introduzione di diversi meccanismi di automazione come ad esempio l'esecuzione dei test (unit test), ha sicuramente migliorato la qualità del codice rilasciato.
- La previsione di un automatismo nella gestione dei ticket. Tale automatismo prevede di assegnare/inoltrare automaticamente un ticket al team di competenza che si occupa della risoluzione della specifica casistica. Questo automatismo ha sicuramente diminuito le tempistiche di gestione ed evasione dei ticket.

8.6 ORGANIZZAZIONE DEL TEAM

Uno dei requisiti fondamentali, affinché una metodologia possa definirsi Agile è quello di prevedere un team Cross-Funzionale.

Nello specifico caso di studio, se osserviamo l'organizzazione del team, vediamo che rimane confermata la figura del Project Manager che svolge una funzione di coordinamento e controllo.

Il Product Owner, nello specifico caso, non è identificabile in un'unica persona. Questo a causa della struttura organizzativa aziendale e in particolare al livello di approvazione della soluzione finale proposta.
Un membro dell'ente di Brand partecipa ai meeting e collabora con i fornitori e l'ente ICT durante tutta la fase di sviluppo del progetto. L'approvazione finale della UX/UI e del software, prima del finale rilascio in produzione è demandata a più persone. Spesso l'approvazione finale proviene da un capo Brand a livello di Region o a livello Global.

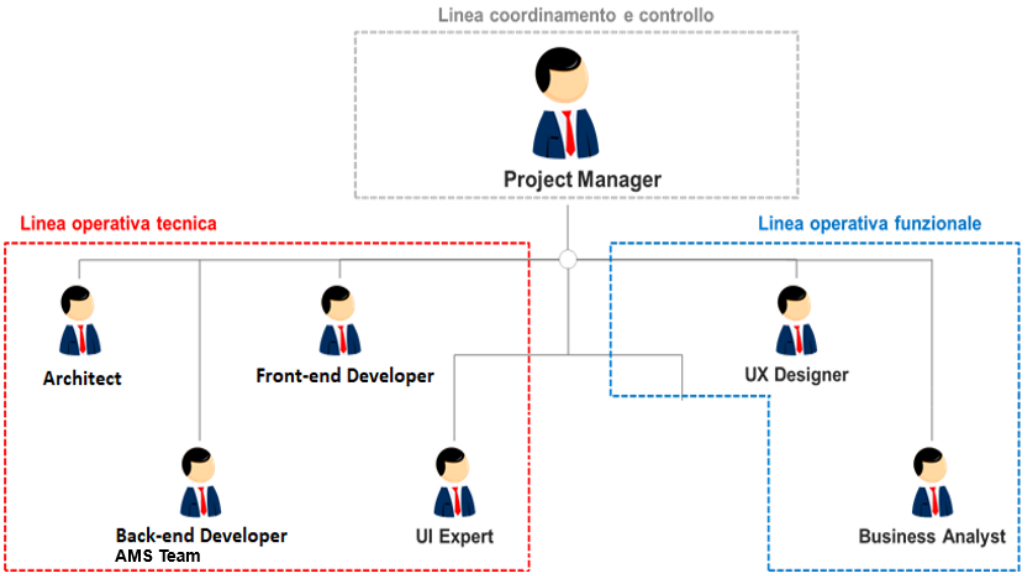


Figura 36 - Organizzazione team

Linea operativa funzionale

La linea operativa funzionale è stata costituita da risorse con approfondite competenze funzionali, analitiche e metodologiche specifiche. Per ciascun ruolo previsto dalla linea funzionale, è stato previsto di inserire alcune figure con maggior seniority, con un grado più elevato di skill ed esperienza, a ciascuna delle quali accomunate dalle figure più junior.

Di seguito sono riportate alcune delle figure professionali per cui è stato previsto il coinvolgimento nelle varie attività che hanno costituito il progetto:

Business Analyst: durante tutta la fase progettuale è stato responsabile della comprensione delle esigenze di Business, delle analisi dei dati, e ha svolto un ruolo di congiunzione tra gli analisti e le figure di business.

User Experience Designer: è stato responsabile di fornire tutti gli input necessari al gruppo di lavoro per garantire che le soluzioni implementate fossero fruibili e gradevoli all'utilizzo da parte degli utenti. Figura professionale focalizzata sul prodotto finale e sull'esperienza utente.

