Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Gestionale
A.a. 2023/2024
Sessione di Laurea luglio 2024

Adozione del piano DevOps nella Pubblica Amministrazione

Relatori:
Prof.re Torchiano Marco

Candidato:
Scannicchio Stefano

SOMMARIO

CAPITOLO 1 – INTRODUZIONE

I Sistemi Informativi rappresentano un insieme integrato di strumenti volti alla raccolta, memorizzazione ed elaborazione dei dati, al fine di fornire informazioni pertinenti e tempestive alle persone interessate. Numerose imprese e organizzazioni si avvalgono di tali sistemi per condurre le proprie attività principali, collaborare con fornitori e clienti, gestire bilanci, coordinare risorse umane e mantenere competitività nel mercato di riferimento. In generale, i Sistemi Informativi costituiscono un insieme di strumenti essenziali per il funzionamento aziendale. La presente tesi si propone di delineare una parte delle attività svolte presso Accenture, un'azienda di consulenza strategica e direzionale operante a livello globale. In particolar modo, il lavoro di tesi è frutto di un'esperienza di tirocinio all'interno di un team di lavoro che si interfaccia quotidianamente con clienti appartenenti alla Pubblica Amministrazione. Difatti, l'obiettivo principale del lavoro descritto in questa tesi è quello di offrire supporto a tali clienti, "trasferendo" all'interno del settore pubblico la metodologia DevOps, descritta dettagliatamente nei capitoli che seguono. Si inizia con una panoramica su argomenti di ampio respiro, quali i Sistemi Informativi, il settore della Pubblica Amministrazione, la differenza tra la metodologia "Waterfall" e quella "Agile", per poi approfondire temi più specifici come le fasi del ciclo di vita di un software. Dopo un'ampia sezione introduttiva di natura teorica, sarà presentato un caso di studio.

1.1 Obiettivi

Questa tesi ha come obiettivo principale l'analisi approfondita dell'adozione della metodologia DevOps all'interno della pubblica amministrazione, con l'intento di evidenziare i benefici, le sfide e le best practice associate a tale trasformazione. Gli obiettivi specifici includono:

- **Esaminare lo stato attuale:** Analizzare lo stato attuale dei processi di sviluppo e gestione IT nella pubblica amministrazione, identificando criticità e aree di miglioramento
- **Valutare l'impatto del DevOps:** Valutare l'impatto dell'implementazione di pratiche DevOps su efficienza, qualità e sicurezza dei servizi pubblici, utilizzando casi di studio e drivers di valutazione
- **Identificare le Best Practice:** Individuare le best practice per un'adozione efficace della metodologia DevOps, considerando le peculiarità e le esigenze specifiche della pubblica amministrazione
- **Analizzare le sfide:** Esplorare le principali sfide e resistenze che possono emergere durante l'adozione di DevOps, proponendo soluzioni e approcci per superarle.

Raggiungendo questi obiettivi, la tesi mira a fornire un contributo alla comprensione e all'implementazione efficace della metodologia DevOps nella pubblica amministrazione, promuovendo un cambiamento positivo nei processi operativi.

In particolare, il lavoro di tesi verrà organizzato nella seguente modalità:

- Nel secondo capitolo saranno definiti e analizzati i Sistemi Informativi e le componenti che li costituiscono;
- Nel terzo capitolo verrà definita la Pubblica Amministrazione e la trasformazione digitale che la caratterizza negli ultimi anni;
- Nel quarto capitolo verrà descritto l'approccio "Agile", e come questo possa aiutare a gestire in maniera più efficiente le dinamiche mutevoli dei progetti software. Tale approccio verrà paragonato con la metodologia "Waterfall", ampiamente utilizzata in passato. In questo capitolo, verranno analizzate dettagliatamente queste due metodologie e, per ognuna di esse, le fasi che le costituiscono, i vantaggi e gli svantaggi;
- Nel quinto capitolo, sarà presentato il DevOps, tema portante di questo lavoro di tesi. Inoltre, verrà fornita dall'autore una descrizione della mentalità e della cultura che si

celano dietro questa pratica, nonché delle funzioni e dei pilastri su cui la metodologia si basa;

- Nel sesto capitolo, verrà presentato il caso applicativo. Si fornirà una panoramica dettagliata sulle fasi in cui si divide il lavoro svolto, le esigenze di business, le metodologie seguite, le tecnologie utilizzate e i risultati raccolti grazie a mirati drivers di valutazione.

1.2 Attività svolte

Con riferimento al sesto capitolo, riguardante il caso applicativo, ho potuto prestare il mio contributo nell'esecuzione di diverse attività. Queste hanno riguardato prevalentemente:

- Redazione di documenti che forniscono delle linee guida per l'amministrazione nell'ambito del ciclo di vita del software;
- Raccolta e formalizzazione dei requisiti funzionali e non funzionali;
- Validazione dei requisiti formalizzati;
- Progettazione delle specifiche funzionali partendo dai requisiti definiti;
- Selezione della modalità di lavoro;
- Riprogettazione infrastrutturale dell'infrastruttura abilitante (in collaborazione con un team di progettisti software) in base alle specifiche esigenze del cliente
- Swot Analysis sulla creazione dell'infrastruttura IT.

CAPITOLO 2 – I SISTEMI INFORMATIVI

2.1 Definizioni ed esempi

I Sistemi Informativi (SI) sono insiemi organizzati di risorse, procedure, reti di comunicazione, dati software e hardware che lavorano insieme per raccogliere, elaborare, archiviare e distribuire informazioni all'interno di un'organizzazione. Le singole componenti elencate in precedenza verranno trattate singolarmente in seguito, per fornire un livello maggiore di dettaglio. Questi sistemi hanno lo scopo di supportare le attività operative, manageriali e decisionali di un'organizzazione attraverso l'uso efficiente delle risorse informative. I sistemi informativi possono essere divisi in diverse categorie in base alla loro funzione e al livello di un'organizzazione in cui vengono utilizzati. Alcuni esempi comuni includono:

- **Sistemi di elaborazione delle transazioni (TPS)**: Gestiscono le transazioni quotidiane e registrano le operazioni di base all'interno di un'organizzazione, come ad esempio ordini, pagamenti e registrazioni di inventario
- **Sistemi di supporto alle decisioni (DSS)**: Forniscono supporto ai manager durante il processo decisionale fornendo informazioni analitiche e strumenti di modellazione
- **Sistemi informativi di gestione (MIS)**: Sono progettati per fornire informazioni di livello medio ai manager per il monitoraggio delle prestazioni e la pianificazione a medio termine
- **Sistemi Enterprise resource planning (ERP)**: Integrano diversi processi aziendali come finanza, risorse umane e gestione della catena di approvvigionamento, in un unico sistema.
- **Sistemi di supporto operativo (OSS)**: Gestiscono le attività operative quotidiane, tra cui l'elaborazione delle transazioni, la gestione delle risorse e la comunicazione interna
- **Sistemi di informazione geografica (GIS)**: Gestiscono e analizzano dati geografici per supportare la pianificazione e le decisioni basate sulla posizione.

L’obiettivo principale di un sistema informativo è quello di migliorare l’efficienza operativa e supportare la presa di decisioni all’interno di un’organizzazione, fornendo informazioni accurate e tempestive.

2.2 Componenti di un Sistema Informativo

Un Sistema informativo efficiente integra e coordina efficacemente tutte le sue componenti (esplicate di seguito) per raggiungere gli obiettivi organizzativi, migliorando l’efficienza operativa e supportando il processo decisionale.

2.2.1 Risorse

Gli utenti e il personale che utilizzano, gestiscono e mantengono il sistema informativo sono parte integrante del sistema. Ciò include gli sviluppatori software, gli amministratori di database, gli utenti finali e gli altri professionisti IT.

È necessario, però, operare un’ulteriore distinzione nel momento in cui si parla di risorse in relazione ad un sistema informativo. Queste, infatti, possono essere anche fisiche o logiche. Le principali risorse fisiche solitamente presenti in un sistema informativo moderno, sono calcolatori (server e postazioni di lavoro), periferiche (scanner, stampanti, modem), apparecchiature di rete, i locali che ospitano le apparecchiature e gli impianti di alimentazione e condizionamento di tali locali. Vale la pena osservare che anche gli operatori umani possono essere considerati per certi aspetti come risorse fisiche. Le risorse logiche presenti in un sistema informativo moderno sono tipicamente i moduli del software applicativo e del software di base (DBMS, Sistema Operativo, Software di Rete), le basi di dati, i registri di configurazione dei dispositivi (per esempio router). Anche in questo caso, possono essere considerate come risorse logiche l'insieme delle regole organizzative comportamentali degli operatori umani.

2.2.2 Procedure

Le procedure definiscono le regole e i processi che guidano l'uso del sistema informativo. Ciò include le procedure di inserimento dati, i processi decisionali, altri flussi di lavoro organizzativi e le procedure di sicurezza. Queste ultime garantiscono la protezione dei dati sensibili e la continuità operativa del sistema informativo. Ciò include il controllo degli accessi, la crittografia dei dati e la gestione delle minacce alla sicurezza. Per quanto concerne i processi nello specifico, è importante saper riconoscere quali sono i processi chiave, per distinguere quelli centrali che contribuiscono al valore del profitto finale e quelli di supporto all'organizzazione. In questa prospettiva, è possibile classificare i processi in tre categorie distinte:

- Processi primari: questi processi sono da considerarsi i più cruciali perché sono quelli che apportano un valore tangibile ai consumatori finali
- Processi di supporto: anche se non apportano valore diretto al prodotto o servizio finale, questi processi sono indispensabili per il corretto funzionamento dei processi primari
- Processi di gestione: sono focalizzati sul monitoraggio e controllo delle attività centrali, sebbene non aggiungano valore direttamente al consumatore finale (Torchiano M., Corno F., Ardito L., "Sistemi Informativi Aziendali", 2022)

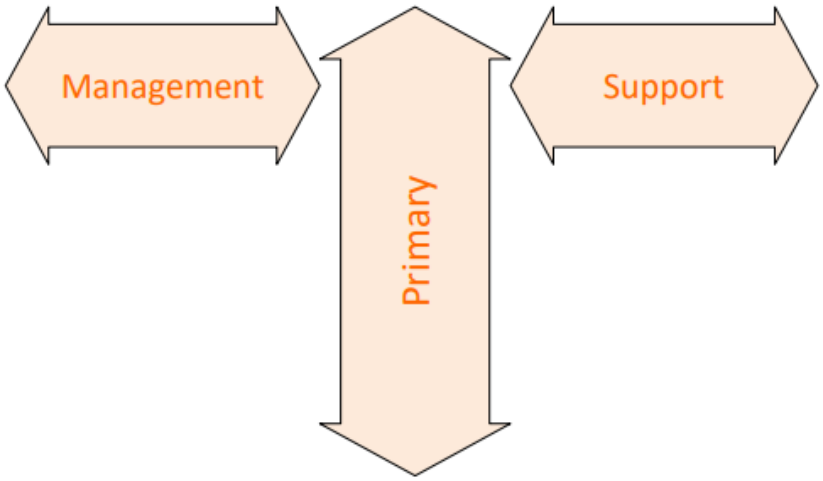


Figura 1 - Modello "a T" dei processi [1]

2.2.3 Reti di comunicazione

I primi sistemi informativi si sono sviluppati come applicazioni monolitiche, ossia singole applicazioni direttamente integrate con il sistema operativo. Nel corso del tempo, hanno subito un'evoluzione, trasformandosi in sistemi multilivello che sfruttavano basi di dati per archiviare informazioni. L'introduzione dei livelli logici distinti ha aperto la possibilità di facilitare la collaborazione tra diversi sistemi informativi. Per rendere ciò possibile, sono stati introdotti diversi strati di comunicazione, ognuno caratterizzato dal livello logico in cui opera. Vediamo, adesso, come diversi SI possono comunicare tra loro:

- **Comunicazione diretta**. L'integrazione delle applicazioni potrebbe essere realizzata attraverso la creazione di sistemi separati che stabiliscono una comunicazione diretta tra di loro. In pratica, è necessario definire un'appropriata interfaccia per ciascuna coppia di sistemi informativi che devono interagire. Si dovranno stabilire regole specifiche di comunicazione per ogni coppia di sistemi informativi, insieme ai relativi moduli di comunicazione. Questo approccio, storicamente il primo adottato, risulta notevolmente complesso da gestire, pur consentendo la definizione di interfacce specifiche tra i vari sistemi, il che potrebbe rivelarsi vantaggioso in determinate circostanze
- **Integrazione tramite middleware.** Un middleware facilita la comunicazione tra moduli attraverso lo scambio di messaggi con regole specifiche. Questo processo virtualizza la comunicazione, ma può garantire affidabilità con la gestione di ritrasmissioni. Tuttavia, presenta limitata flessibilità, richiedendo adattatori per uniformare i formati dei dati tra i sistemi. Il middleware può anche supportare sistemi di notifica tramite un message broker
- **Integrazione tramite workflow.** Un'altra forma di integrazione coinvolge l'uso di sistemi di workflow, impiegati sia per la comunicazione interna, che esterna nelle aziende. Questo tipo di integrazione comporta la codifica delle attività che compongono i processi aziendali, suddividendo l'azienda in macro-processi secondo lo schema della catena del valore di Porter. Si stabiliscono collegamenti tra i processi

e verso l'esterno, specificando i dati scambiati e i passi necessari per l'integrazione. Dal punto di vista tecnologico, si utilizza l'approccio di modellazione attraverso BPMN (Business Process Modeling Notation), con un livello di astrazione generalmente alto.

Nel corso degli anni, si è passati da reti centralizzate, in cui le risorse si trovavano tutte in uno stesso ambiente IT, a sistemi distribuiti in cui le risorse sono distanti tra loro. Le principali reti informatiche di cui usufruiamo ormai quotidianamente sono, infatti, reti distribuite:

- reti distribuite localmente (LAN – Local Area Network)
- reti distribuite geograficamente (WAN – Wide Area Network)
- reti metropolitane (MAN – Metropolitan Area Network)
- reti di accesso: ultimo segmento che collega l'utente con la centrale più vicina
- reti backbone: collegamento tra le varie centrali (Torchiano M., Corno F., Ardito L., "Sistemi Informativi Aziendali", 2022)

Consentendo non solo il collegamento a Internet e l'uso condiviso di server, ma anche la condivisione di dati tra più computer, le reti informatiche rappresentano lo strumento principe della comunicazione digitale. Ne esistono di diversi tipi, in base all'estensione e allo strumento di trasmissione utilizzato. E che si tratti di gestire pochi computer all'interno di un'abitazione privata, oppure decine o centinaia di una struttura aziendale, la comunicazione tra questi dispositivi, oggi, è assolutamente possibile.

2.2.4 Dati

I dati sono sequenze di fatti semplici, rappresentanti eventi che si verificano all'interno di un'organizzazione o nel mondo fisico, prima di essere organizzati in una forma comprensibile e utilizzabile. Un singolo numero o una sequenza di numeri costituisce un esempio di dato. È fondamentale distinguere i dati dalle informazioni, poiché queste ultime sono dati che hanno acquisito una forma significativa per gli esseri umani all'interno dei processi organizzativi, come nel caso delle decisioni. Ad esempio, un numero interpretato come misura di un fenomeno rilevante per il funzionamento dell'organizzazione rappresenta

un esempio di informazione, come nel caso di 23 confezioni di prodotto ordinate. Pertanto, il dato è un elemento conosciuto, un'informazione grezza o elementare ed è solitamente costituito da simboli che devono essere elaborati e contestualizzati. Il dato di per sé è un numero, un testo, un segnale video o audio; si tratta di una semplice segnalazione che ha sicuramente un valore proprio, ma può assumere significati differenti a seconda del contesto in cui si trova. All'interno del sistema informativo, la collezione di dati è chiamata base di dati o database. Compito della base di dati non è solo quello di memorizzare i dati, ma anche di rappresentare le relazioni tra di essi. La memorizzazione-archiviazione del dato, deve essere quindi attentamente valutata e attribuita a un suo contesto, e quindi organizzata: è importante saper valutare il database e conoscerne i concetti di base, così che i dati memorizzati possano poi essere analizzati in informazioni vere e in grado di offrirci i dovuti vantaggi. All'interno del sistema informativo, il software atto specificatamente a gestire i dati è detto Sistema di Gestione della base di dati o Database Management System (Torchiano M., Corno F., Ardito L., "Sistemi Informativi Aziendali", 2022).

2.2.5 Software

Il termine "software" si riferisce ad un insieme di procedure, istruzioni e documentazione che indicano al computer come comportarsi e consentono agli utenti di eseguire diverse operazioni. Essenzialmente, si tratta di un gruppo di programmi che gestiscono il funzionamento del computer, realizzati sia in linguaggio macchina, eseguibile direttamente dal computer, sia in un linguaggio di programmazione di alto livello. Esempi di software includono applicazioni come Microsoft Word, Photoshop, Google Chrome, nonché sistemi operativi come Windows e OS. I software si suddividono principalmente in due categorie: software di sistema e software applicativi. I software di sistema comprendono i sistemi operativi che controllano direttamente l'hardware, gestendo le operazioni dell'unità centrale e controllando schermo, stampanti, supporti di memoria e altro ancora. D'altra parte, i software applicativi sono progettati per svolgere funzioni specifiche, consentendo agli utenti di eseguire operazioni come scrittura, gestione di dati, calcolo, elaborazione di immagini.

Esempi di software applicativo includono Microsoft Word, Excel, Paint e MySQL. Il software è un programma sviluppato tramite un linguaggio di programmazione. Un programma è composto da una serie di istruzioni logiche che permettono al computer di compiere operazioni utili per l'utente finale. In generale, l'utente interagisce con le applicazioni software attraverso il desktop di un computer o lo schermo di un dispositivo portatile. L'applicazione software comunica con il sistema operativo, il quale, a sua volta, comunica con l'hardware.

2.2.6 Hardware

L'hardware rappresenta la parte fisica e tangibile di un sistema informativo. Fanno parte di esso:

- Periferiche di input e di output. Tutti i dispositivi che mettono in comunicazione il computer con l'esterno sono detti genericamente periferiche oppure dispositivi periferici di input/output. Alcuni dispositivi sono solo di ingresso perché inviano dati al computer e non li ricevono mai (come il mouse e la tastiera), altri sono solo di uscita perché ricevono dati dal computer senza inviarne mai (come il monitor e le casse audio), altri sono contemporaneamente di ingresso e di uscita (come i dischi).
- Computer
- Dispositivi di storage (CD, Hard Disk, ecc.)
- Dispositivi di telecomunicazione (VPN, internet, ecc.).

2.3 Requisiti

Un sistema informativo, inoltre, dovrebbe anche presentare determinati requisiti in modo da poter fornire un servizio efficace:

- **Velocità di risposta alle richieste dell'utente**: per ottenere tempi di risposta più brevi è essenziale implementare più server in parallelo

- **Scalabilità**: questa caratteristica del sistema consiste nella capacità di gestire contemporaneamente un elevato numero di utenti. La spesa operativa è proporzionale al numero di dispositivi che possono utilizzare il sistema informativo simultaneamente
- **Disponibilità**: definita come la percentuale di tempo per cui il sistema è accessibile senza interruzioni
- **Affidabilità**: rappresenta la capacità del sistema di operare anche in presenza di guasti. Poiché l'interruzione dei processi comporta spesso costi, l'obiettivo è di risolvere i guasti senza interrompere il servizio, evitando che i clienti ne siano consapevoli.

Questi fattori hanno contribuito al successo del cloud computing. Utilizzando questa tecnologia, gli utenti pagano una tariffa in relazione alla scalabilità desiderata e alla velocità di risposta necessaria. Il cloud computing consente agli utenti di regolare agilmente il numero di server utilizzati, garantendo che gli investimenti siano limitati al periodo di necessità, a differenza del modello tradizionale di affitto dei server che richiede un dimensionamento basato sul picco e presenta una minore flessibilità e maggiore inerzia al cambiamento.

2.4 Livelli e funzioni

I sistemi informativi possono essere classificati in macroaree. Tale classificazione si opera nella "Piramide di Anthony", il cui schema di riferimento ha due dimensioni: il livello organizzativo e la funzione aziendale.

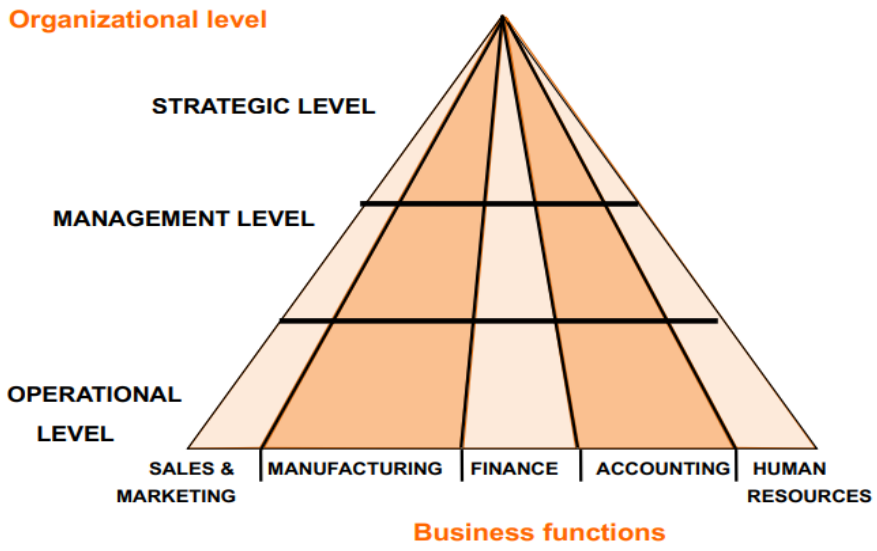


Figura 2 - Piramide di Anthony [1]

La prima dimensione (dal basso verso l'alto) riguarda i livelli aziendali [1].

Operativo: questo livello gestisce le attività quotidiane necessarie per far progredire l'organizzazione. Caratterizzato da un'elevata frequenza di operazioni e notevoli quantità di dati, si occupa di transazioni ed eventi, ricevendo la denominazione generale di Transaction Processing Systems (TPS) per i sistemi informativi che operano a questo livello

Gestionale: questo livello ha a che fare con le attività di gestione dei processi svolti a livello operativo ed è caratterizzato da una frequenza di attività più lenta e da informazioni più sintetiche fornite ai manager, spesso derivanti dall'aggregazione dei dati utilizzati a livello operativo e dagli indicatori

Strategico: questo livello gestisce le attività di indirizzo strategico dell'organizzazione o delle sue parti. Caratterizzato da un periodo di lavoro decisamente più lungo rispetto agli altri

livelli, fornisce quantità di indicatori ancora più ridotte e informazioni più astratte ai manager di alto livello.

La seconda dimensione, su cui si basa la piramide, rappresenta le diverse funzioni aziendali. Per ogni intersezione funzione-livello, è possibile identificare una serie di processi aziendali e tipologie di sistemi informativi che li supportano. La tecnologia ha progressivamente permesso l'automazione dei livelli operativi per aumentare l'efficienza e il controllo. Anche i livelli superiori sono stati influenzati dall'IT, poiché molte delle elaborazioni che una volta erano compiute a livello di management, ora vengono gestite da computer o business intelligence, producendo dati su cui il management baserà le proprie decisioni. L'automatizzazione riguarda principalmente le parti ripetitive delle attività gestite dai manager. Questa rappresentazione a piramide, sebbene utile, presenta una limitazione nel posizionare tutte le funzioni sulla stessa scala, non considerando le differenze di specificità tra le funzioni aziendali, alcune delle quali sono più specifiche per un'azienda, mentre altre sono più trasversali e applicabili a diverse realtà aziendali.

CAPITOLO 3 – LA PUBBLICA AMMINISTRAZIONE

3.1 Definizioni

La Pubblica Amministrazione è l'insieme delle istituzioni e degli organi che svolgono funzioni pubbliche nell'ambito di uno Stato o di una collettività. La sua missione principale è di gestire e fornire servizi pubblici per il bene comune garantendo il funzionamento ordinato della società e il soddisfacimento delle esigenze della collettività. La Pubblica Amministrazione è responsabile di attività quali la regolamentazione, la normativa, la gestione delle risorse pubbliche, la sicurezza, la giustizia, l'istruzione e la sanità, agendo nell'interesse generale e nell'osservanza del quadro giuridico stabilito. La sua struttura può variare in base al sistema di governo e la forma di Stato e la sua efficacia è spesso misurata in termini di trasparenza, efficienza e capacità di rispondere alle esigenze della società.

3.2 Struttura e tipologie

La Pubblica Amministrazione si caratterizza per la sua struttura gerarchica di tipo piramidale. Al vertice, detiene il massimo potere con il Governo e i ministri, mentre procedendo verso la base, si incontrano sfere di potere sempre meno estese. La sua organizzazione comprende sia organi centrali, come i ministeri che esercitano le loro funzioni su tutto il territorio nazionale, sia organi periferici. Quando le responsabilità amministrative sono assegnate direttamente agli organi di Stato, si parla di amministrazione diretta. Diversamente, l'amministrazione indiretta è affidata ad enti pubblici distinti, come province e regioni.

In generale, la Pubblica Amministrazione può essere suddivisa in amministrazione centrale e amministrazione locale. Si fa riferimento all'amministrazione centrale quando le attività di competenza sono condotte da uffici situati nella capitale e detengono l'autorità su tutto il territorio nazionale. Sono considerati parte della Pubblica Amministrazione centrale i ministeri e gli organi centrali dello Stato. Al contrario, l'amministrazione periferica o locale è

caratterizzata dalla presenza di sedi locali con autorità territoriale limitata. Rientrano nella Pubblica Amministrazione locale le regioni, le province e i comuni.

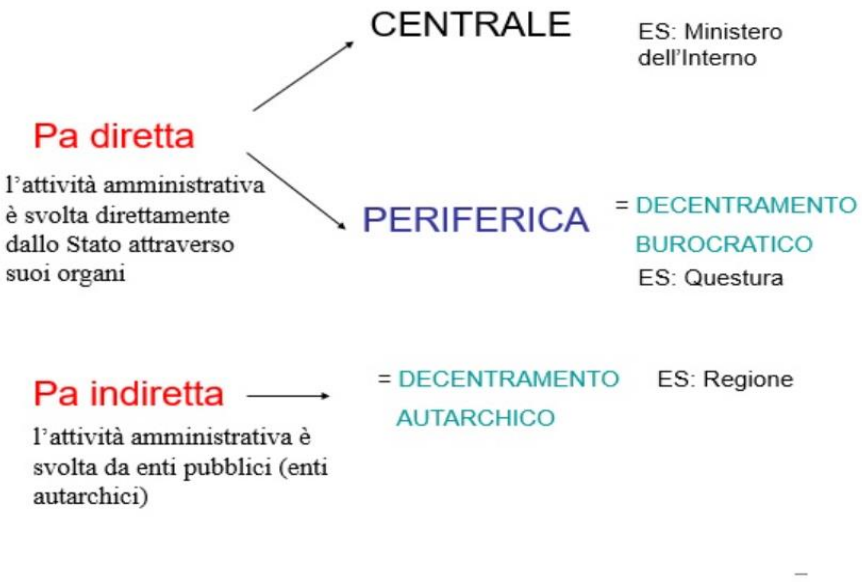


Figura 3 - Tipologie di Pubblica Amministrazione [2]

In base al tipo di attività svolta, distinguiamo le PA nelle seguenti tipologie:

ATTIVA	Realizza in modo effettivo e concreto i fini pubblici. Le funzioni svolte dai ministeri sono prevalentemente di tipo attivo
CONSULTIVA	Quando ha come fine la redazione di pareri, ossia di indicazioni o valutazioni di natura giuridica, tecnica ed economica. L'attività è svolta da appositi organismi

DI CONTROLLO	Si occupa di riesaminare l'attività posta in essere da altri organi, al fine di verificarne l'esattezza e la correttezza. Un esempio di tale tipologia è rappresentato dalla Corte dei Conti.

Tabella 1 - Tipologie di Pubblica Amministrazione

Il fenomeno in cui le tecnologie ICT vengono impiegate dalla Pubblica Amministrazione per migliorare l'efficacia e l'efficienza, nonché per raggiungere una maggiore trasparenza e semplificazione dei servizi offerti alla collettività, è comunemente chiamato con il termine "E-Government".

3.3 L'E-Government

Il termine "E-Government" si riferisce genericamente all'utilizzo di tecnologie innovative nei processi amministrativi delle pubbliche amministrazioni, finalizzato a fornire servizi ai cittadini. La diffusione rapida di Internet e delle nuove tecnologie di rete ha portato ad una prevalente erogazione online dei servizi. Numerosi esempi illustrano come le nuove tecnologie consentano la creazione di sportelli virtuali, equiparabili ai tradizionali sportelli fisici, che variano da siti web istituzionali a portali territoriali. Tuttavia, è importante sottolineare che l'e-government va oltre la mera informatizzazione della pubblica amministrazione. Il termine è corretto solo quando l'impiego delle tecnologie innovative contribuisce chiaramente al miglioramento dei servizi finali offerti agli utenti (cittadini e imprese) e alla qualità della vita democratica di un Paese (Jeong, 2007). In linea con questa definizione, sia l'OCSE che la Commissione Europea concordano sull'e-government come l'uso delle nuove tecnologie dell'informazione e della comunicazione (ICT) da parte delle pubbliche amministrazioni, con il potenziale di trasformare le strutture e le procedure amministrative, nonché di aumentare la partecipazione al processo democratico.

3.3.1 E-Government – Il cambiamento nella Pubblica Amministrazione

L'impatto delle nuove tecnologie sui processi amministrativi della PA deve quindi essere quello di migliorare il rapporto con i cittadini e le imprese in termini di:

- Maggiore efficienza dell'atto amministrativo
- Maggiore trasparenza dei procedimenti
- Maggiore rapidità nel compimento dei compiti istituzionali e nella risposta alle richieste dei cittadini
- Maggiore adattamento ai bisogni degli utenti ed alla richiesta di servizi innovativi.

In questa prospettiva, appare fondamentale sia la centralità dell'utente finale (cittadino o impresa), sia la consapevolezza organizzativa che i processi di cambiamento rapidi e intensi, quali quelli che scaturiscono dall'introduzione di tecnologie innovative nella macchina della pubblica amministrazione, possano essere avviati solo a partire dal front-end delle organizzazioni, cioè dal luogo dove più fortemente l'organizzazione percepisce la pressione della domanda di servizi da parte dei suoi clienti. È evidente che l'introduzione di soluzioni di e-government, con l'apertura di sportelli online di accesso ai servizi, produce un cambiamento non solo nelle modalità di interazione tra cittadino e PA, ma anche nell'organizzazione del funzionamento interno della PA stessa. Pertanto, devono essere riorganizzati in termini tecnologici i processi di back-office che preparano l'erogazione online del servizio finale. Solo un back-office funzionante consente l'erogazione efficace di servizi e tale riorganizzazione deve essere sempre orientata al miglioramento del servizio all'utente finale, cittadino od impresa. Il cambiamento che così scaturisce, guidato dall'esterno, in alcuni casi viene contrapposto ad un tipo di cambiamento della pubblica amministrazione che "parte dall'interno" (tipico dei processi di informatizzazione interna avvenuto nell'ultimo decennio) motivato da esigenze di efficienza interna, organizzativa ed economica. La riorganizzazione del processo di back-office, inoltre, riveste notevole importanza quando coinvolge non solo più uffici della stessa amministrazione, ma più amministrazioni, prevedendo adeguati strumenti di cooperazione tra i processi ed i servizi dei diversi enti per erogare servizi in modo omogeneo. Lo scopo della riorganizzazione

interna delle amministrazioni pubbliche deve infatti essere quello di mostrare al cittadino la PA come un'unica entità omogenea, con la quale interagire attraverso uno sportello online virtualmente unificato, a prescindere da qualsiasi vincolo geografico, organizzativo e di accesso. In questo modo, l'onere della conoscenza e dell'adeguamento ai bisogni, si sposta dal cittadino all'amministrazione, che, sulla base delle esigenze dei cittadini, organizza il proprio back-office e la propria offerta di servizi. Effettivamente, il successo delle iniziative di e-government appare legato alla capacità di organizzare l'offerta di servizi innovativi online sulla base delle esigenze dei cittadini e di scegliere strumenti e modalità di erogazione che si adattino quanto più possibile alle necessità ed alle peculiarità di tutti i possibili utenti. Solo in questo caso, gli utenti di un servizio saranno propensi ad utilizzare il canale di erogazione innovativo piuttosto che uno dei canali di erogazione tradizionali.

A questo riguardo, occorre in conclusione tener presenti le diverse variabili che possono influenzare l'uso, da parte dei cittadini, dei servizi di e-government realizzati dalle amministrazioni pubbliche. Esse riguardano:

1. L'individuazione dei servizi di maggiore utilità per gli utenti (focalizzazione sui servizi prioritari)
2. L'eliminazione dei cosiddetti "falsi servizi", ossia di servizi "generati dal produttore", ma di scarsa utilità per gli utenti finali
3. Il superamento del divario tecnologico che può rendere difficile, se non impossibile l'uso dei servizi online per particolari categorie di utenti
4. La garanzia di accesso universale ai servizi erogati online
5. L'individuazione di canali alternativi di erogazione dei servizi, per consentire ad ogni cittadino di scegliere il canale di erogazione che più si adatta alle sue peculiarità e ai suoi bisogni
6. Il rafforzamento dell'infrastruttura di sicurezza e di tutela della privacy, laddove l'interazione richieda l'inserimento, da parte dei cittadini, di dati sensibili o informazioni confidenziali[1]

[1] Tratto da http://qualitapa.gov.it/sitoarcheologico/www.urp.it/sito-storico/www.urp.it/allegati/6_e-government.pdf

3.4 La trasformazione digitale nella Pubblica Amministrazione

L'adozione delle tecnologie dell'informazione e della comunicazione (ICT) da parte della pubblica amministrazione è avvenuta con un certo ritardo rispetto alle imprese e le organizzazioni private. Tuttavia, soprattutto negli ultimi anni, numerosi governi internazionali hanno riconosciuto le molteplici potenzialità di queste tecnologie e hanno iniziato a ridurre questo gap tecnologico aumentando gli investimenti nel settore delle ICT. La riduzione di questo divario è favorita principalmente dalla costante diminuzione dei costi delle tecnologie informatiche, specialmente per quanto riguarda la componente hardware. La trasformazione digitale della pubblica amministrazione è un obiettivo prioritario nei prossimi anni. Un obiettivo che porterà alla scomparsa della carta, alla riduzione dei costi e all'erogazione di servizi molto più efficienti e utili per il cittadino. L'intero processo, sicuramente complesso, si basa su questi quattro punti chiave:

- Gli attori del processo
- Il piano triennale per l'informatica nella PA
- Il modello strategico di evoluzione
- Digital transformation

3.4.1 Gli attori del processo

Gli enti e i soggetti coinvolti nel processo di trasformazione digitale della pubblica amministrazione sono:

1. Governo
2. Dipartimento della Funzione Pubblica
3. Ministero dell'Economia e delle Finanze
4. Commissario per l'attuazione dell'Agenda digitale
5. Comitato di indirizzo dell'AgID
6. L'AgID (Agenzia per l'Italia Digitale)
7. Amministrazioni regionali e Province autonome

8. Tutte le amministrazioni
9. Società in house
10. Enti strumentali
11. Società Consip e centrali di committenza.

Le funzioni di ciascun attore e i rapporti che intercorrono tra tutti i soggetti coinvolti sono rappresentati nella figura di seguito:

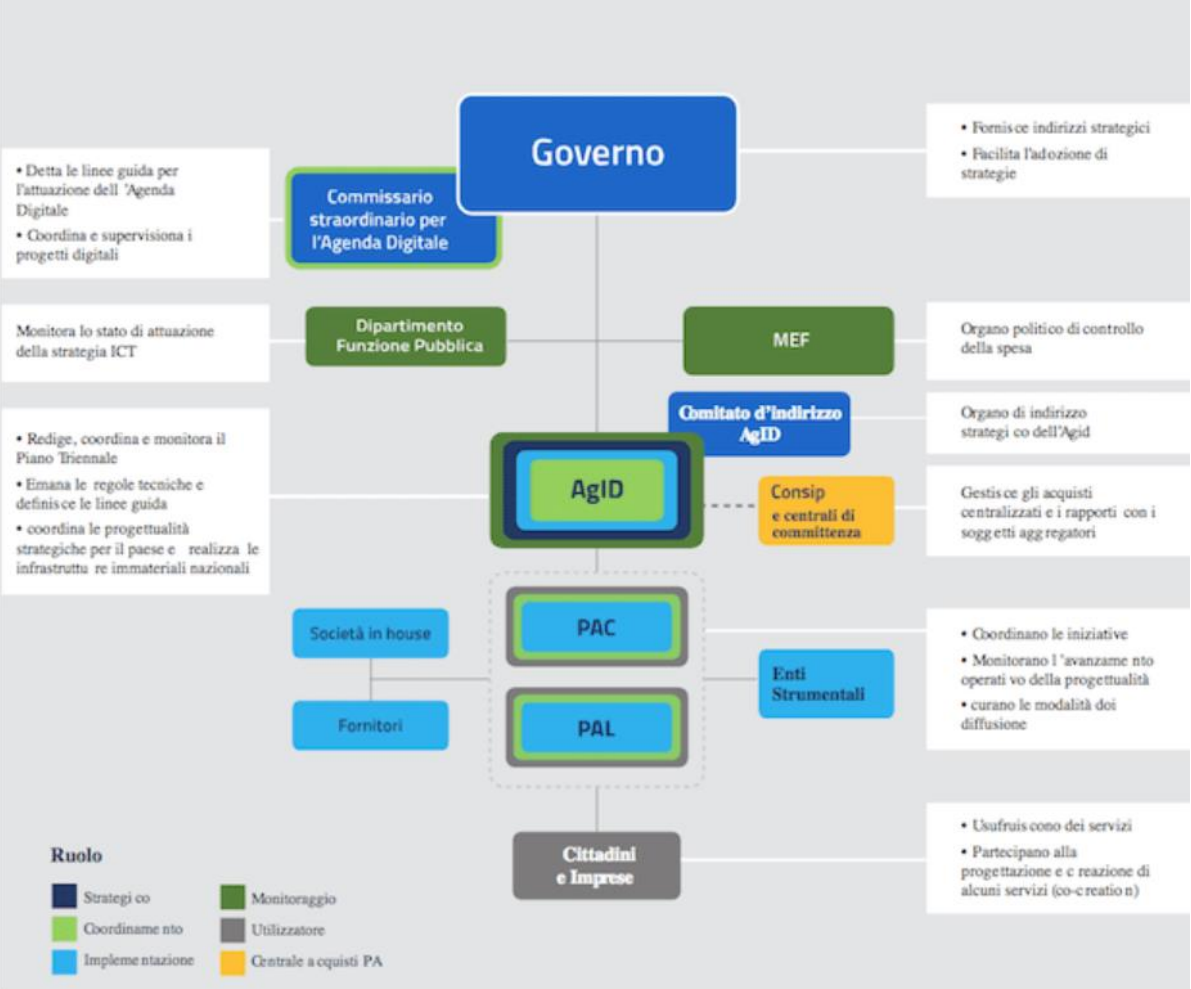


Figura 4 - Attori del processo di trasformazione digitale della Pubblica Amministrazione [3]

3.4.2 Il Piano triennale per l'informatica nella Pubblica Amministrazione

Il Piano triennale è uno strumento di pianificazione al quale ministeri ed enti pubblici devono contribuire e fare riferimento al fine di raggiungere l'obiettivo della completa informatizzazione della pubblica amministrazione. Nella tabella di seguito sono elencati una serie di punti chiave per riuscirci.