Linea operativa tecnica

La linea operativa tecnica è stata costituita da risorse con approfondite competenze tecnologiche ed informatiche relative alle tecnologie utilizzate.

Analogamente a quanto già riportato per la linea funzionale, anche nel caso della linea tecnica si è previsto di inserire alcune figure con maggior seniority, con un grado più elevato di skill ed esperienza, a ciascuna delle quali sono state accomunate delle figure più junior, così da garantire un corretto equilibrio nei costi.

Di seguito sono riportate alcune delle figure professionali per cui è stato previsto il coinvolgimento nelle varie attività:

User Interface Design Expert: è stato responsabile per la realizzazione di proposte volte a sviluppare l'interfaccia utente del sito web, partendo dai KPI descritti e dalle indicazioni del team UX.

Developers: membri del system integrator team che si sono occupati degli sviluppi del codice, della delivery e di interfacciarsi con il team operazione per applicare i principi di DevOps. Il medesimo team si occupa anche della Maintenance applicativa, post Go Live.

Architect: figura interna aziendale, che si è occupato della progettazione dell'architettura includendo gli ambienti AEM e i meccanismi di automazione.

8.7 VANTAGGI DEI MODELLI APPLICATI

Nel seguito riporto una sintesi dei vantaggi delle due metodologie “Waterfall” e “Agile” che ho riscontrato nel seguente caso di studio:

I vantaggi del modello Waterfall applicato a questo progetto

- ✓ Le fasi di progetto sono state chiaramente distinte sin dall’inizio
- ✓ E’ stato possibile generare documentazione del processo di sviluppo attraverso traguardi ben definiti
- ✓ Già all’inizio del progetto è stato possibile stimare il costo, la stima per la realizzazione dello stesso. Questo è un aspetto molto importante per il contesto reale in cui opera l’azienda. I tempi sono fissi, legati spesso ad un lancio di un nuovo modello del Brand e non possono essere modificati. Il budget viene approvato in fase di pre-avvio del progetto e una volta approvata da parte del Finance, non può essere modificato. Tutto ciò che viene aggiunto in corso d’opera risulta essere un extra costo difficilmente gestibile. La crisi degli ultimi anni, legati anche al fattore Covid-2019 ha giocato un ruolo molto importante su questo aspetto. I budget destinati ai progetti sono difficilmente approvati e una volta approvati sono difficilmente modificabili.
- ✓ E’ stato possibile rappresentare il progetto lungo l’asse temporale
- ✓ Non sono state stravolte le abitudini del gruppo di lavoro abituato a lavorare secondo una metodologia più tradizionale e meno Agile.

I vantaggi della metodologia Agile nella fase di Design

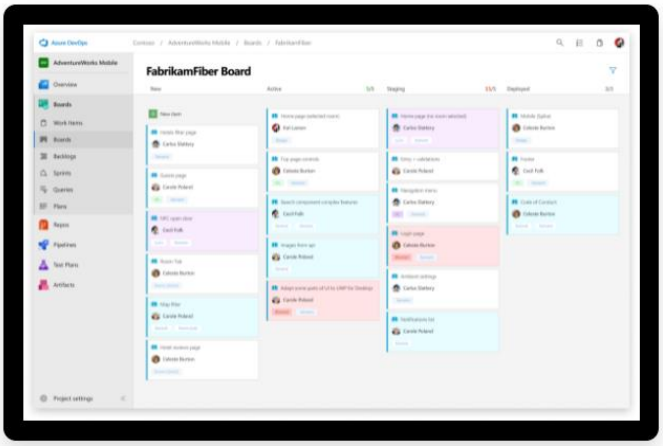
- ✓ E’ stato possibile suddividere un insieme di funzionalità all’interno di ciascun Sprint, che hanno costituito per la fase di Design, un Backlog di lavoro.
- ✓ Le attività definite all’interno di ciascuno Sprint ha definito l’insieme delle funzionalità analizzate, disegnate, condivise con il cliente e nel caso rilavorate
- ✓ Il focus è stato principalmente sul cliente e sui KPI prefissati
- ✓ E’ stata introdotta una maggiore flessibilità nell’accettare i cambiamenti in questa specifica fase di Design.
- ✓ E’ stata promossa la comunicazione tra tutti i team di lavoro
- ✓ I meeting ricorsivi hanno consentito di avere una piena consapevolezza rispetto l’avanzamento di progetto, dello stato as-is e del to-be, intercettando in tempo eventuali ritardi, problematiche e incentivandone la risoluzione mediante il confronto.

8.8 MARGINI DI MIGLIORAMENTO

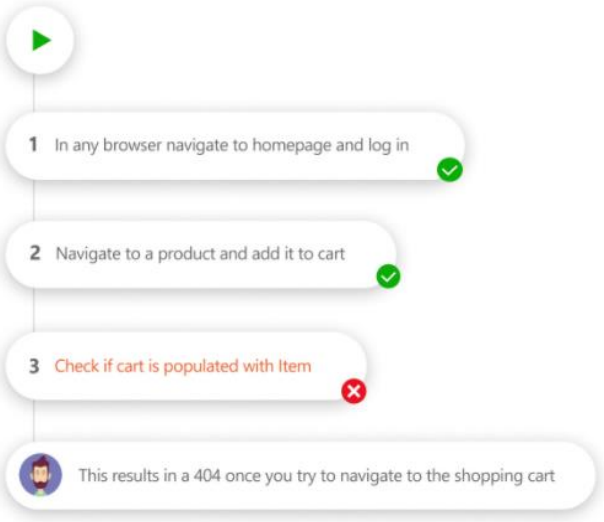
Analizzando tutti gli aspetti teorici della metodologia Agile e confrontandoli con la realtà del caso di studio, sicuramente è possibile prevedere degli aspetti migliorativi per il futuro.
Non mi soffermo tanto sul processo quanto più sugli strumenti di DevOps che il mercato offre e che potrebbero essere adottati per migliorare la gestione del progetto e le sue fasi.
Ecco qui nel seguito una lista di Tool che potrebbero essere proposti.

Azure Boards

Si tratta di dashboard che consentono di pianificare, analizzare e verificare il lavoro di diversi team.



Sono strumenti che consentono di tenere traccia del lavoro con lavagne Kanban, backlog, dashboard dei team e creazione di report personalizzati.
Combina la pianificazione di sprint con trascinamento della selezione e la verifica flessibile degli elementi di lavoro con la tracciabilità completa, per ottenere una posizione perfetta in cui salvare tutte le tue idee, grandi e piccole.
Consente di pianificare incontri informali e riunioni.
Tieni traccia di tutte le tue idee in ogni fase dello sviluppo e mantiene allineato il team in merito a tutte le modifiche del codice collegate direttamente agli elementi di lavoro (Services DevOps)



Azure Test Plans

Consente di automatizzare il processo di test sulle applicazioni desktop, mobile e tablet.
Durante l'esecuzione dei test puoi acquisire dati avanzati sullo scenario per risolvere tempestivamente i difetti individuati (Services DevOps).

9 CONCLUSIONI

Il caso di studio affrontato in questa relazione, mi consente di poter esprimere delle valutazioni finali circa l'efficacia complessiva dell'applicazione delle metodologie agili.
Se rifletto su quelle che possono essere le barriere nell'adozione di tale metodologia, ritengo che queste possano essere così classificate:

- Barriere legate al *cambiamento della cultura aziendale*
- Difficoltà nell'inserimento di metodologie Agile in un contesto/ambiente con mentalità non Agile. La gestione di progetti come quelli descritti nel caso d'uso, sono stati per moltissimi anni gestiti adottando un approccio tradizionale.
- Mancanza di persone che abbiano appreso gli skill necessari, anche solo per mezzo di esperienza durante i progetti gestiti negli anni.

Come descritto nel caso d'uso, nonostante quelle che sono le barriere, le grandi organizzazioni stanno via via affiancando a quelle che sono le metodologie di Project Management tradizionali quelle Agili.
Gestire un progetto interamente in Agile, nel contesto descritto, abbiamo visto non essere possibile per diverse motivazioni.
L'adozione però di metodologie più iterative, anche solo per alcune fasi del progetto in esame (fase di Design), unite ad alcuni principi di DevOps applicati alla fase di Development e Delivery, ha sicuramente portato ad ottenere dei risultati significativi

La collaborazione tra le persone, in particolare tra i membri del team di Web Agency, parte IT e parte business, è stata fondamentale per il successo del progetto.
Da un lato, la parte business ha molto gradito il cambio di approccio, ritenendolo un modo molto semplice ed efficace per assicurarsi il pieno soddisfacimento delle proprie esigenze e mettere la parte IT in condizione di sviluppare degli strumenti software che realmente potessero soddisfare il requisito espresso.
Dall'altro, la parte IT ha avuto la possibilità di comprendere più a fondo il dominio di business, recependo in maniera più chiara i requisiti. Inoltre, l'adozione di uno strumento come Adobe Cloud Manager e Jenkins, ha consentito loro di essere molto più veloci e autonomi nei rilasci delle soluzioni sui diversi ambienti previsti.
Il risultato è stato inedito: in precedenza, con il tradizionale approccio Waterfall e con un'organizzazione che vedeva gli sviluppatori e chi si occupava di operation come due enti totalmente distinti e non collaborativi, i tempi medi per il primo rilascio - nel caso progetti simili per dimensione e complessità - erano dell'ordine di qualche giorno.