INTEROPERABILITA'	Interazione e scambio di informazioni
GESTIONE DEI DATI	Strumenti per gestire in maniera efficiente la grande mole di dati della Pubblica Amministrazione
INFRASTRUTTURE FISICHE	Le infrastrutture necessarie per una corretta ed efficiente condivisione dei dati
SERVIZI DIGITALI	Gli strumenti necessari per raggiungere gli obiettivi del Piano triennale
SICUREZZA	Chi è preposto alla sicurezza dei dati e cosa deve fare per garantirla

Tabella 2 - Punti chiave per la completa informatizzazione della Pubblica Amministrazione

Il piano triennale per l'informatica nella pubblica amministrazione prevede che il concetto di **interoperabilità** debba essere esplicato in questo modo: implementare dei sistemi informatici efficienti che garantiscano la piena interazione e la massima facilità nello scambio di informazioni all'interno della pubblica amministrazione.

Grazie all'interoperabilità sarà possibile:

- Uniformare le architetture informatiche
- Promuovere l'utilizzo di procedure condivise.

Un altro aspetto chiave è la **gestione dei dati**. La pubblica amministrazione ogni giorno produce e gestisce una gran mole di dati e informazioni. Per sfruttare appieno le grandi

potenzialità occorre fare in modo che tali dati siano gestiti in maniera più sistemica. Il Piano triennale individua tre grandi aree:

- Basi di dati di interesse nazionale. Queste devono: contenere informazioni verificate, valide e veritiere; essere omogenee per tipologie di dati contenuti; contenere informazioni importanti per lo svolgimento delle attività della pubblica amministrazione
- Open data. I dati gestiti dalla pubblica amministrazione dovranno essere liberamente usabili e riutilizzabili da tutti i soggetti che hanno accesso a quei determinati dati
- Vocabolari controllati e modelli dei dati. Due strumenti che concorrono a creare quel sistema in cui tutti i dati sono organizzati e condivisi con modalità stabilite e omogenee.

Se fino a questo momento si è parlato di best practice dal punto di vista gestionale, adesso si entra più nello specifico: quali sono gli strumenti concreti che permetteranno di raggiungere gli obiettivi del Piano triennale? Le **infrastrutture fisiche** individuate sono:

- Data center. Tratta di un ambiente nel quale sono allocate più infrastrutture ICT. Nella pubblica amministrazione ci sono svariati data center in funzione e la loro gestione rappresenta circa il 39% della spesa ICT annuale nelle PA. Questa spesa è riducibile e valorizzando i data center che funzionano meglio e accorpando quelli simili
- Cloud. Il Piano triennale prevede che venga realizzato un ambiente cloud omogeneo e condiviso in tutta la pubblica amministrazione, che renderà molto più semplice condividere e modificare documenti e informazioni
- Connettività. I due punti precedenti non possono prescindere dall'avere una connessione stabile e potente. Inoltre, vanno previsti strumenti come antivirus e firewall in materia di sicurezza dei dati.

Per quanto concerne i **servizi digitali**, affinché vengano raggiunti i macro-obiettivi e i micro-obiettivi stabiliti dal Piano triennale per l'informatica nella pubblica amministrazione, l'AgID (Agenzia per l'Italia digitale) individua e mette a disposizione cinque strumenti:

- Repository del codice sorgente

- Catalogo delle API con documentazione, ambienti di test e sandbox
- Documentazione tecnica
- Strumenti di service design
- Strumenti per la gestione di progetto.

L'ultimo punto chiave riguarda la **sicurezza dei dati**. A tal fine, il soggetto individuato è il CERT-PA (Computer Emergency Readiness/Response Team). Fa parte dell'AgID e il suo compito è quello di aiutare le pubbliche amministrazioni nel prevenire e nell'affrontare possibili minacce informatiche tramite l'adozione di:

- Cyber Security Knowledge Base (un database dove sono raccolte le informazioni sulle infrastrutture tecnologiche della PA e sugli eventi di sicurezza verificatisi in tali contesti)
- National vulnerability database (una di tutte le possibili vulnerabilità informatiche nella PA)
- Un sistema di prevenzione di possibili problemi e malfunzionamenti informatici[2]

3.4.3 Il modello strategico di evoluzione

Il Modello Strategico di evoluzione del sistema informativo della pubblica amministrazione è il quadro di riferimento all'interno del quale si innestano i progetti e le attività specifiche da realizzare per raggiungere gli obiettivi del Piano triennale. Di seguito è riportato ciò che prevede tale modello.

[2] Tratto da https://www.bucap.it/news/trasformazione-digitale-pubblica-amministrazione.htm

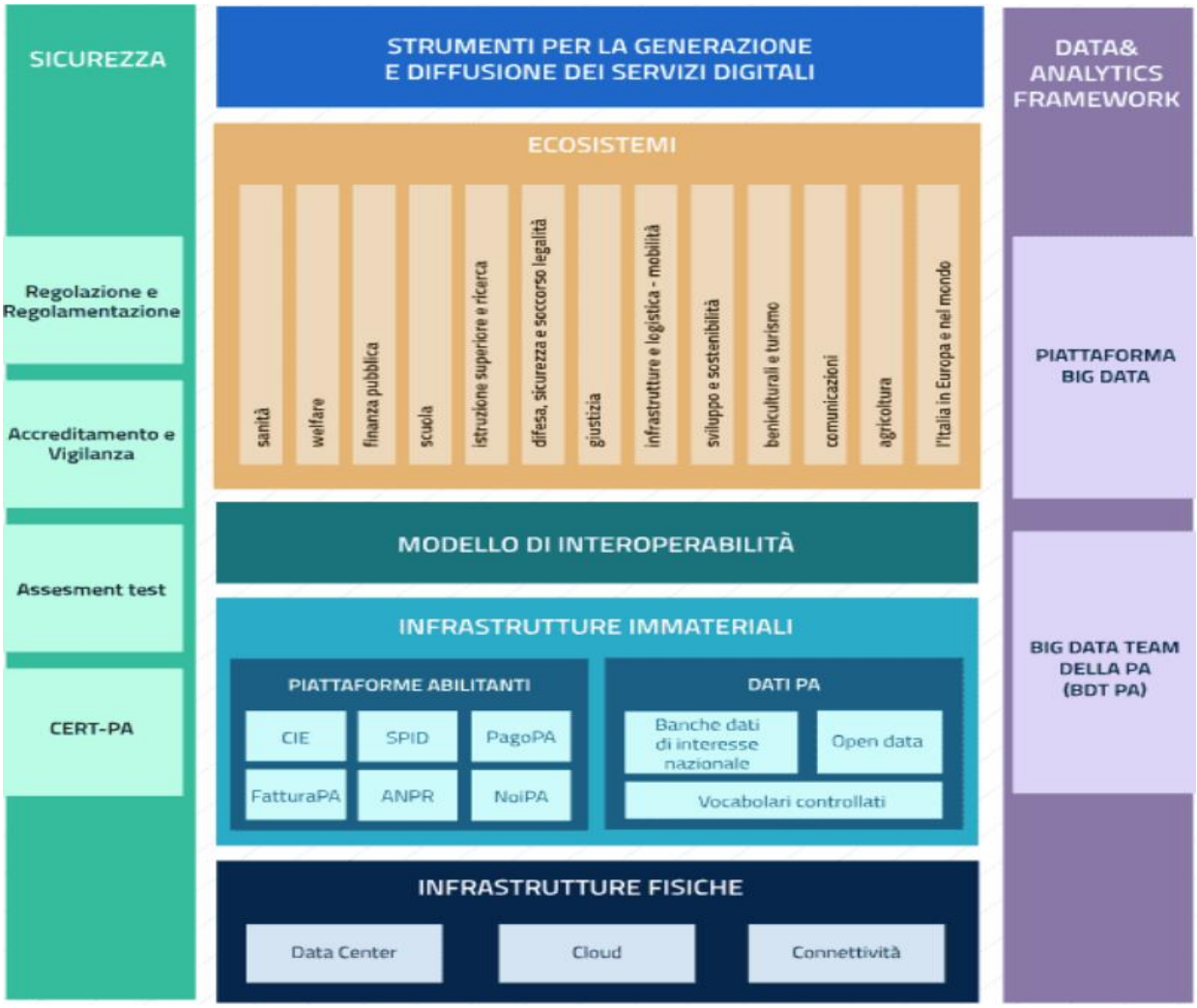


Figura 5 - Modello strategico di evoluzione [3]

In questa immagine sono riassunti i concetti principali indicati in precedenza e il quadro generale in cui devono essere inseriti. Gli scopi e le funzioni del Modello strategico di evoluzione sono: facilitare il coordinamento tra le pubbliche amministrazioni, consentire lo sviluppo di servizi digitali utili per il cittadino, agevolare l'informatizzazione della pubblica amministrazione, promuovere soluzioni che riducano i costi e migliorino i servizi. Gli attori coinvolti nel processo di trasformazione digitale della pubblica amministrazione devono organizzare il proprio operato in base a quanto stabilito dal modello.

3.4.4 Il piano nazionale di ripresa e resilienza (PNRR)

Il processo di digitalizzazione delle pubbliche amministrazioni riveste un ruolo fondamentale all'interno del piano nazionale di ripresa e resilienza. La digitalizzazione, insieme all'innovazione e alla sicurezza nel contesto della pubblica amministrazione, costituisce una delle tre componenti principali della Missione numero 1 del Piano, denominata "Digitalizzazione, innovazione, competitività e cultura". Con la legge di bilancio 2020 sono state previste diverse misure volte a promuovere e valorizzare l'informatizzazione della pubblica amministrazione. In seguito, è stato emanato il decreto-legge 1° marzo 2021, n.22[3], il quale, oltre a ridefinire le competenze di alcuni ministeri, ha influito sulle responsabilità del Governo riguardanti l'innovazione tecnologica la transizione digitale. Tale decreto prevede che il Presidente del Consiglio promuova, indirizzi e coordini l'azione del governo su diverse tematiche, tra cui la strategia italiana per la banda ultra-larga, la digitalizzazione delle pubbliche amministrazioni e delle imprese, nonché le infrastrutture digitali, sia materiali che immateriali.

Nel Piano nazionale di ripresa e resilienza, viene attribuito un notevole spazio alla digitalizzazione, sia nel contesto della pubblica amministrazione che nell'ambito del sistema produttivo. La prima missione del PNRR, intitolata "digitalizzazione, innovazione, competitività e cultura", si pone come obiettivo generale "l'innovazione del Paese in chiave digitale, affinché si avvii un autentico cambiamento strutturale". Essa contempla diverse aree di intervento:

- Digitalizzazione e modernizzazione della pubblica amministrazione
- Riforma della giustizia
- Innovazione del sistema produttivo
- Implementazione della banda larga
- Investimenti nel patrimonio turistico e culturale

[3] Parlamento italiano, Piano Nazionale di Ripresa e Resilienza, decreto-legge 1° marzo 2021, n. 22, recante disposizioni urgenti in materia di riordino delle attribuzioni dei Ministeri"

Le azioni della missione sono articolate in tre componenti principali: digitalizzazione, innovazione e sicurezza nella PA; digitalizzazione, innovazione e competitività del sistema produttivo; turismo e cultura 4.0.

CAPITOLO 4 - LA METODOLOGIA AGILE

In questo capitolo verrà approfondito il modello a cascata (Waterfall), per poi analizzare, in base al contesto evolutivo degli ultimi anni in merito al ciclo di vita di un progetto (estendibile al campo informatico, come nel caso in questione), la diffusione dilagante del nuovo approccio, denominato "Agile".

4.1 La metodologia Waterfall

Il metodo Waterfall (in italiano "a cascata"), originariamente presentato dal Dr. Winston W. Royce in un articolo del 1970 (sebbene non fosse ancora denominato così), rappresenta un modello classico nel project management per la gestione del ciclo di vita del progetto. Quando l'ingegneria e lo sviluppo del software hanno iniziato ad integrare metodi di gestione dei progetti, il modello Waterfall è emerso come uno dei primi approcci adottati. Questo modello adotta un approccio lineare e sequenziale. L'approccio "Waterfall", rappresenta una delle prime metodologie di sviluppo software adottate nell'industria. Questo modello sequenziale organizza il processo di sviluppo in una serie di fasi ben definite, in cui ogni fase successiva dipende strettamente da quella precedente. La linearità di questo approccio implica che il passaggio da una fase all'altra avvenga in modo unidirezionale e che ogni fase debba essere completata prima di procedere alla successiva, quindi, non vi è alcuna sovrapposizione tra le fasi. Sebbene il modello Waterfall abbia dimostrato la sua efficacia in progetti con requisiti stabili e ben compresi, presenta alcune limitazioni nell'affrontare cambiamenti dei requisiti durante il ciclo di sviluppo, in quanto tali modifiche richiedono spesso un ritorno alle fasi precedenti. La rigidità del modello Waterfall ha stimolato l'evoluzione di metodologie più flessibili, come l'approccio Agile, che mirano a gestire in maniera più efficiente le dinamiche mutevoli dei progetti software.

4.2 Le fasi del modello Waterfall

Tale modello, che è conosciuto anche come "Linear Sequential Life Cycle Model" si basa su una sequenza lineare di fasi ben definite nello sviluppo del progetto, tra cui l'analisi dei requisiti, il design, lo sviluppo (implementazione), il collaudo (testing), il rilascio (l'installazione) e, eventualmente, la manutenzione. Inoltre, nel contesto di questa metodologia, il prodotto viene consegnato al cliente al termine dell'intero processo.

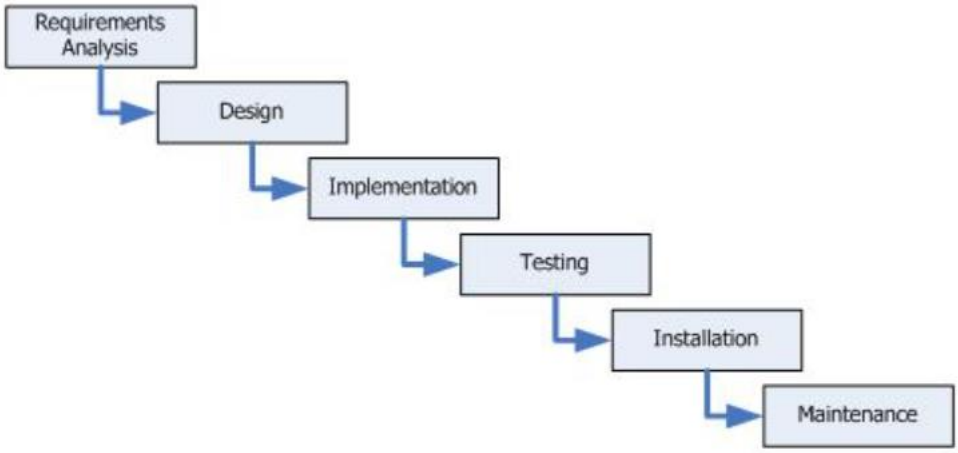


Figura 6 - Fasi del modello Waterfall [4]

Nei paragrafi seguenti, sarà analizzata in dettaglio ciascuna delle fasi del ciclo di vita di un software, seguendo l'applicazione della metodologia Waterfall.

4.2.1 Analisi dei requisiti

La fase di raccolta dei requisiti costituisce una tappa cruciale nel ciclo di vita del software, in quanto implica l'identificazione delle funzioni necessarie al software, la definizione dell'interazione degli utenti con esso e la gestione dei dati che dovrà affrontare. Il successo o il fallimento del progetto sono spesso determinati in questa fase. Se gli ingegneri del software non comprendono in modo accurato i requisiti degli utenti, il software rischia di non soddisfare le loro esigenze. Gli obiettivi principali della fase di raccolta dei requisiti includono la comprensione delle necessità degli utenti, la definizione di requisiti funzionali

e non funzionali, la creazione di un piano per il software e la minimizzazione dei rischi di fallimento del progetto. Le attività svolte in questa fase comprendono interviste agli utenti, osservazioni dirette, raccolta formale dei requisiti, stesura della documentazione dei requisiti e la pianificazione dettagliata del software. Sebbene questa fase possa richiedere tempo e sforzi considerevoli, è cruciale per garantire il successo complessivo del progetto.