Inoltre la comunicazione tra i diversi membri del team di lavoro ha consentito di discutere su eventuali problematiche con orientamento comune nel trovare una soluzione. Spesso questo scenario si è verificato quando, a fronte di una proposta di UX/UI la soluzione implementativa non sempre era fattibile o comunque richiedeva delle complesse lavorazioni. Spesso il consiglio da parte di uno sviluppatore ha facilitato la gestione delle complessità.

Un altro vantaggio è derivato dalla scomposizione dei complessivi requisiti di business, in pezzi più piccoli.
Le singole funzionalità organizzate in backlog e analizzate progressivamente, in collaborazione con il team di progetto, ha consentito di definire una chiara priorità tra le varie attività.

Il processo Agile ha consentito di focalizzarsi e concentrarsi sulle funzionalità più importanti in termini di Design, consentendo di definire delle priorità tra gli elementi definiti in fase di MVP. Questo ha consentito anche al Business di ridefinire le priorità durante la fase di analisi/Design. Il fatto di poter spostare rapidamente l'attenzione su alternative proposte ha portato ad un pieno soddisfacimento del cliente.

L'incentivazione della comunicazione tra team è stato un aspetto molto importante.
Il progetto ha previsto la schedulazione settimanale di due meeting della durata di un'ora ciascuno.

- Il primo meeting era finalizzato alla comunicazione/condivisione di alcuni aspetti funzionali e tecnici tra Team di UX/UI Design e Team ICT. L'obiettivo è stato quello di verificare/confrontarsi durante l'avanzamento del progetto sulle evoluzioni delle attività, consentendo di fare delle valutazioni sulla fattibilità del lavoro.
- Un secondo meeting in cui oltre al Team UX/UI Design e il team ICT, veniva coinvolto anche il Business, al fine di essere tutti allineati sull'avanzamento delle attività e per smarcare eventuali open points.

Questi meeting hanno portato a creare più discussioni eliminando il desiderio di "trattenere" idee, problemi o soluzioni. Allo stesso tempo, questi incontri brevi e frequenti sono stati più mirati e produttivi.

Possiamo dunque concludere che, la convivenza simultanea dell'approccio tradizionale con uno più Agile, è stata fattibile. La presenza di una metodologia non ha implicato l'assenza dell'altra.

L'impatto della intelligenza artificiale nel Marketing

Guardando il futuro del mercato e le evoluzioni tecnologiche, credo che un ruolo molto rilevante, anche per il caso d'uso trattato, sarà dettato

dall'introduzione e dall'utilizzo sempre più spinto dell'Artificial Intelligence (AI) e del Machine Learning.

E' un dato di fatto che l'Intelligenza Artificiale sta rapidamente diventando sempre più centrale nel mondo digitale quotidiano e il mondo del marketing e della pubblicità non fa eccezione.

L'intelligenza artificiale nel marketing consiste nell'uso dei dati dei clienti, dell'apprendimento automatico e di altri concetti computazionali per prevedere l'azione o l'inazione di una persona. Può assumere enormi quantità di dati e aiutare i professionisti del marketing a segmentarli facilmente. Pertanto, gli esperti di marketing potrebbero ulteriormente suddividere i dati per creare contenuti personalizzati per il loro pubblico. Con l'IA, le aziende possono creare ottime tecniche di analisi di marketing per indirizzare i potenziali clienti giusti. Ciò aiuterà gli esperti di marketing digitale a fornire ai clienti i contenuti giusti sul canale giusto al momento giusto.
Inoltre, un altro motivo per cui l'IA è importante è che il 76% dei clienti si aspetta che le aziende comprendano le loro esigenze e aspettative.
AIM aiuterà i marketer di contenuti a capire chi è esattamente il loro target di riferimento, creando così un'esperienza personale per ogni cliente (Mercuri, 2020).