Al fine di assicurare una raccolta dei requisiti efficace, è consigliabile:

- Coinvolgere gli utenti chiave
- Utilizzare diverse tecniche di raccolta per ottenere informazioni approfondite
- Documentare accuratamente i requisiti
- Ottenere l'approvazione dei requisiti dagli utenti per garantire l'allineamento con le esigenze reali.

L'analisi dei requisiti costituisce il fondamento su cui si baseranno le fasi successive del progetto.

4.2.2 Design

La fase di progettazione costituisce la terza tappa nel ciclo di vita del software, durante il quale il team di sviluppo elabora un progetto dettagliato del software da realizzare. Questo progetto deve comprendere tutti gli aspetti del software, quali l'architettura, la struttura dei dati, le interfacce utente e i test.

Gli obiettivi principali della fase di progettazione sono:

- Creare un progetto dettagliato del software
- Definire l'architettura, la struttura dei dati, le interfacce utente e i test del software
- Ottenere l'approvazione del progetto da parte dei responsabili.

Solitamente, la fase di progettazione è affidata ad un architetto del software, un professionista specializzato nella progettazione di sistemi software. Le attività svolte in questa fase comprendono riunioni con i responsabili per discutere i requisiti del progetto, la

redazione di un documento di progettazione dettagliato e l'ottenimento dell'approvazione delle parti coinvolte. I consigli per garantire il successo della fase di progettazione sono simili a quelli della fase di pianificazione iniziale. È cruciale coinvolgere i responsabili del progetto, monitorare costantemente eventuali cambiamenti e assicurarsi di ottenere l'approvazione necessaria da parte delle figure chiave coinvolte.

4.2.3 Implementazione

La fase di implementazione costituisce una delle fasi cruciali nel ciclo di vita di un software. Dopo aver definito gli obiettivi e completato la progettazione, questa fase si concentra sulla effettiva scrittura del codice sorgente sulla trasformazione delle specifiche in un'applicazione funzionante. Durante l'implementazione, gli sviluppatori traducono il linguaggio di progettazione in una serie di istruzioni precise e comprensibili al computer. Sfruttando linguaggi di programmazione come Java, C++, Phyton o altri. I programmatori compongono il codice che delineerà il funzionamento del software. L'implementazione richiede solide competenze tecniche e una comprensione approfondita del linguaggio di programmazione utilizzato. Gli sviluppatori devono attenersi agli standard di codifica, organizzare il codice in modo coerente e adottare le migliori pratiche di programmazione per garantire l'efficienza, l'affidabilità e la manutenibilità del software. In questa fase possono essere impiegati strumenti di sviluppo, come gli ambienti di sviluppo integrati (IDE) per agevolare la scrittura del codice e la rilevazione degli errori. Gli sviluppatori possono altresì servirsi di librerie di codici predefiniti o framework per accelerare il processo di sviluppo e sfruttare soluzioni comuni. Dopo aver redatto il codice, gli sviluppatori conducono una serie di test per verificare la corretta esecuzione del software. Test unitari, test di integrazione e test di sistema vengono impiegati per individuare eventuali errori o bug nel software e correggerli prima del rilascio. Durante l'implementazione è fondamentale documentare il codice in modo esaustivo per agevolare la manutenzione futura e favorire la comprensione da parte di altri sviluppatori. La documentazione dovrebbe includere commenti nel codice, istruzioni di utilizzo ed eventuali requisiti di configurazione o dipendenze.

4.2.4 Testing

La fase di testing nel ciclo di vita del software artificiale costituisce il processo finalizzato all'individuazione e correzione degli errori. Normalmente, un team di tester del software si occupa di condurre questo processo, impiegando una gamma di tecniche volte a rilevare difetti presenti nel codice, nella progettazione e nell'implementazione del software. Gli obiettivi principali della fase di test includono:

1. Indicare correggere gli errori presenti nel software
2. Garantire che il software sia conforme ai requisiti specificati
3. Verificare che il software sia intuitivo nell'uso e affidabile
4. Assicurare che il software rispetti gli standard prestabiliti.

La fase di test è suddivisa in diversi momenti poiché ciascuno di essi si concentra su un aspetto specifico del software. Le fasi più comuni di test includono:

1. **Unitario**: questo tipo di test verifica le singole unità di codice, come funzioni o metodi
2. **Di integrazione**: qui si prova come diverse unità di codice interagiscono tra di loro
3. **Di sistema**: questo test valuta come il software interagisce con l'hardware e i software esterni
4. **Di accettazione**: in questa fase, gli utenti finali testano il software per garantire che esso soddisfi i loro requisiti.

La fase di test non si limita solo all'individuazione degli errori, ma può anche contribuire all'ottimizzazione del software. Analizzando i risultati dei test, gli sviluppatori possono identificare aree di miglioramento, ottimizzare le prestazioni, migliorare l'usabilità e correggere eventuali difetti.

Al fine di seguire le best practice, è essenziale:

- Pianificare attentamente il processo di test
- Utilizzare una varietà di tecniche
- Coinvolgere gli utenti finali
- Documentare accuratamente i risultati

- Risolvere rapidamente gli errori.

È fondamentale sottolineare che i test dovrebbero essere eseguiti in modo rigoroso e sistematico, utilizzando piani di test definiti in precedenza e criteri di accettazione ben definiti. Inoltre, i test dovrebbero essere ripetuti ogni volta che vengono apportate modifiche al software, garantendo che le nuove funzionalità non introducano nuovi problemi.

4.2.5 Installazione

Una volta fatti tutti i test necessari e raggiunto un efficace livello di qualità, il software è pronto per essere installato. Tale fase è conosciuta anche come “deployment”.

Il concetto di “andare in produzione” assume varie sfumature a seconda del tipo di software in questione. Nel contesto dei programmi destinati alla vendita o distribuzione, inclusi quelli gratuiti, la produzione segna il momento in cui il prodotto viene rilasciato sul mercato. Nel caso di software realizzato su misura per un cliente, questa fase corrisponde all'installazione e al collaudo presso la sede del cliente. Infine, per le web application, il processo di produzione coinvolge l'installazione, il collaudo sul web server e l’ottimizzazione di quest’ultimo. Da questo momento inizia la vita operativa del software, durante la quale esegue le funzioni per cui è stato progettato e sviluppato.

4.2.6 Manutenzione

Per garantire il continuo e affidabile funzionamento del software nel corso del tempo, è essenziale procedere a periodici aggiornamenti offrendo un supporto tempestivo e costante agli utenti che lo utilizzano quotidianamente.

Inoltre, nel corso della vita operativa di un software in produzione, potrebbero essere necessari interventi correttivi o di aggiornamento sull'applicazione, i quali potrebbero coinvolgere nuove fasi di progettazione, sviluppo e test. Questi interventi possono essere suddivisi in due categorie principali:

1. **Manutenzione ordinaria**: comprende interventi correttivi che si rendono necessari a causa di errori eventualmente trascurati durante i test o generati dall'utilizzo del programma in condizioni non previste durante la fase di progettazione
2. **Manutenzione correttiva**: in questo contesto, gli interventi mirano a modificare o arricchire il software con nuove funzionalità, giustificate da nuove esigenze operative.

Gli obiettivi principali della fase di manutenzione sono quelli di mantenere il software in uno stato di buona salute, rispondere alle esigenze degli utenti finali e assicurare che il software sia conforme agli standard. La fase di manutenzione è generalmente condotta da un team di ingegneri del software, i quali impiegano diverse tecniche al fine di migliorare e correggere il software. Alcune delle tecniche più comuni di manutenzione comprendono: correzione di bug (errori nel software), aggiornamento di funzionalità, miglioramento delle prestazioni e aggiornamento della sicurezza del software per proteggerlo dalle minacce.

Le fasi appena delineate, come menzionato precedentemente, sono associate al modello di sviluppo a cascata. Questo modello è ampiamente diffuso e adattabile a diverse tipologie di progetti, ma presenta talvolta una limitata flessibilità: le fasi devono essere affrontate in modo sequenziale, rendendo complesso retrocedere da una fase all'altra. Ogni modifica all'interno di una fase può causare ritardi e slittamenti nel programma, influenzando inevitabilmente la data di consegna, attorno alla quale gravitano tutte le fasi del processo.

4.3 Vantaggi e svantaggi della metodologia Waterfall

Negli ultimi anni, il metodo Waterfall ha registrato una graduale diminuzione a favore di approcci più agili; tuttavia, è importante riconoscere che questo metodo presenta comunque una serie di vantaggi. In particolare, progetti e organizzazioni di dimensioni considerevoli, che richiedono una rigorosa suddivisione in fasi e rispettive scadenze, potrebbero trarre notevoli benefici dall'adozione di tale approccio.

Di seguito, i principali vantaggi di tale modello:

- ✓ **Adattabilità alle squadre in movimento**: l'utilizzo di questo metodo consente al progetto di mantenere una progettazione più dettagliata e solida grazie alle fasi iniziali di pianificazione. Questo aspetto è particolarmente adatto per team di grandi dimensioni, dove i membri possono variare nel corso del ciclo di vita del progetto
- ✓ **Organizzazione strutturata delle forze**: sebbene alcuni possano percepirlo come un vincolo, il modello a cascata impone al progetto e all'organizzazione di essere estremamente disciplinati nella struttura. La gestione dettagliata è inevitabile per affrontare aspetti critici dei progetti significativi
- ✓ **Possibilità di apportare modifiche di progettazione anticipate**: apportare modifiche in fase avanzate può risultare complesso, ma il modello Waterfall agevola le alterazioni nelle prime fasi del ciclo di vita. Ciò è particolarmente vantaggioso durante la compilazione delle specifiche, consentendo modifiche immediate e con sforzo minimo prima dell'inizio della codifica o implementazione
- ✓ **Adeguato allo sviluppo orientato alle pietre miliari**: grazie alla sua struttura lineare intrinseca, questo metodo si adatta bene a organizzazioni o team che operano in un contesto incentrato su pietre miliari e date fisse. Con frasi chiare e comprensibili, è possibile definire una timeline per l'intero processo, assegnando pietre miliari per ciascuna fase. Pur mantenendo la necessità di rispettare le scadenze, il metodo Waterfall è particolarmente adatto a progetti con tempi definiti.

Tale metodologia, di contro, presenta anche una serie di svantaggi, analizzati di seguito:

1. **Vincoli del design non adattivo**: Il difetto più evidente del modello a cascata è la sua mancanza intrinseca di adattabilità lungo tutte le fasi del ciclo di vita del progetto. Se un test nella fase 5 rileva un difetto fondamentale nella progettazione del sistema, si rende necessario non solo un significativo passo indietro, ma in alcuni casi può addirittura mettere in discussione la legittimità e il funzionamento dell'intero sistema. Anche se esperti team di sviluppo argomentano che tali situazioni dovrebbero essere prevenute con una progettazione accurata fin dall'inizio, non è sempre possibile considerare tutti gli scenari e ogni progetto comporta inevitabilmente il suo carico di rischi

2. **Ignora il feedback dei clienti a metà processo**: A causa del rigoroso processo previsto da questo metodo, il feedback da parte degli utenti o clienti è fornito tardi nel ciclo di sviluppo, risultando talvolta insufficiente o tardivo. Essendo un processo interno, la metodologia Waterfall esclude l'utente finale o il cliente coinvolto in un progetto. I clienti spesso preferiscono essere partecipi dell'evoluzione di un progetto, aggiungendo opinioni e chiarendo ciò che vogliono
3. **Periodo di test ritardato**: Mentre i modelli più moderni cercano di integrare i test durante lo sviluppo, il modello a cascata posticipa la fase di test nel processo. Ciò significa non solo che la maggior parte dei bug o di problemi di progettazione non verranno scoperti fino alla fase finale, ma incoraggia anche pratiche di codifica carenti, poiché i test sono considerati solo come un'aggiunta successiva
4. **Difficoltà nell'apportare cambiamenti**: La metodologia, nella sua forma più convenzionale, non consente l'implementazione di modifiche o aggiornamenti improvvisi o dell'ultimo minuto. Un cambiamento repentino nelle direttive del progetto potrebbe annullare una considerevole parte del progresso raggiunto fino a quel momento, con conseguenti ripercussioni sul rispetto delle scadenze.

4.4 La metodologia Agile

Nel corso degli anni, sono emersi diversi approcci nel project management, tra cui nel 2001 ha preso forma il Movimento Agile, nato in risposta alle richieste dei clienti come alternativa al tradizionale metodo Waterfall. L'approccio Agile si concentra sulla gestione flessibile e incrementale dei progetti, con un processo iterativo supportato da feedback empirici. Utilizzando team auto-organizzati e cross funzionali, l'approccio Agile favorisce la collaborazione per ottenere risultati in modo adattivo. Diverse metodologie, come SCRUM, XP e Kanban, sono emerse nell'ambito dell'approccio Agile. Questa evoluzione ha progressivamente ridotto l'adozione dell'approccio tradizionale in vari settori, alcuni in modo significativo e altri in modo più graduale. Nel contesto attuale, caratterizzato da cambiamenti rapidi, nuove tecnologie, cicli di time-to-market brevi e condizioni di mercato

mutevoli, oltre ad altri fattori sociali ed economici, è essenziale selezionare e utilizzare il giusto approccio di project management per garantire il successo nell'implementazione e nel controllo dei progetti.

4.4.1 I valori del manifesto Agile

L'enorme ritardo tra i requisiti aziendali (le applicazioni e le funzionalità richieste dai clienti) e la fornitura di tecnologia che rispondeva a tali esigenze ha portato all'annullamento di molti progetti. Il business, i requisiti e i clienti sono cambiati durante questo periodo di ritardo e il prodotto finale non soddisfaceva più le esigenze attuali. I modelli di sviluppo software di allora, guidati dal modello a cascata, non soddisfacevano la domanda di velocità e non sfruttavano la rapidità con cui il software poteva essere modificato.

Nel 2000, un gruppo di diciassette "leader del pensiero" si sono incontrati in una riunione in cui è stato scritto formalmente il Manifesto Agile.

"Stiamo scoprendo modi migliori per sviluppare software facendolo e aiutando gli altri a farlo. Attraverso questo lavoro siamo arrivati a considerare importanti:

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

Il software funzionante *più che la documentazione esaustiva*

La collaborazione col cliente *più che la negoziazione dei contratti*

Rispondere al cambiamento *più che seguire un piano*

Ovvero, fermo restando il valore delle voci a destra, consideriamo più importanti le voci a sinistra" (Kent Beck et al., 2001)

Il Manifesto Agile è composto da quattro valori fondamentali e che guidano l'approccio Agile allo sviluppo software. Ogni metodologia Agile applica i quattro valori in modo diverso, ma tutte si affidano a loro per guidare lo sviluppo e la fornitura di software funzionanti di alta qualità.

Di seguito i 4 valori sopracitati:

1. **Priorità delle persone e delle interazioni rispetto a processi e strumenti.** Il primo valore del Manifesto Agile enfatizza l'importanza delle persone e delle loro interazioni rispetto ai processi e agli strumenti. Questo concetto sottolinea che sono le persone che rispondono alle esigenze aziendali e guidano lo sviluppo, favorendo la flessibilità e la reattività al cambiamento. La comunicazione, ad esempio, risulta più fluida quando guidata da individui anziché da processi programmati
2. **Software funzionante oltre alla documentazione completa.** In passato si dedicava notevole tempo alla documentazione dettagliata nel processo di sviluppo e consegna del software. Agile non elimina la documentazione, ma la semplifica, valorizzando maggiormente il software funzionante. I requisiti sono espressi attraverso storie degli utenti, fornendo agli sviluppatori tutte le informazioni necessarie senza eccessive complicazioni documentali
3. **Collaborazione continua con i clienti oltre alla negoziazione del contratto.** Mentre la negoziazione si focalizza sulla definizione dettagliata delle consegne, la collaborazione è un concetto più ampio nel Manifesto Agile. Contrariamente ai modelli a cascata, che coinvolgono il cliente solo prima e dopo il processo di sviluppo, Agile promuove il coinvolgimento continuo del cliente. Questo coinvolgimento può includere demo periodiche o addirittura la partecipazione quotidiana di un utente finale alle riunioni del team per garantire che il prodotto soddisfi costantemente le esigenze aziendali
4. **Adattarsi al cambiamento piuttosto che seguire un piano.** Mentre lo sviluppo tradizionale considerava il cambiamento come un inconveniente, Agile abbraccia la flessibilità. Con iterazioni brevi, le priorità possono essere regolate di iterazione in iterazione, consentendo l'aggiunta di nuove funzionalità in modo rapido. Agile vede il cambiamento come un miglioramento costante del progetto, offrendo valore aggiuntivo attraverso l'adattamento continuo.

4.4.2 I principi del manifesto Agile

I dodici princìpi sono i princìpi guida per le metodologie incluse sotto il titolo "The Agile Movement." Descrivono una cultura in cui il cambiamento è benvenuto e il cliente è il fulcro del lavoro. Dimostrano anche che l'intento del movimento "è quello di portare lo sviluppo in linea con le esigenze aziendali" (Alistair Cokburn, cofirmatario del Manfesto Agile).

I dodici principi dello sviluppo Agile includono:

1. **Soddisfazione del cliente tramite consegne tempestive e continue del software** La contentezza del cliente è massimizzata attraverso la regolare consegna di software funzionante, evitando lunghe attese tra le diverse release
2. **Adattamento continuo dei requisiti durante lo sviluppo.** La flessibilità nel modificare requisiti o richieste di funzionalità previene ritardi nel processo di sviluppo
3. **Consegne frequenti di software funzionanti.** Scrum abbraccia questo principio, operando in sprint o iterazioni per garantire la consegna regolare di software funzionante
4. **Collaborazione tra stakeholder aziendali e sviluppatori per l'intero progetto.** Decisioni ottimali emergono dall'allineamento tra l'azienda e il team tecnico durante tutto il progetto
5. **Supporto, fiducia e motivazione delle persone coinvolte.** I team motivati mostrano maggiore efficacia rispetto a quelli insoddisfatti, evidenziando l'importanza di supporto e fiducia
6. **Interazioni faccia a faccia.** La comunicazione raggiunge il massimo successo quando i team di sviluppo sono co-localizzati e interagiscono direttamente
7. **Il software funzionante come principale misura di progresso.** Il progresso è definito principalmente dalla consegna di software funzionante al cliente, riflettendo il risultato finale del processo
8. **Processi agili per sostenere un ritmo di sviluppo coerente.** L'adozione di processi agili consente ai team di stabilire e mantenere un ritmo ripetibile nella consegna di software funzionante

9. **Attenzione ai dettagli tecnici e al design per migliorare l'agilità.** Competenze tecniche adeguate e un design robusto sono fondamentali per sostenere il ritmo, migliorare costantemente il prodotto e adattarsi al cambiamento
10. **Semplicità nello sviluppo necessario per il completamento del lavoro.** L'approccio è orientato a sviluppare solo ciò che è essenziale per portare a termine il lavoro in corso
11. **Team auto-organizzati per favorire grandi architetture, requisiti e design.** Membri del team con competenze e motivazioni intrinseche assumono responsabilità decisionali, promuovendo la creazione di prodotti di qualità attraverso comunicazione e condivisione di idee
12. **Riflessioni periodiche sull'aumentare l'efficacia.** Il percorso verso l'auto-miglioramento coinvolge l'ottimizzazione dei processi, lo sviluppo delle competenze e l'adozione di tecniche che rendono il lavoro del team più efficiente.

L'obiettivo di Agile è allineare lo sviluppo alle esigenze aziendali, e il successo di questo approccio è manifesto. I progetti Agile mettono al centro il cliente, promuovendo la guida e la partecipazione diretta da parte dei clienti stessi. Di conseguenza, Agile ha assunto una posizione predominante nello sviluppo software, diventando una prospettiva consolidata nell'intero settore e, in effetti, costituendo un settore autonomo.

4.5 Le fasi del modello Agile

L'attenzione rivolta alle necessità, alle iterazioni, alle relazioni e alla collaborazione, ha naturalmente portato alla necessità di rivedere il processo definito nel metodo Waterfall. Ciò ha comportato il passaggio da un processo rigido e lineare a un approccio più agile, snello e adattabile basato su fasi di sviluppo iterative, conosciute come Sprint. Ma come si svolgono queste fasi? Ogni Sprint permette il rilascio di una parte dell'applicativo contenente funzionalità specifiche precedentemente concordate. Il cliente ha quindi l'opportunità di

utilizzare e testare il software, fornendo al team di sviluppo preziosi feedback. Questi feedback sono essenziali per orientare l'implementazione nelle iterazioni successive, avvicinando progressivamente il prodotto finale a un livello qualitativo superiore e più aderente alle sue esigenze.

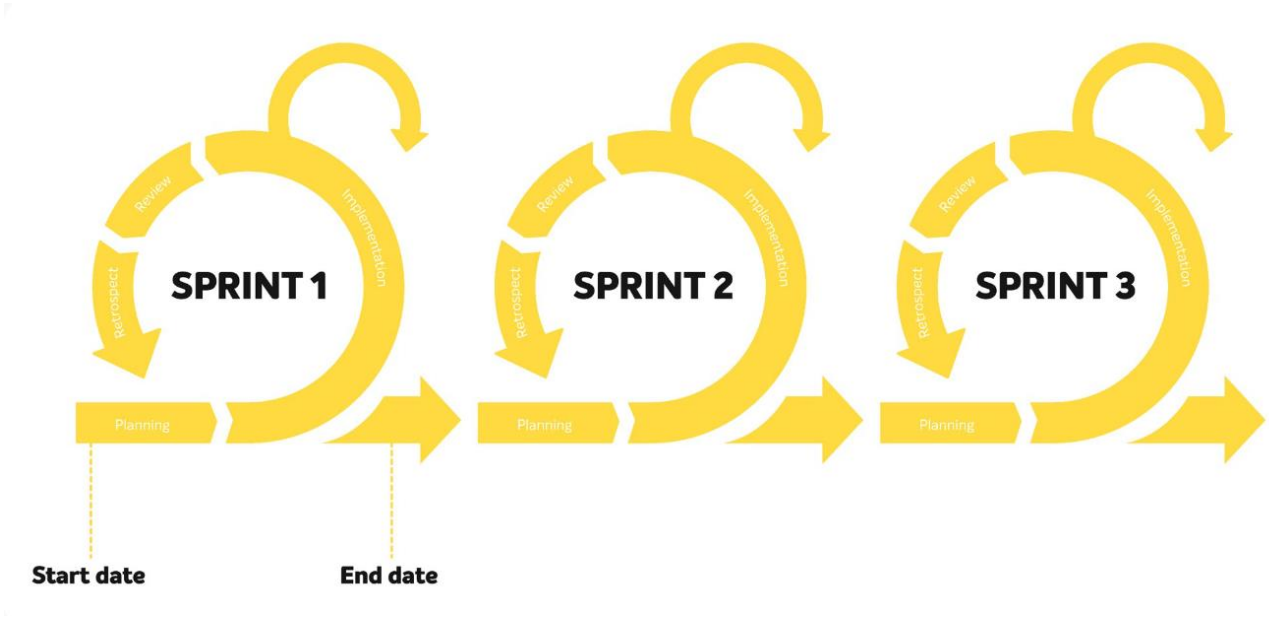


Figura 7 - Gli "sprint" del modello Agile [5]

Dunque, prima, dopo e durante gli sprint, ci sono degli eventi, quattro tipi differenti che hanno ognuno una funzione diversa:

- Sprint Planning
- Daily Stand-up
- Sprint Review
- Retrospective Meeting

Analizziamo ogni sprint più nel dettaglio.

4.5.1 Sprint planning

All'inizio di ciascuno sprint, il team si riunisce in un processo collaborativo per determinare il lavoro da svolgere durante lo sprint stesso. Questa sessione, denominata "sprint planning", ha una durata massima di 8 ore per uno sprint di quattro settimane. Durante lo sprint planning, il team svolge le seguenti attività:

1. Definisce lo sprint goal, cioè il risultato che può essere conseguito durante lo sprint
2. Suddivide gli elementi presenti nel product backlog, necessari per raggiungere lo sprint goal
3. Definisce il processo per raggiungere il risultato identificando le azioni da intraprendere per elaborare gli elementi e la quantità di lavoro associata
4. Formula lo sprint backlog.

Per "Sprint Backlog" si intende un elenco prioritario di funzionalità che un prodotto dovrebbe contenere. A volte viene indicato come un elenco di cose da fare ed è considerato un "artefatto" all'interno del framework di sviluppo del software Scrum.

Al termine della riunione, il team assicura che tutti comprendano e assimilino lo Sprint Goal e lo Sprint Backlog. Durante questa fase conclusiva, il team pone domande, richiede chiarimenti e approfondimenti per ottenere informazioni dettagliate sul lavoro da affrontare. È importante sottolineare che, al termine dello Sprint Planning, è il team, non il product owner, a determinare la quantità di lavoro che può essere gestita nel successivo sprint.

4.5.2 Daily stand-up

Il Daily Stand-up si basa sulle seguenti caratteristiche:

- Dura sempre 15 minuti
- Avviene sempre nello stesso luogo
- Sono presenti sempre le stesse persone

- Avviene sempre alla stessa ora.

Durante questo evento, il team di sviluppo verifica e mette a punto la pianificazione della giornata attraverso 3 domande: “Che cosa ho fatto ieri”?, “Che cosa faccio oggi”?, “C’è qualche impedimento che mi ostacola nell’andare avanti? Queste tre domande fanno sì che ogni membro del team comprenda dove si è arrivati con il lavoro svolto e cosa manca ancora da completare. Inoltre, dichiarare cosa verrà fatto durante la giornata aumenta il senso di responsabilità del team. È importante capire che il Daily Stand-up non deve concentrarsi solamente sulla risoluzione dei problemi, ma sugli aggiornamenti da comunicare a tutte le figure dello Scrum team per fare in modo che siano connessi tra di loro.

4.5.3 Sprint review

L’ ultimo evento di Scrum è la Sprint Review, un incontro che si tiene al termine di ogni sprint e che coinvolge lo Scrum team, il product owner, lo Scrum master e gli stakeholders. Questa riunione informale permette al team Scrum di collaborare con gli stakeholders per esaminare l'incremento del Prodotto e raccogliere i loro feedback, in linea con lo sprint goal stabilito durante lo sprint planning. Sulla base delle informazioni acquisite durante lo sprint review, il product backlog viene adattato e vengono condivisi i piani per gli sprint successivi. Durante questa sessione, è più prioritario presentare l'obiettivo generale che si intendeva raggiungere, piuttosto che addentrarsi nei dettagli di ciascun elemento degli sprint.

4.5.4 Retrospective meeting

Lo Sprint Retrospective è un'occasione formale per lo Scrum team di ispezionarsi e creare un piano per miglioramenti da attuare durante il prossimo sprint. Lo Scrum team ispeziona diversi aspetti su come è andato lo sprint:

- Persone
- Relazioni
- Processi
- Strumenti
- Regole
- Definition of done.

Durante questo meeting vengono selezionati incrementi realizzabili da sperimentare nel prossimo sprint attraverso le seguenti domande: “che cosa è andato bene?”, “che cosa non ha funzionato?”, “quali impedimenti ostacolano il team?” Se la durata dello sprint fosse superiore ad un mese, il team rischierebbe di perdere di vista l’obiettivo, aumentando complessità e rischi nell’implementazione del progetto. Gli sprint di breve durata permettono a tutti i membri del team, inclusi il product owner e lo Scrum master, di mantenere un controllo diretto sui progressi, monitorare l’andamento e ispezionare le modifiche apportate al prodotto.

4.6 Vantaggi e svantaggi della metodologia Agile

Steve Denning l'ha espresso chiaramente in un articolo su Forbes datato gennaio 2018: “Agile is eating the world”. In altre parole, le imprese che abbracciano, anche se non necessariamente con questa denominazione, i principi dell'Agile sono quelle che prevalgono nella sfida competitiva del mercato e godono di crescente successo. È l'innovazione manageriale, piuttosto che prodotti o servizi specifici, a conferire grande successo a molte organizzazioni in diversi settori. Questa rivoluzione ha origini nel settore digitale, dove si trovano numerosi esempi concreti anziché casi isolati. Poiché la produzione di applicazioni rappresenta l'aspetto chiave di queste aziende, l'Agile si è affermato come il modo più naturale e vantaggioso di operare. Questo modello, dunque, presenta importanti vantaggi e svantaggi, che vediamo di seguito:

I vantaggi:

- ✓ L'attenzione è concentrata sul cliente
- ✓ L'approccio si dimostra flessibile nell'accogliere i cambiamenti
- ✓ Il processo di sviluppo del software si caratterizza per una maggiore rapidità nel ciclo di vita
- ✓ Vengono favorite comunicazioni efficienti
- ✓ Si rivela particolarmente adatto per progetti con finanziamenti non fissi
- ✓ La prevedibilità del programma è garantita mediante l'analisi di ciascuno sprint.

Gli svantaggi:

- ✓ Nel contesto Agile è previsto che ogni membro del team sia interamente dedicato al progetto in questione
- ✓ Comporta un aumento dei costi necessario per gestire modifiche di priorità e l'aggiunta di sprint supplementari
- ✓ L'approccio Agile richiede un elevato livello di partecipazione da parte del cliente, il quale potrebbe non essere disponibile ad impegnarsi completamente.

CAPITOLO 5 – DEVOPS

Il termine "DevOps" nasce dalla fusione di "Development" (sviluppo) e "Operations" (operazioni), ma va oltre la mera combinazione di questi concetti, incorporando una vasta gamma di idee e procedure. DevOps comprende aspetti come la sicurezza, la collaborazione, l'analisi dei dati e molti altri elementi.

DevOps descrive gli approcci finalizzati ad accelerare i processi che consentono a un'idea, come una nuova funzionalità software, una richiesta di miglioramento o una correzione di bug, di attraversare l'intero ciclo, dallo sviluppo al rilascio in un ambiente di produzione in cui può apportare valore all'utente. Tali approcci presuppongono una comunicazione frequente tra i team operativi e di sviluppo. All'interno di ciascun team, il lavoro è orientato all'empatia tra i colleghi, richiedendo scalabilità e modalità di provisioning flessibili. DevOps offre maggiore efficienza attraverso self-service e automazione, consentendo agli sviluppatori di collaborare strettamente con il personale IT operativo, accelerando le fasi di creazione, test e rilascio del software senza compromettere l'affidabilità.

Naturalmente, questo approccio comporta modifiche più frequenti al codice e un utilizzo più dinamico dell'infrastruttura. Tali esigenze, tuttavia, non possono essere soddisfatte con le tradizionali strategie di gestione e richiedono cambiamenti sostanziali per ottenere vantaggi concreti.

DevOps integra lo sviluppo (Dev) e le operazioni (Ops), unendo persone, processi e tecnologie nell'intero ciclo di vita dell'applicazione, compresa la sua pianificazione, sviluppo, distribuzione e gestione operativa. Tale approccio facilita il coordinamento e la collaborazione tra ruoli precedentemente separati, come lo sviluppo, le operazioni IT, la progettazione della qualità e la sicurezza.

I team che abbracciano la cultura, le procedure e gli strumenti di DevOps mirano a consolidare la fiducia nelle applicazioni sviluppate, rispondere in modo più efficace alle esigenze dei clienti e raggiungere più rapidamente gli obiettivi aziendali. L'adozione di DevOps consente ai team di generare valore in modo continuo per i clienti, garantendo la produzione di prodotti di qualità superiore e maggiore affidabilità.

Per realizzare pienamente DevOps, è essenziale abbracciare i principi culturali ad esso associati. Coltivare una mentalità DevOps richiede trasformazioni significative nel modo in cui le persone lavorano insieme. Quando le organizzazioni abbracciano la cultura DevOps, favoriscono un ambiente in cui i team possono prosperare con elevate performance. Anche se l'adozione di pratiche DevOps automatizza e ottimizza i processi tramite la tecnologia, i suoi benefici completi non possono essere realizzati senza una piena adesione alla cultura da parte dell'organizzazione e dei suoi membri.

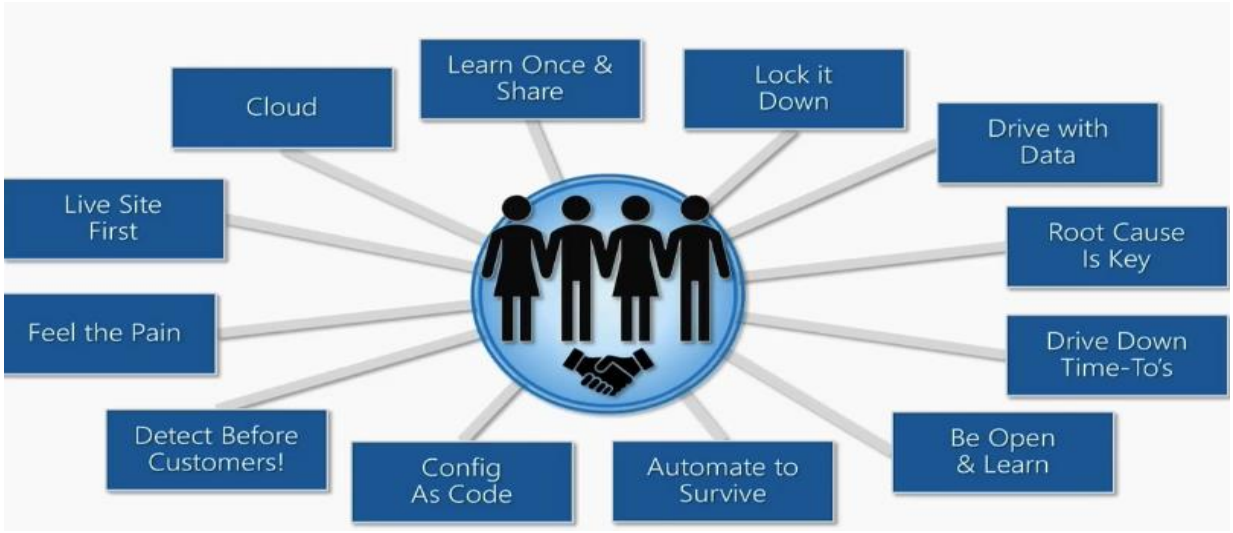


Figura 8 - Gli elementi fondamentali della cultura DevOps [6]

Le seguenti pratiche rappresentano elementi fondamentali della cultura DevOps:

1. **Collaborazione, visibilità e allineamento**: Un pilastro distintivo di una cultura DevOps sana è la collaborazione tra i team, la quale ha origine dalla visibilità condivisa. Sviluppo, IT e altri reparti devono condividere processi, priorità e questioni DevOps per lavorare in armonia. La pianificazione congiunta consente loro di allinearsi meglio sugli obiettivi e sulle metriche di successo aziendali.
2. **Cambiamenti nell'ambito e nella responsabilità**: A mano a mano che i team si allineano, acquisiscono la proprietà e partecipano ad altre fasi del ciclo di vita del

prodotto, non limitandosi alle sole attività centrali dei propri ruoli. Ad esempio, gli sviluppatori assumono la responsabilità non solo dell'innovazione e della qualità nello sviluppo, ma anche delle prestazioni e della stabilità operative delle loro modifiche. Allo stesso tempo, gli operatori IT devono includere aspetti di governance, sicurezza e conformità fin dalle fasi di pianificazione e sviluppo.

3. **Cicli di rilascio più brevi**: I team DevOps mantengono l'agilità rilasciando il software in cicli più rapidi. Questa pratica semplifica la pianificazione e la gestione dei rischi poiché il progresso è graduale, riducendo l'impatto sulla stabilità del sistema. Ridurre i tempi di rilascio consente inoltre alle organizzazioni di adattarsi più prontamente alle esigenze dei clienti e alle pressioni della concorrenza.
4. **Apprendimento continuo**: I team DevOps ad alte prestazioni abbracciano una mentalità di crescita costante. Non temono il fallimento, ma lo considerano parte integrante del processo di apprendimento. Si impegnano costantemente nel miglioramento, mirando ad aumentare la soddisfazione dei clienti e ad accelerare l'innovazione e l'adattabilità al mercato.

5.1 Cultura

La cultura è una base fondamentale per il DevOps poiché richiede una mentalità di crescita e apprendimento continuo per avere successo. Il supporto della leadership è uno degli elementi critici per il suo successo. Prima di discutere di come appare la cultura DevOps, consideriamo il ruolo della cultura nella capacità di un'organizzazione di adottare il DevOps. Secondo George Spafford, analista della azienda di ricerca e consulenza Gartner:

"La resistenza culturale e i bassi livelli di disciplina nei processi creeranno tassi significativi di fallimento per le iniziative DevOps" (Spafford George et al.,"The Phoenix Project", 2013)

Gene Kim, autore di The Phoenix Project e DevOps Handbook, invece, afferma:

"Il DevOps è un viaggio pieno di sfide, e raramente queste sfide sono semplicemente dovute alla tecnologia sbagliata o ai processi sbagliati. In realtà, gli ostacoli più grandi e più difficili tendono ad essere culturali. E se sbagli la cultura, anche se fai tutto il resto correttamente, ti stai dirigendo verso la frustrazione, costi aggiuntivi e probabilmente fallimento" (Kim Gene et al.,"The Phoenix Project", 2013)

La cultura è l'eredità sociale di un gruppo. È un modello di risposte scoperte, sviluppate o inventate durante la storia del gruppo nel gestire i problemi che sorgono dalle interazioni tra i suoi membri e tra loro e il loro ambienti. Essa determina:

- Cosa è accettabile o inaccettabile
- Cosa è importante o non importante
- Cosa è giusto o sbagliato
- Cosa è fattibile o non fattibile
- Chi assumi, licenzi e promuovi.