Oltre dunque al miglioramento della proposta e dell'operato delle Web Agency, l'AI potrebbe essere impiegata anche in fase di offerta, generando ad esempio pagine web da disegni, sfruttando modelli di *deep learning*, piuttosto che per la generazione di contenuti automatici.

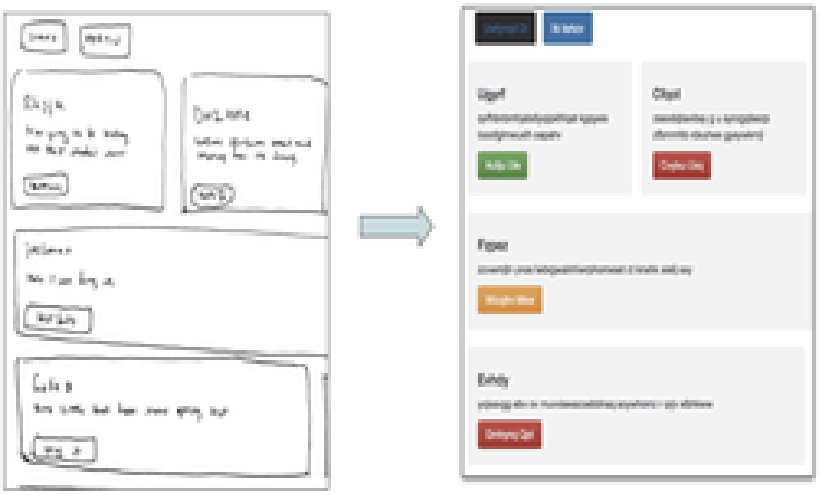


***Figura 37** - Implementazione AI su web*

In conclusione, come visto, il futuro delle aziende risiederà nella varietà e nella qualità dei servizi digitali e delle tecnologie proposte. L'offerta dovrà essere

sempre più amplificata a livello globale e per farlo sarà necessario sottostare ad una maggiore efficienza.

Le piattaforme hardware e software dovranno essere adeguatamente implementate per reggere la sfida. L'insieme di tutti i fattori descritti, contribuirà a sostenere la crescita delle aziende in ambito digitale, ma soprattutto nella cultura dell'innovazione nell'immediato futuro prossimo. Questa innovazione dovrà prevedere attività differenziate non solo di ricerca e sviluppo, ma anche soprattutto di formazione del personale e, in qualche caso, anche degli utenti. Le tipologie interessate alla trasformazione non saranno solamente di tipo operativo, ma riguarderanno anche ambiti decisionali e strategici.

10 RINGRAZIAMENTI

“Non permettere mai a nessuno di dire che non sai fare qualcosa. Se hai un sogno, tu lo devi proteggere. Se vuoi qualcosa, vai e inseguila”

Tratto da “La ricerca della felicità”

Vorrei ringraziare, innanzitutto, il Prof. Marco Torchiano per la sua disponibilità e per aver accettato di essere il relatore della mia tesi di laurea.

Ringrazio i miei genitori che mi hanno sempre sostenuto durante questi anni, in tutte le mie scelte.

Ringrazio mio marito che mi ha incoraggiato nel perseguire il sogno di completare questo percorso di studi.

Ringrazio mia figlia Diletta, la mia forza ed il mio coraggio.

11 BIBLIOGRAFIA E SITOGRAFIA

(2019, 3 21). Tratto da Digital Guide IONOS: https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/modello-a-cascata/

(2019, 3 4). Tratto da Digital Guide IONOS: https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/continuous-integration/

(2019, 05 27). Tratto da Digital Guide IONOS: https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/continuous-delivery/

Adobe. (s.d.). *Introduction to Cloud Manager*. Tratto da Adobe: https://experienceleague.adobe.com/docs/experience-manager-cloud-manager/using/introduction-to-cloud-manager.html?lang=en

Barbieri, G. (2020, 04 30). Tratto da https://www2.deloitte.com/it/it/blog/italy/2020/coronavirus---automotive---giorgio-barbieri.html

Cloud Manager. (s.d.). Tratto da Adobe I/O: https://www.adobe.io/apis/experiencecloud/cloud-manager.html

Corbucci, D. (Seconda Edizione). *Agile Project Management.* Franco Angeli.