La ricerca di Gartner mostra che entro il 2023, il 90% delle iniziative DevOps fallirà a causa delle limitazioni degli approcci gestionali utilizzati dalla leadership.

La responsabilità principale della leadership è la creazione di un ambiente che favorisca una cultura DevOps di successo.

5.2 Mentalità

In seguito a recenti studi, sono stati stilati gli undici esempi di mentalità da avere per applicare coerentemente la metodologia DevOps.

1. **La mentalità di leadership**. Gartner fa le seguenti raccomandazioni:

    - Identificare leader trasformazionali dando priorità a specifiche caratteristiche comportamentali necessarie per guidare un'iniziativa DevOps, mettendo meno enfasi sulle competenze tecniche

- Sviluppare leader trasformazionali abbracciando il fallimento come opportunità di apprendimento
- Gestire leader trasformazionali dando loro il potere di prendere decisioni senza esitazioni e fornendo obiettivi chiari e metriche chiave.

Poiché il DevOps è trasformativo, i leader delle infrastrutture e delle operazioni (I&O) devono identificare candidati visionari, adattabili, motivanti, capaci di dare potere e responsabili.

2. **Mentalità orientata al cliente.** Essere orientati al cliente significa:

    - Ascoltare e comunicare con i nostri clienti
    - Misurare ciò che è importante
    - Accogliere il feedback negativo durante la produzione
    - Costruire, misurare e imparare
    - Utilizzare il feature toggling per una distribuzione agile. Esso è un interruttore di funzionalità nello sviluppo del software e fornisce un'alternativa al mantenimento di più rami di funzionalità nel codice sorgente.
    - Raccogliere dati in modo ampio, ma attento

3. **Mentalità lean-thinking.** La mentalità lean-thinking inizia con una comprensione dettagliata del valore che il cliente attribuisce ai prodotti e ai servizi. L'organizzazione si concentra sull'eliminare gli sprechi in modo da poter fornire il valore che il cliente si aspetta al più alto livello di redditività. Il flusso di valore comprende l'intero ciclo di vita del prodotto, dalle materie prime all'uso da parte del cliente e alla disposizione finale del prodotto. Per eliminare gli sprechi, l'obiettivo finale del Lean è puntare ad una comprensione accurata e completa del flusso di valore.

    Flusso: Comprendere il flusso è essenziale per eliminare gli sprechi. Se il flusso di valore si interrompe in qualsiasi punto, lo spreco è l'inevitabile risultato. Il principio di produzione lean del flusso consiste nel creare una catena del valore senza interruzioni nel processo produttivo e in cui ogni attività è in sintonia con le altre.

Pull: Il principio lean del pull aiuta a garantire il flusso, facendo in modo che nulla venga prodotto in anticipo, il che comporterebbe la creazione di un inventario di lavori in corso e interromperebbe il flusso sincronizzato. Piuttosto che utilizzare l'approccio tradizionale americano alla produzione, basato su previsioni e programmi, l'approccio pull impone che nulla venga prodotto fino a quando il cliente non lo ordina.

Perfezione: I praticanti del lean si sforzano di raggiungere la perfezione. La marcia verso un processo perfetto avviene attraverso miglioramenti continui che affrontano le cause alla radice dei problemi di qualità e degli sprechi di produzione. La ricerca incessante della perfezione è ciò che spinge gli utenti dell'approccio a scavare più a fondo, misurare di più e cambiare più spesso rispetto ai loro concorrenti

4. **Mentalità del pensiero sistemico.** Una mentalità del pensiero sistemico enfatizza le prestazioni dell'intero sistema, non le prestazioni di un singolo sito di lavoro o dipartimento. Ci si concentra su tutti i flussi di valore aziendale abilitati dall'IT. In altre parole, inizia quando i requisiti sono identificati dall'azienda o dall'IT, sono sviluppati nello sviluppo e poi trasferiti nelle operazioni IT, dove il valore viene poi consegnato al cliente come servizio
5. **Mentalità di rimozione degli sprechi.** Una mentalità lean si concentra sull'identificare e rimuovere i sette sprechi mortali che non sono di valore per il cliente:

    - Lavoro parzialmente completato
    - Processi aggiuntivi
    - Funzionalità aggiuntive
    - Cambiamento di attività
    - Attesa
    - Movimento
    - Difetti

6. **Il pensiero della teoria dei vincoli.** La teoria dei vincoli è una metodologia per identificare e rimuovere i vincoli (anche chiamati "colli di bottiglia") che limitano la capacità di produzione. Nella pratica, si inizia identificando il fattore più importante che ostacola il raggiungimento di un obiettivo. Si lavora per ridurre al minimo quel fattore fino a quando non costituisce più un limite

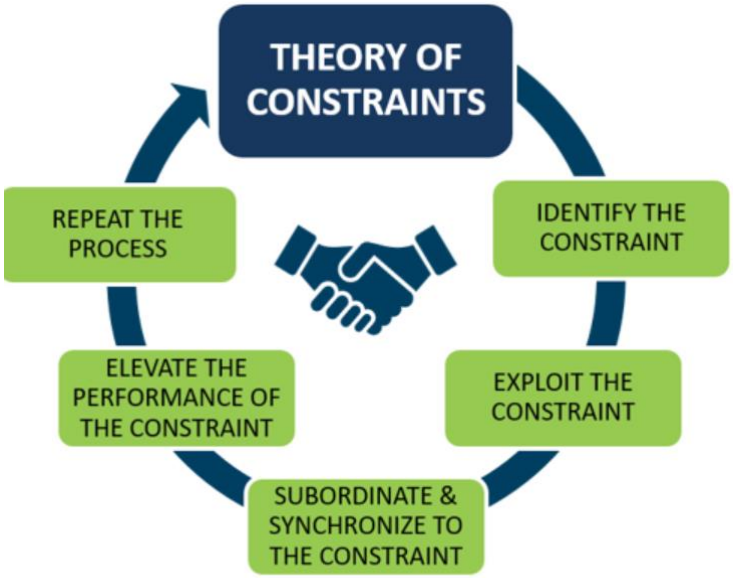


Figura 9 - La teoria dei vincoli [7]

7. **Mentalità di bilanciamento tra allineamento e autonomia.** È necessario raggiungere un equilibrio tra allineamento e autonomia. Troppo allineamento porta a meno innovazione, meno motivazione e meno collaborazione. Troppa autonomia porta a più caos, confusione e conflitto, riducendo anche la coerenza

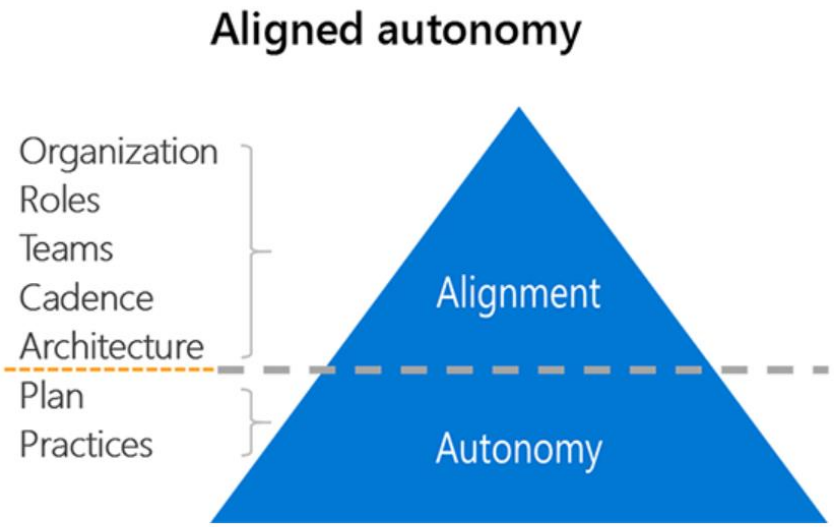


Figura 10 - Il bilanciamento tra allineamento e autonomia [8]

8. **Mentalità dello "shift left" nel testing.** Lo "spostamento a sinistra" nel testing è un approccio utilizzato per accelerare il testing del software e facilitare lo sviluppo spostando il processo di testing in un punto precedente del ciclo di sviluppo. Lo "spostamento a sinistra" è un riferimento allo spostamento del testing a sinistra su una linea temporale. Questo aiuta a costruire qualità e identificare problemi in modo più tempestivo per ridurre lo spreco di ripetizioni. Lo "shift left" nel testing è progettato per essere un modello migliore per lo sviluppo in corsia veloce perché i modelli di testing tradizionali che aspettano fino a più tardi nel ciclo di sviluppo, possono causare dei colli di bottiglia nello sviluppo
9. **Mentalità di sicurezza.** Per adottare una mentalità di sicurezza, le squadre devono:

    - Promuovere la consapevolezza
    - Definire i loro principi
    - Vivere secondo i loro principi

10. **Mentalità dello sviluppo guidato da ipotesi.** Utilizzare un approccio Lean Product per sviluppare in cicli più brevi e utilizzare lo sviluppo guidato da ipotesi, aiuta a creare piccoli esperimenti per ottenere feedback dai nostri clienti e decisioni basate sui dati.

    L'approccio dello sviluppo guidato da ipotesi:

    - Inizia da un'ipotesi (qualcosa accettato come vero senza prova)
    - Articola l'ipotesi da testare
    - Esegue sperimentazioni e test
    - Esamina le prove

11. **Mentalità del "live-site".** Per un team DevOps, non c'è posto migliore della produzione. Tutto ciò che si esegue è finalizzato a migliorare l'esperienza dei clienti. Per creare un sito stabile e ad alte prestazioni, è necessario applicare le migliori pratiche di operazioni continue in modo disciplinato e continuo per mantenere il sito in salute. I fattori chiave della nostra cultura "live-site" includono:

    - Rilevare prima che i clienti sentano il dolore
    - Guidare con i dati
    - Configurare come codice
    - Automatizzare per sopravvivere
    - Essere aperti e imparare.

5.3 Pilastri e funzioni

Dopo un considerevole dibattito, in seguito a collaborazione e affinamento, sono stati identificati i seguenti quattro pilastri (pillars), derivati da Persone/Processo/Prodotto:

1. Cultura
2. Lean product (prodotto snello)
3. Architettura
4. Tecnologia.

Inoltre, sono state anche identificate otto funzioni, o capacità, che includono varie pratiche all'interno di ciascuna funzione o tra funzioni. Queste sono:

- Continuous Planning
- Continuous Integration
- Continuous Delivery
- Continuous Operations
- Continuous Quality
- Continuous Security
- Continuous Collaboration
- Continuous Improvement.

Di queste funzioni, si focalizzerà l'attenzione, nei paragrafi successivi e nel quinto capitolo (che tratterà un caso applicativo), su tre di esse: Continuous Integration, Continuous Delivery e Continuous Operations, con l'aggiunta del Source Code Management, il cui ruolo verrà analizzato in seguito. Queste quattro funzioni corrispondono ad altrettante fasi del ciclo di vita di un software.

5.3.1 Source Code Management

Il controllo delle versioni, anche noto come controllo del codice sorgente, consiste nell'attività di monitorare e gestire le modifiche apportate al codice del software nel corso del tempo. I sistemi di controllo delle versioni sono strumenti software che agevolano i team di sviluppo nel gestire le modifiche al codice sorgente. In un contesto di sviluppo accelerato, tali sistemi consentono di operare con maggiore rapidità ed efficienza. Essi risultano particolarmente utili per i team DevOps, contribuendo a ridurre i tempi di sviluppo e ad aumentare il successo delle distribuzioni.

Questi strumenti mantengono un registro dettagliato di ogni modifica apportata al codice all'interno di un database apposito. In caso di errori, gli sviluppatori hanno la possibilità di retrocedere nel tempo e confrontare le versioni precedenti del codice per individuare e correggere l'errore, minimizzando così le interruzioni per tutti i membri del team.

Gli sviluppatori di software operanti in team sono costantemente impegnati nella scrittura di nuovo codice sorgente e nella modifica di quello già esistente. Il codice relativo a progetti, app o componenti software è solitamente organizzato in una struttura a cartelle o ad "albero di file". In questo contesto, un membro del team potrebbe concentrarsi sulla creazione di una nuova funzione, mentre un altro si occupa di risolvere un bug, entrambi modificando parti diverse del codice.

Il controllo delle versioni rappresenta una soluzione a questo scenario, permettendo ai team di tenere traccia di ogni modifica apportata da ciascun membro e di evitare conflitti derivanti dal lavoro simultaneo di più sviluppatori. È possibile che le modifiche apportate da un membro siano incompatibili con quelle fatte contemporaneamente da un altro. È essenziale individuare e risolvere tali problemi in modo ordinato, senza interrompere il lavoro degli altri membri del team. Inoltre, nel corso del processo di sviluppo, ogni modifica potrebbe introdurre nuovi bug, pertanto, il software non può essere considerato affidabile fino a quando non viene testato. Di conseguenza, lo sviluppo e i test procedono di concerto fino alla preparazione di una nuova versione.

Il Source Code Management offre numerosi vantaggi, tra cui:

1. È necessaria una registrazione completa e dettagliata di tutte le modifiche apportate a ciascun file nel corso del tempo, comprese creazioni, eliminazioni e modifiche del contenuto, effettuate da diversi collaboratori nel corso degli anni. Ogni strumento di controllo versione adotta un proprio metodo per gestire rinomine e spostamenti dei file. Questa cronologia deve includere informazioni sull'autore, la data e annotazioni riguardanti lo scopo di ogni modifica. Possedere una cronologia completa facilita il ritorno a versioni precedenti per analizzare le cause principali dei difetti e risolvere eventuali problemi delle versioni precedenti del software. Nel contesto di un software attivamente sviluppato, quasi tutte le versioni possono essere considerate “precedenti”

2. La possibilità di creare e unire rami di sviluppo è essenziale per consentire ai membri del team di lavorare simultaneamente su flussi di lavoro separati. Questo strumento non solo agevola il lavoro collaborativo, ma può anche essere utile per gli sviluppatori che lavorano in modo autonomo, consentendo loro di gestire modifiche indipendenti. Creare un "branch" in un VCS consente di mantenere più flussi di lavoro separati e poi unirli tramite "merge" per combinare le modifiche in modo coerente. Molti team seguono la pratica di creare rami per singole funzionalità o rilasci, o entrambi. La scelta del flusso di lavoro dipende dalle esigenze specifiche del team

3. È importante poter tracciare ogni modifica del software e collegarla ai sistemi di gestione progetti e monitoraggio bug. Annotare ogni modifica con un messaggio chiaro sul suo scopo e intento è cruciale non solo per l'analisi delle cause principali e altre indagini, ma anche per ottimizzare altri processi. Disporre di una cronologia annotata del codice durante la lettura facilita la comprensione del suo comportamento e della sua progettazione, consentendo agli sviluppatori di apportare modifiche coerenti e in linea con la visione a lungo termine del sistema.

Questo è particolarmente rilevante quando si lavora con codice esistente e aiuta gli sviluppatori a stimare con precisione il lavoro futuro.

5.3.2 Continuous Integration

Nell'ambito dell'ingegneria del software, l'integrazione continua, o Continuous Integration (CI) in inglese, si riferisce alla pratica di allineare frequentemente le diverse componenti di un progetto all'interno di un ambiente di lavoro condiviso. Ciò implica che gli sviluppatori caricano il proprio codice prodotto nel Sistema di Controllo delle Versioni (SCM) più volte al giorno. Utilizzando appositi software, viene eseguita automaticamente la compilazione del codice e il testing delle unità, l'integrazione e l'analisi statica del codice. La Continuous Integration è stata concepita principalmente per affrontare il cosiddetto "Integration Hell", una situazione in cui gli sviluppatori eseguono la compilazione del prodotto troppo raramente, accumulando piccoli problemi che rendono, infine, il processo di compilazione estremamente lungo e difficile da correggere. La presenza dei test è cruciale, pertanto la Continuous Integration (CI) viene spesso associata alla pratica del Test-Driven Development (TDD). I test di unità e di integrazione hanno obiettivi distinti: i test di unità analizzano singole funzionalità del software in modo isolato, garantendo esecuzioni rapide, mentre i test di integrazione verificano l'interazione corretta tra le diverse "parti" sviluppate dagli sviluppatori. Sebbene i test di integrazione siano generalmente più lenti dei test di unità, è essenziale che non richiedano troppo tempo per essere completati, altrimenti potrebbero scoraggiare gli sviluppatori dall'eseguirli. I test di integrazione sono particolarmente adatti per progetti con architetture a microservizi, in cui numerosi servizi devono essere integrati e possono essere sviluppati in parallelo. Il vantaggio della CI risiede nell'avere costantemente un prodotto aggiornato, funzionante e testato. Al fine di mantenere l'integrità del codice disponibile a tutto il team, non è accettabile lasciare il lavoro con una build fallita. In caso di build fallita, ci sono due opzioni:

- Cercare di risolvere il problema nel minore tempo possibile
- Tornare all’ultima versione funzionante del software.

Nella pratica, l'integrazione continua segue i seguenti passaggi:

1. Sviluppo del codice in ambiente locale
2. Esecuzione della build in locale; in caso di errore, ritornare al punto precedente
3. Verifica tramite lo strumento remoto di build del codice nel SCM se il processo di build è in corso. In caso affermativo, attendere il completamento
4. Effettuare un push del codice all'interno del SCM. Automaticamente, il processo di build remoto dovrebbe avviarsi. In caso di fallimento, ritornare al primo passaggio.

Per ottimizzare ulteriormente l'esecuzione dei test, è consigliabile creare script di configurazione che replicano con precisione gli ambienti di utilizzo finale del software. Questo approccio riduce la necessità di risolvere problemi di compatibilità al momento della consegna del prodotto.

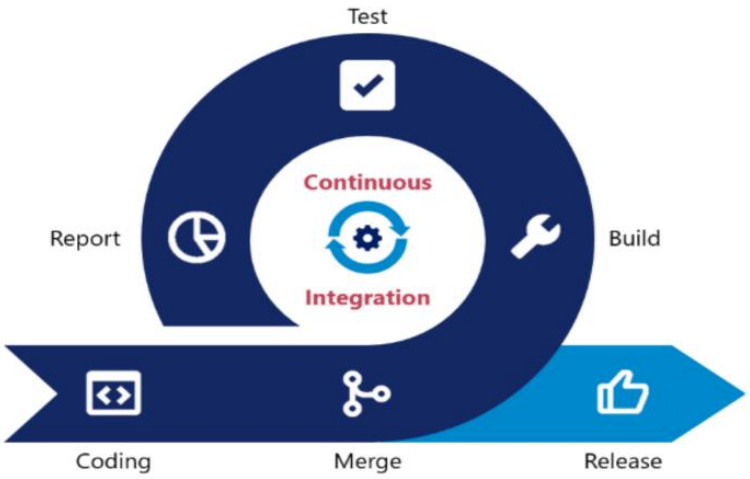


Figura 11 - Continuous Integration [9]

5.3.3 Continuous Delivery

Continuous Delivery (CD), nota anche come Distribuzione Continua, è un'approccio metodologico per lo sviluppo e la distribuzione del software che mira a fornire codice in modo rapido, affidabile e continuo. L'obiettivo principale di CD è consegnare software di alta qualità in tempi brevi, riducendo così i costi complessivi. Questo metodo si concentra

sulla consegna frequente di piccole modifiche, le quali vengono testate e validate. Si cerca di mantenere una coerenza elevata tra l'ambiente di test e quello di produzione, utilizzando processi automatizzati di test e distribuzione per accelerare le varie fasi. Spesso, la CD viene combinata con la metodologia di Continuous Integration, formando l'approccio CI/CD che viene applicato lungo l'intero ciclo di vita del software, dalla scrittura del codice iniziale fino alla consegna del prodotto completo all'utente finale.

In sintesi, è un approccio che permette ai team di rilasciare prodotti di qualità in modo frequente e prevedibile, automatizzando il processo che va dal repository del codice sorgente all'ambiente di produzione. Alcune organizzazioni ancora effettuano il rilascio manuale dei prodotti, trasferendoli da un team all'altro, come mostrato nel diagramma seguente. In questa configurazione, gli sviluppatori e il personale operativo si trovano spesso agli estremi opposti della catena di operazioni. Questo setup causa ritardi nelle consegne, generando frustrazione nei team e insoddisfazione nei clienti. Il prodotto viene pubblicato solo dopo un lungo processo, ritardando il conseguimento di profitti.

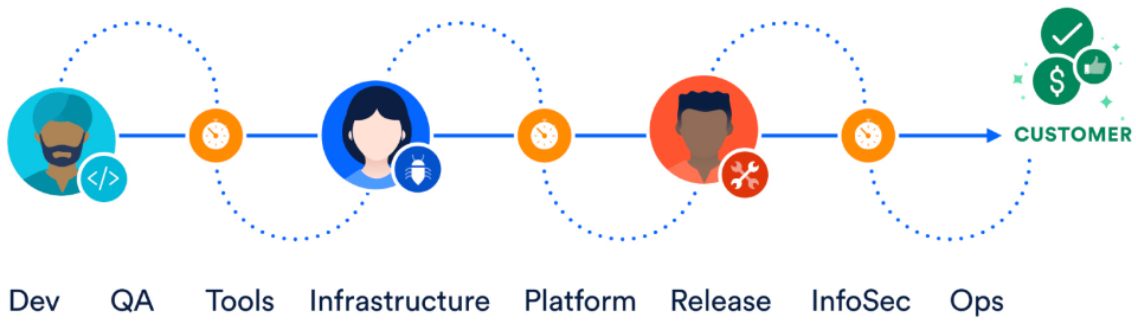


Figura 12 - Continuous Delivery (Fonte: documento Accenture)

La pipeline di Continuous Delivery può includere una fase di controllo manuale immediatamente prima della produzione. Tali controlli manuali richiedono l'intervento umano e possono essere necessari all'interno della pipeline per determinati scenari aziendali. Alcuni di questi controlli sono di dubbia utilità, mentre altri rappresentano uno strumento valido, specialmente nei casi in cui il team aziendale deve prendere decisioni dell'ultimo minuto riguardo al rilascio del prodotto. Il team di progettazione mantiene una versione

rilasciabile del prodotto dopo ogni sprint e il team aziendale decide se distribuirlo a tutti i clienti, a un campione di destinatari o a persone di una specifica località geografica. L'architettura del prodotto che attraversa la pipeline è un elemento chiave nel determinare la sua struttura. Un'architettura del prodotto fortemente accoppiata può generare un modello grafico complesso in cui diverse fasi della pipeline si ostacolano a vicenda prima di arrivare alla produzione. Inoltre, l'architettura del prodotto influisce sulle varie fasi della pipeline e sugli artefatti prodotti in ciascuna fase. Inizialmente, la pipeline genera i componenti, ovvero le unità distribuibili e testabili più piccole del prodotto. Ad esempio, una raccolta creata dalla pipeline può essere considerata un componente ed è ciò che definiamo “la fase dei componenti”. I sottosistemi, costituiti da componenti debolmente accoppiati, sono le unità distribuibili ed eseguibili più piccole. Alcuni esempi di sottosistemi sono i server e i microservizi in esecuzione in un container. Questa fase è definita come “Sottosistema”. A differenza dei componenti, i sottosistemi possono essere valutati durante uno stand-up e testati. Di conseguenza, è possibile configurare la pipeline in modo da assemblare un sistema partendo da sottosistemi debolmente accoppiati nei casi in cui sia necessario rilasciare un sistema nella sua interezza. Questa fase è chiamata “Sistema”.

Questo approccio radicale implica che il sottosistema più veloce debba adeguarsi alla velocità del più lento. "La catena è forte quanto il suo anello più debole" è un detto comune che si usa per avvertire i team che possono essere vittime di questo modello di architettura. Una volta che il sistema assemblato è stato convalidato, viene trasferito all'ambiente di produzione senza ulteriori modifiche, in quella che è la fase finale chiamata “Produzione”.

È importante notare che queste fasi sono più logiche che fisiche, create principalmente per suddividere un problema complesso in parti più piccole e gestibili. Il numero di fasi può variare in base all'architettura e ai requisiti specifici. È essenziale che velocità e qualità siano equilibrati. Il test continuo è una tecnica che prevede l'integrazione di test automatizzati nella pipeline di distribuzione del software, convalidando ogni modifica. I test vengono eseguiti in ogni fase della pipeline per assicurare la validità degli artefatti prodotti. I test unitari e l'analisi statica del codice convalidano i componenti nella fase “Componente”, mentre i test funzionali, di prestazioni e di sicurezza convalidano i sottosistemi nella fase

“Sottosistema”. I test di integrazione, prestazioni e sicurezza convalidano i sistemi nella fase “Sistema”, mentre gli smoke test convalidano il prodotto nella fase “Produzione”. La Continuous Delivery aggiunge valore in tre modi: migliora la velocity, la produttività e la sostenibilità dei team di sviluppo software.

VELOCITY

Il concetto di "**velocity**" denota una velocità deliberata e non dannosa. Le pipeline sono progettate per consegnare prodotti di alta qualità ai clienti. Tuttavia, se i team non sono attenti, le pipeline potrebbero rilasciare codice difettoso più velocemente del previsto. L'automatizzazione delle pipeline di distribuzione del software supporta le organizzazioni nell'adattarsi più efficacemente alle evoluzioni del mercato.

PRODUTTIVITA’

Quando le pipeline gestiscono attività prolungate, come l'invio di una richiesta per ogni modifica che va in produzione, si verifica un aumento significativo della **produttività** rispetto all'intervento umano. Di conseguenza, i team Scrum possono dedicarsi alla qualità dei prodotti anziché dissipare energie nella gestione logistica. Il risultato è che i membri del team risultano più soddisfatti e coinvolti nel loro lavoro, motivati a contribuire costantemente al successo dell'intera squadra.

SOSTENIBILITA’

La **sostenibilità** è cruciale per tutte le aziende, non solo quelle nel settore tecnologico. L'affermazione "Il software sta divorando il mondo" è ormai obsoleta: il software ha già permeato ogni aspetto del mondo contemporaneo. Indipendentemente dal settore, le imprese si avvalgono della tecnologia per distinguersi dalla concorrenza e superare i rivali. Grazie all'automazione, è possibile ridurre o eliminare le attività manuali ripetitive e

suscettibili di errori, permettendo così all'azienda di innovare in modo più efficiente e rapido per soddisfare le esigenze dei clienti.

5.3.4 Continuous Operations

Il Continuous Operations mira a garantire che i sistemi informatici e le applicazioni siano sempre disponibili e funzionanti in modo continuo, senza interruzioni. Questo concetto è spesso associato a pratiche come il Continuous Integration (Integrazione Continua) e il Continuous Deployment (Distribuzione Continua), che consentono agli sviluppatori di rilasciare regolarmente piccole modifiche al codice in produzione in modo rapido e sicuro.
Per raggiungere una modalità di operazioni continue, vengono utilizzati strumenti di monitoraggio dei sistemi, automazione dei processi di gestione delle infrastrutture e delle applicazioni, nonché strategie di resilienza e di ripristino in caso di fallimenti o problemi. L'obiettivo principale del CO è garantire che i servizi digitali siano sempre disponibili e soddisfino le aspettative degli utenti finali, riducendo al minimo i tempi di inattività e migliorando la stabilità complessiva dei sistemi IT.
Il Continuous Operations offre una serie di vantaggi significativi per le aziende che adottano questa pratica:

1. **Affidabilità e disponibilità continua**: Garantisce che i servizi e le applicazioni siano sempre disponibili per gli utenti finali, riducendo al minimo i tempi di inattività e migliorando la continuità operativa
2. **Risposta rapida ai cambiamenti**: Consente alle organizzazioni di adattarsi rapidamente alle nuove esigenze del mercato e agli sviluppi tecnologici, grazie alla capacità di rilasciare rapidamente nuove funzionalità e aggiornamenti
3. **Migliore qualità del software**: La distribuzione continua e l'automazione dei test consentono di individuare e correggere errori in modo tempestivo, migliorando la qualità complessiva del software

4. **Efficienza operativa**: Automatizzando processi ripetitivi e riducendo il bisogno di intervento umano, si ottiene un'efficienza operativa maggiore, liberando risorse per compiti di maggiore valore aggiunto
5. **Risparmio di costi**: La riduzione dei tempi di inattività e l'ottimizzazione delle risorse possono portare a risparmi significativi sui costi operativi
6. **Incremento della soddisfazione del cliente**: Offrire servizi più affidabili e reattivi ai bisogni dei clienti porta ad una maggiore soddisfazione e fidelizzazione degli stessi
7. **Agilità aziendale**: Favorisce una cultura aziendale agile, in grado di adattarsi rapidamente ai cambiamenti del mercato e di innovare in modo continuo.

In definitiva, il Continuous Operations consente alle aziende di rimanere competitive, migliorare la qualità del servizio offerto e rispondere in modo efficace alle sfide del mercato moderno.

CAPITOLO 6 – CASO APPLICATIVO

In questo capitolo verrà trattato il caso applicativo in cui saranno analizzati i vari strumenti utilizzati per portare le metodologie Agile e DevOps all'interno del contesto della Pubblica Amministrazione, rappresentato, nel caso specifico, dall'ufficio Cliente, a cui Il Team Architetture dell'Amministrazione offre supporto, con l'obiettivo ultimo di risolvere tutte le criticità incontrate e ridurre i disservizi.

Infatti, coerentemente con i principi e l'ottica DevOps, è stato possibile rendere il delivery dei prodotti più efficiente, permettere una maggiore scalabilità e disponibilità, una maggiore innovazione e automazione, nonché ambienti operativi più stabili e avere un'ampia visibilità sui risultati del sistema.

Questo è stato possibile seguendo un approccio meticoloso e puntuale, suddividendo il lavoro in due marco-fasi (redazione linee guida e riprogettazione infrastrutturale dell'architettura abilitante), analizzate e descritte dettagliatamente nei paragrafi di seguito citati.

La Pubblica Amministrazione presenta spesso un'infrastruttura IT obsoleta e frammentata, con sistemi legacy che sono difficili da mantenere e aggiornare. Ci sono anche preoccupazioni per la sicurezza dei dati e la mancanza di efficienza operativa.

Per risolvere tali criticità ci viene incontro la metodologia DevOps.

L'approccio messo in atto, infatti, è stato quello di:

1. Creare una **cultura di collaborazione e comunicazione tra team** di sviluppo, operazioni IT e sicurezza informatica. Questo può essere facilitato attraverso l'implementazione di processi DevOps che favoriscono la trasparenza e la condivisione delle responsabilità
2. Utilizzare strumenti di **automazione** per semplificare e velocizzare il rilascio del software e la gestione dell'infrastruttura. Ad esempio, l'automazione dei processi di sviluppo, test e distribuzione riduce il rischio di errori umani e migliora la coerenza e l'affidabilità delle implementazioni

3. Adottare **l'infrastruttura come codice**, consentendo al personale IT di definire e gestire l'infrastruttura utilizzando script e modelli, anziché configurare manualmente i singoli componenti. Questo porta a una maggiore scalabilità, coerenza e tracciabilità dell'infrastruttura
4. Implementare strumenti di **monitoraggio** e telemetria per raccogliere dati sulle prestazioni dell'infrastruttura e del software in tempo reale. Questo consente di identificare rapidamente eventuali problemi e migliorare continuamente le prestazioni e la sicurezza
5. Integrare pratiche di **sicurezza** nel ciclo di vita dello sviluppo e nell'automazione dell'infrastruttura. Ad esempio, incorporare test di sicurezza automatizzati nel processo di integrazione continua per identificare e correggere tempestivamente le vulnerabilità del software
6. Promuovere una **cultura di apprendimento continuo e miglioramento** attraverso la retroazione e il monitoraggio delle prestazioni. Questo include la partecipazione a conferenze, la formazione del personale e la condivisione delle migliori pratiche all'interno dell'organizzazione.

L’esecuzione e lo sviluppo di questo progetto sono stati eseguiti tra Ottobre 2023 e Marzo 2024. In particolare, la stesura delle linee guida, che costituisce la prima fase, è stata realizzata nell'ultimo trimestre del 2023. Successivamente, nel primo trimestre del 2024, è stata realizzata la riprogettazione strutturale dell'architettura abilitante.

6.1 Prima Fase: Linee Guida

6.1.1 Contesto

Il contesto dell'Amministrazione è caratterizzato da un assetto organizzativo suddiviso in una serie di aree operative afferenti ad ambiti differenti: Area Dati, Area Penale, Area Civile e Area Amministrativa. Le varie aree godono di una marcata autonomia che ha portato alla nascita di altrettanti ecosistemi ICT indipendenti con differenze che spaziano dal parco applicativo, passando dai prodotti software utilizzati, per arrivare alle metodologie ed ai processi adottati.

In termini di gestione del codice sorgente, le aree operative si sono fino ad ora affidate ai fornitori di riferimento per la definizione della metodologia di sviluppo, nella maggior parte dei casi demandando loro anche le attività di gestione dei repository del codice sorgente.

Dopo un'accurata analisi si è notato che queste grandi aree non comunicavano tra di loro né condividevano processi.
Si è deciso, dunque, di adottare una metodologia DevOps che andasse a "rompere" i silos, velocizzare i processi e garantire una comunicazione efficace tra le diverse aree operative.
Per fare ciò, le aree DevOps sono state prioritizzate in maniera graduale, focalizzando l'attenzione su quattro aree ritenute di maggior rilevanza, nel seguente ordine:
Source Code Management, Continuous Integration, Continuous Delivery, Continuous Operations.
Sono state, dunque, stilate delle linee guida che andassero a regolamentare e definire le modalità e i processi dell'adozione DevOps per quanto riguarda le suddette aree di riferimento.
Questa fase ha lo scopo di fornire delle linee guida per l'Amministrazione nell'ambito del ciclo di vita del software (Software Development Life Cycle – SDLC), focalizzandosi sulla gestione del codice sorgente (Source Code Management – SCM). Verrà trattato solo il Source Code Management in quanto l'approccio utilizzato è uguale per le altre tre fasi.

Come primo passo, sono stati individuati i cosiddetti **pain points**, ovvero quegli elementi che vengono interpretati come dei limiti o delle debolezze reali dei prodotti o servizi che l'azienda offre.

Nel caso in questione essi sono:

- **Assenza di autonomia**. Gli asset dell'Amministrazione (codice sorgente, artefatti di rilascio) vengono prodotti esternamente e sono conservati secondo modalità poco compatibili con il paradigma DevOps
- **Difformità tra le aree operative**. Assenza di procedure, strumenti e linee guida comuni tra le varie aree operative, che portano i team ad operare come silos isolati, causando ridondanza e inefficienza

L'analisi dello stato attuale dei processi di sviluppo e gestione IT nella pubblica amministrazione rivela un quadro complesso e variegato, caratterizzato da numerose sfide e opportunità di miglioramento. Molti enti pubblici operano ancora con infrastrutture legacy, sistemi frammentati e processi burocratici che rallentano l'innovazione e l'efficienza. Una delle principali criticità riscontrate è la mancanza di integrazione tra i diversi dipartimenti IT, che spesso operano in silos, limitando la collaborazione e lo scambio di informazioni. Questo isolamento porta a inefficienze, duplicazioni di sforzi e difficoltà nel coordinamento dei progetti. Inoltre, i processi di sviluppo software nella pubblica amministrazione tendono ad essere rigidi e poco flessibili, con lunghe tempistiche per la pianificazione e l'implementazione dei progetti. Questo rallentamento è spesso aggravato da approcci di gestione del cambiamento poco efficaci e da una resistenza culturale all'adozione di nuove tecnologie e metodologie. La mancanza di automazione nei processi di test e rilascio rappresenta un ulteriore ostacolo, causando ritardi e aumentando il rischio di errori umani.

In seguito a questo, sono stati delineati gli **obiettivi** dell'organizzazione, ovvero:

- **Ownership degli asset**. Internalizzazione dei repository del codice sorgente, mantenendo ove possibile la compatibilità con le metodologie e le toolchain dei fornitori, ma garantendo maggiore autonomia e controllo degli asset
- **Definizione di un approccio comune**. Individuare tool e procedure da adottare per tutte le aree operative.

6.1.2 Esigenze di business

- **Necessità di internalizzazione del codice,** attraverso l'utilizzo di un repository di codice sorgente che tenga traccia in maniera strutturata dei cambiamenti che di volta in volta vengono messi in produzione. Tale prerequisito costituisce la base per la realizzazione di Pipeline di automazione CI/CD (Continuous Integration/Continuous Delivery), oggetto di linee guida separate.
- **Necessità di standardizzazione ed armonizzazione** modalità di sviluppo, al fine di identificare un approccio comune a tutte le aree operative dell'Amministrazione in termini di strumenti software, strategie di branching, nomenclatura e modalità di integrazione con il contesto multi-fornitore.