Customer Journey. (s.d.). Tratto da Osservatori . net Digital Innovation: https://blog.osservatori.net/it_it/customer-journey-significato-mappa

Federauto. (2016, 10 19). Tratto da Federauto : https://www.federauto.eu/2016/10/era-digitale-e-cambiamenti-nel-settore-automotive-retail/

Fracasso, G. (2018, 11 19). *Cloud-nella-digital-transformation*. Tratto da Digital Leaders: https://www.digital-leaders.it/blog/cloud-nella-digital-transformation

Fracasso, G. (2018, 11 19). *Digital leaders*. Tratto da Digital leaders: https://www.digital-leaders.it/blog/cloud-nella-digital-transformation

Gonzalo, F. (2017, 3 01). *UNDERSTANDING THE DIFFERENCE BETWEEN MOBILE-FIRST, ADAPTIVE AND RESPONSIVE DESIGN*. Tratto da Frederic Gonzalo: https://fredericgonzalo.com/en/2017/03/01/understanding-the-difference-between-mobile-first-adaptive-and-responsive-design/

Kopf, B. (s.d.). *Designers*. Tratto da https://www.toptal.com/designers/ui/figma-design-tool

L'impatto del Covid-19 sull'Automotive. (s.d.). Tratto da Deloitte: https://www2.deloitte.com/it/it/blog/italy/2020/coronavirus---automotive---giorgio-barbieri.html

Mercuri, M. (2020, 01 20). Tratto da Dell Technologies: https://www.delltechnologies.com/it-it/blog/l-intelligenza-artificiale-cambia-il-futuro-della-comunicazione-e-quindi-il-nostro/

Metodologia Agile. (s.d.). Tratto da Wikipedia: https://it.wikipedia.org/wiki/Metodologia_agile

Metodologia Agile. (2021, 02 19). Tratto da Wikipedia: https://it.wikipedia.org/wiki/Metodologia_agile

Microsoft Azure. (s.d.). Tratto da Microsoft Azure: https://azure.microsoft.com/it-it/overview/what-is-devops/

Minimum_Viable_Product. (2021, 05 11). Tratto da Wikipedia: https://it.wikipedia.org/wiki/Minimum_Viable_Product

Modello a cascata. (2019, 3 21). Tratto da Digital Guide IONOS: https://www.ionos.it/digitalguide/siti-web/programmazione-del-sito-web/modello-a-cascata/

Mora, F. (s.d.). *DevOps - Guida per integrare Development e Operations e produrre software di qualità.* Apogeo.

Pittet, S. (2021). *Atlassian CI/CD*. Tratto da Atlassian CI/CD: https://www.atlassian.com/continuous-delivery/continuous-deployment

PUBLISHED, B. J. (2020, 08 12). Tratto da PM - Project Management: https://project-management.com/agile-vs-waterfall/

Quarto, F. (s.d.). *Esperienza Utente*. Tratto da WebLux: https://www.weblux.it/articolo/202/esperienza-utente-usabilita-dei-siti-web#:~:text=%22Internet%20%C3%A8%20diventato%20uno%20strumento,%C3%A8%20pi%C3%B9%20importante%20che%20mai.%22

Quarto, F. (s.d.). *Weblux*. Tratto da https://www.weblux.it/articolo/202/esperienza-utente-usabilita-dei-siti-web

Scrum-master-qualities. (s.d.). Tratto da Scrum-master-qualities: https://blog.qatestlab.com/2019/07/30/scrum-master-qualities/

Services DevOps. (s.d.). Tratto da Microsoft Azure: https://azure.microsoft.com/en-us/services/devops/

Software Development Life Cycle (SDLC). (s.d.). Tratto da Vskills: https://www.vskills.in/certification/blog/software-development-life-cycle-sdlc/

Sviluppo Software. (2020, 02 6). Tratto da Network Digital 360: https://www.zerounoweb.it/software/sviluppo-software/architetture-a-microservizi-e-container-perche-puntare-sullapproccio-devops/

Waterfall. (s.d.). Tratto da Smartsheet: https://www.smartsheet.com/content-center/best-practices/project-management/project-management-guide/waterfall-methodology

what-is-sprint-retrospective. (s.d.). Tratto da what-is-sprint-retrospective: https://appinventiv.com/blog/what-is-sprint-retrospective/

Zanotti, L. (2020, 12 11). *Network Digital 360*. Tratto da Network Digital 360: https://www.zerounoweb.it/techtarget/searchdatacenter/programmazione-software-e-l-ora-degli-it-shop-devops/

12 INDICE FIGURE