Al fine di indirizzare le esigenze di business, sono stati identificati due ambiti di standardizzazione. Gli ambiti identificati sono:

- **Tecnologia**: scelta del tool da adottare su tutte le aree, ovvero scelta di una tecnologia SCM (Source Code Management) abilitante per le esigenze dell'Amministrazione
- **Metodologia**: identificazione delle strategie di branching e delle metodologie di sviluppo ammesse, in ottica di interoperabilità con i fornitori. Dunque, si definirà un metodo di lavoro che sia flessibile e funzionale nel contesto multi-fornitore dell'Amministrazione.

6.1.3 Tecnologia

Per soddisfare i requisiti aziendali, è essenziale identificare un software che possa fornire tutte le funzionalità richieste per l'Amministrazione, in particolare nel contesto del Source Code Management. Ciò include l'adozione di un Sistema di Controllo Versione (VCS) in grado di gestire le seguenti capacità:

- Conservazione del codice sorgente
- Gestione strutturata e collaborativa delle modifiche
- Tracciamento delle versioni

In dettaglio, queste capacità devono rispondere a una serie di requisiti NON funzionali derivanti dall'ambiente operativo multi-fornitore dell'Amministrazione:

- **Il codice sorgente deve essere conservato nei sistemi dell'Amministrazione** per garantire:
    - L'autonomia dell'Amministrazione nella creazione degli artefatti di rilascio tramite il supporto delle pipeline di automazione
    - La possibilità di eseguire vari tipi di test, inclusi test di unità, test di regressione e analisi di sicurezza statica, tra gli altri
    - Una gestione strutturata delle modifiche e delle versioni per rendere lo storico dei cambiamenti accessibile e leggibile
- **Il sistema deve garantire flessibilità nell'integrazione con i sistemi dei fornitori, mantenendo al contempo modalità di accesso sicure**. Ad esempio, deve essere possibile per il fornitore di sviluppo lavorare anche in assenza di connessione con i sistemi dell'Amministrazione.

Nella seguente tabella è possibile osservare la mappatura delle capability del VCS con i relativi requisiti NON funzionali.

Macro Blocco	Capability di riferimetno	Descrizione requisito dell'Amministrazione associato
Source Code Management (SCM)	Conservazione del codice sorgente	Il codice deve essere mantenuto all'interno dell'Amministrazione al fine di avere sempre a disposizione il sorgente degli applicativi in produzione
	Gestione strutturata e collaborativa delle modifiche	Il sistema deve essere in grado di funzionare efficacemente nel contesto multi-fornitore tipico dell'Amministrazione, assicurando contemporaneamente la sicurezza degli asset. In particolare, è richiesta la capacità di operare in un ambiente distribuito
	Tracking delle versioni	Al momento del rilascio di una nuova versione dell'applicativo, deve essere consentita l'associazione di un tag che identifichi chiaramente la versione su un ramo principale, ospitato sui server dell'Amministrazione.

Tabella 3 - Mappatura delle capability del VCS con i relativi requisiti non funzionali

Per adempiere alle esigenze delineate in precedenza, la decisione è stata presa a favore di un sistema di controllo versione di tipo GIT, con particolare attenzione verso il prodotto **GITLAB**.

In particolare:

- È possibile installare GitLab ON-SITE, consentendo così la conservazione del codice sorgente all'interno dei sistemi dell'Amministrazione

- GitLab supporta tutti i comandi standard di Git, ereditando la natura di repository distribuito e garantendo così la flessibilità di integrazione con i sistemi esterni del fornitore di sviluppo. Questa flessibilità è pienamente realizzabile seguendo la metodologia di sviluppo corretta
- GitLab è conforme alle linee guida sull'acquisizione e il riutilizzo di software per le pubbliche amministrazioni.

Al fine di consentire l'applicazione delle metodologie di sviluppo previste, è necessario che la versione di GitLab utilizzata permetta di impostare dei passaggi di approvazione obbligatori sulle "pull request" e di delegarli ad utenze specifiche.

6.1.4 Metodologia

I Sistemi Informativi (SI) di proprietà dell'Amministrazione sono sviluppati e manutenuti da un'ampia pletora di fornitori. Questo paragrafo ha l'obiettivo di definire una metodologia di lavoro strutturata che regoli le fasi di sviluppo e di trasferimento del codice sorgente all'interno dell'Infrastruttura del Ministero tenendo conto del contesto.

In particolare, Il flusso di lavoro deve rispondere alle esigenze di business descritte nel paragrafo 5.1.2, da cui si traggono i seguenti requisiti non funzionali:

- Isolamento tra Dominio Fornitore e Domino Cliente;
- Flessibilità nell'integrazione tra Dominio Fornitore e Domino Cliente;
- Possibilità di intervenire con hotfix in produzione senza pregiudicare le fasi di sviluppo.

L'ecosistema dell'Amministrazione si suddivide in due **domini** principali:

- **Dominio Fornitore**: rappresenta l'insieme degli ambienti che vengono utilizzati dai fornitori durante la fase di sviluppo interna, ovvero prima che il codice venga riversato all'interno dei repository del cliente. Si lascia al fornitore di sviluppo la libertà di decidere quali e quanti ambienti predisporre all'interno della propria infrastruttura

- **Dominio Cliente**: rappresenta l'insieme degli ambienti e dei rami di proprietà del Ministero di cui fa parte il Cliente, indipendentemente dal tipo di installazione (in cloud oppure on-premises).

La strategia di branching è direttamente correlata alla numerosità degli ambienti di sviluppo. In particolare, per il dominio Cliente sono stati definiti i seguenti ambienti:

- **Collaudo**: rappresenta il punto di integrazione tra i sistemi del fornitore di sviluppo ed il repository dell'Amministrazione ed è l'ambiente dove vengono eseguiti i primi test
- **Preproduzione**: ambiente multifunzionale intermedio tra collaudo e produzione. Non è sempre presente nelle progettualità legacy
- **Produzione**: ambiente esposto verso l'utente finale. La distribuzione avviene a valle del superamento dei test nell'ambiente Collaudo e di Preproduzione (se presente).

Una volta individuata la tecnologia, è necessario standardizzare le modalità di lavoro attraverso la definizione della strategia di branching, introducendo delle fasi di code review e step di controllo garantendo, per quanto possibile, la compatibilità con i flussi di sviluppo dei fornitori.

Dopo aver identificato dei casi di sviluppo attraverso il coinvolgimento del personale dell'Amministrazione e dei fornitori e raccolto alcuni feedback sulle strategie di branching attualmente adottate dai fornitori dell'Amministrazione, sono state raccolte delle evidenze, sulla base delle quali sono state definite tre possibili **modalità di lavoro**, che si differenziano per la strategia di branching adottata e la cui scelta dipende dal contesto della singola progettualità:

- **Branched Flow Multi Ramo**: l'Amministrazione detiene un ramo di codice per ogni ambiente
- **Branched Flow Singolo Ramo**: l'Amministrazione detiene solo il ramo principale
- **Git Flow**: l'Amministrazione detiene tutti i rami, inclusi quelli short-lived.

Per quanto riguarda i branch Git utilizzati per lo sviluppo ed il versionamento, essi possono essere classificati in due tipologie:

- **Long-lived:** vengono creati per mantenere le modifiche nel tempo ed agiscono come sorgente dei rami di breve durata
- **Short-lived:** hanno durata temporanea, vengono "staccati" da rami long-lived e vengono utilizzati per supportare le fasi di sviluppo delle nuove feature.

OPZIONE 1 – BRANCHED FLOW MULTI-RAMO

Le modalità operative selezionate per il contesto Amministrazione hanno lo scopo di facilitare la movimentazione del codice sorgente dal Domino Cliente al Dominio Fornitore e viceversa, garantendo una ragionevole autonomia dei fornitori durante la fase di sviluppo e mantenendo il controllo dell'Amministrazione sul codice e sui rilasci.

Nella tabella di seguito sono elencati e descritti i rami che devono essere adottati nel flusso di lavoro dell'Amministrazione, evidenziandone la tipologia, il dominio, gli ambienti e la funzione.

Nome ramo	Tipologia	Dominio	Ambienti correlati	Funzione
main	Long-lived	Cliente	PRODUZIONE	Ramo principale. Contiene la cronologia ufficiale dei rilasci
stage	Long-lived	Cliente	COLLAUDO	Ramo di COLLAUDO. Ha la funzione di gestire l'integrazione di nuove funzionalità da parte del fornitore di sviluppo. Contiene la versione completa delle commit
preprod	Long-lived	Cliente	PREPRODUZIONE	Ramo di PREPRODUZIONE. Ha la funzione di testare, in un ambiente molto simile a quello di produzione, sia le funzionalità

				rilasciate in ambiente di collaudo, sia la corretta funzionalità dell'intero artefatto. Contiene la versione completa delle commit
feature-{nome}	Short-lived	Fornitore	-	Ramo "staccato" dal ramo di COLLAUDO come base per la realizzazione delle nuove feature da parte del fornitore di sviluppo
hotfix-{nome}	Short-lived	Fornitore	-	Ramo "staccato" dal ramo di PRODUZIONE come base per la realizzazione delle hotfix da parte del fornitore di sviluppo.

Tabella 4 - "Rami" da adottare nel flusso di lavoro dell'Amministrazione – Opzione Branched flow multi-ramo

Le fasi che caratterizzano la metodologia Branched Multi Ramo sono indicate di seguito:

- **Sviluppo e rilascio in ambiente di Collaudo**: il fornitore di sviluppo acquisisce il codice nel suo Dominio, effettua gli sviluppi secondo la propria metodologia e il nuovo codice attraverso il push del ramo "feature" nel dominio Cliente. Il codice viene quindi unito al ramo "stage" attraverso una "pull request" approvata dallo stesso responsabile del fornitore di sviluppo
- **Rilascio in Preproduzione:** qualora l'applicativo venga giudicato idoneo per passare in preproduzione, viene eseguita da parte del responsabile del fornitore di sviluppo una nuova pull request sul ramo di "preprod", a valle della cui accettazione da parte di utenti dell'Amministrazione avviene il rilascio in ambiente di Preproduzione. Qualora tutti i test di Preproduzione vadano a buon fine, viene data l'approvazione per il deploy dell'artefatto in Produzione

- **Rilascio in Produzione:** qualora l'applicativo venga giudicato idoneo per passare in produzione, viene eseguita da parte del responsabile del fornitore di sviluppo una nuova pull request sul ramo principale (main), a valle della cui accettazione da parte di utenti dell'Amministrazione avviene il rilascio in ambiente di Produzione
- **Hotfix:** qualora vengano riscontrati dei comportamenti errati dei sistemi o degli artefatti rilasciati in ambiente di produzione, viene clonato il repository dal fornitore di sviluppo all'interno del suo dominio. Il fornitore di sviluppo crea un proprio ramo a partire da quello di "preprod", a partire dal tag attualmente presente in ambiente di produzione, per lavorare le correzioni; dopo aver testato il corretto comportamento, provvede a riportare le modifiche sul dominio dell'Amministrazione attraverso le operazioni di push del ramo creato e di "pull request" verso il ramo di "preprod". A valle dell'accettazione della pull request da parte di utenti dell'Amministrazione, si effettua un rilascio in ambiente di preproduzione per verificare il corretto funzionamento dell'artefatto. A valle dei controlli, il fornitore di sviluppo crea una "pull request" verso il ramo "main" per allineare le modifiche: a valle dell'accettazione della pull request da parte di utenti dell'Amministrazione viene rilasciato l'artefatto in ambiente di produzione.

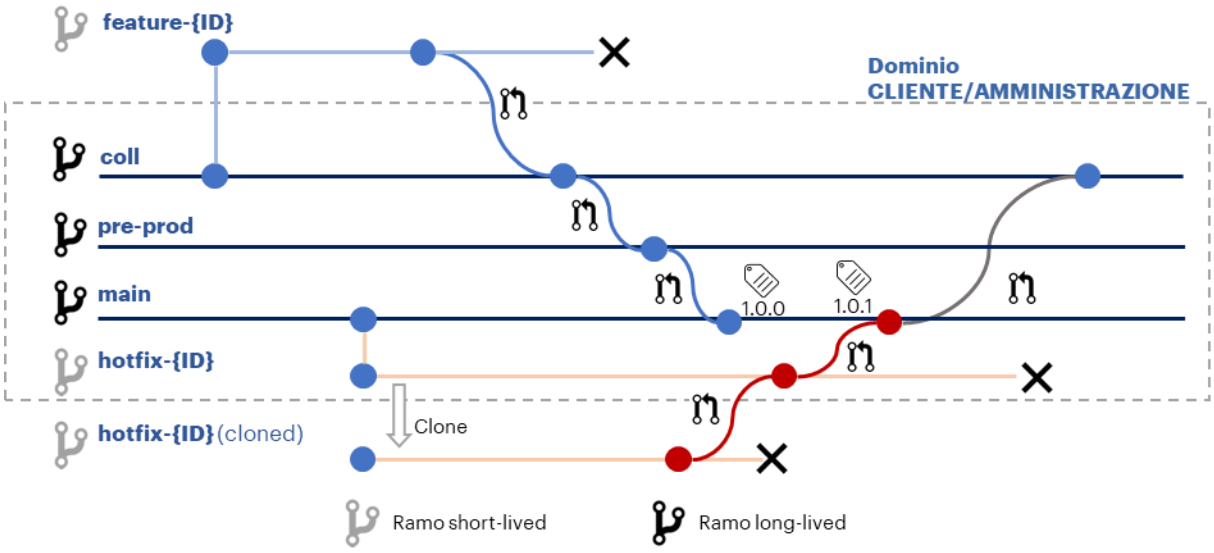


Figura 13 - Opzione Branched flow multi-ramo (1) (Fonte: documento Accenture)

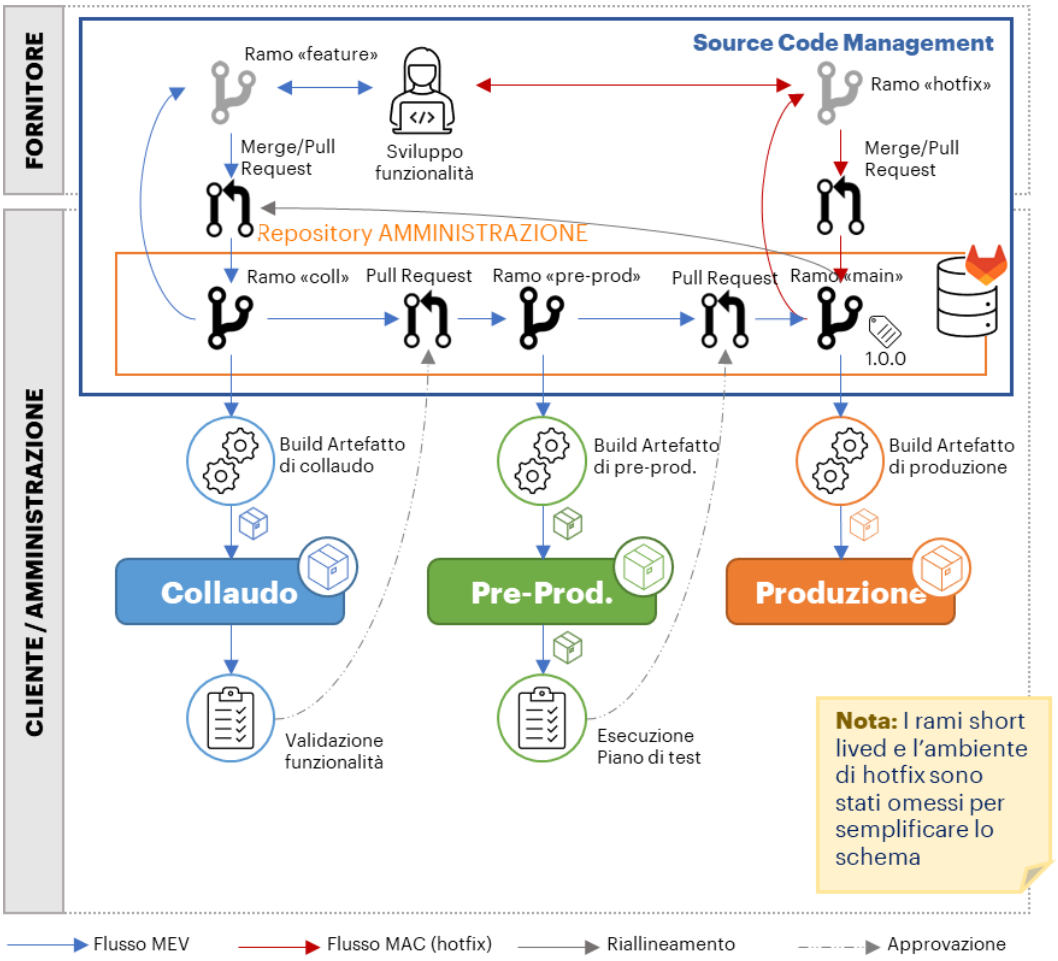


Figura 14- Opzione Branched flow multi-ramo (2) (Fonte: documento Accenture)

OPZIONE 2 – BRANCHED FLOW SINGOLO RAMO

Le modalità operative descritte precedentemente nel paragrafo 5.1.4 sono adatte a progettualità complesse e già avviate. Nelle prime fasi di sviluppo, oppure per specifiche esigenze dettate dalla contingenza, è possibile implementare un flusso semplificato che non includa la presenza dei rami di collaudo e preproduzione. In questo caso, si ha un solo ramo principale su cui vengono eseguite le pull request: si tratta di un più classico approccio "Trunk Flow".

Nella seguente tabella sono elencati e descritti i rami che devono essere predisposti nel flusso di lavoro dell'Amministrazione, evidenziandone la tipologia, il dominio, gli ambienti e la funzione.

Nome ramo	Tipologia	Dominio	Ambienti correlati	Funzione
main	Long-lived	Cliente	PRODUZIONE, PREPRODUZIONE, COLLAUDO	Ramo principale. Contiene la cronologia ufficiale dei rilasci
feature-{nome}	Short-lived	Fornitore	-	Ramo "staccato" dal ramo principale come base per la realizzazione delle nuove feature da parte del fornitore di sviluppo
hotfix-{nome}	Short-lived	Fornitore	-	Ramo "staccato" dal ramo di PRODUZIONE come base per la realizzazione delle hotfix da parte del fornitore di sviluppo.

Tabella 5 - "Rami" da adottare nel flusso di lavoro dell'Amministrazione – Opzione branched flow singolo ramo

Le fasi che caratterizzano la metodologia Branched Singolo Ramo sono indicate di seguito:

- **Sviluppo e rilascio in ambiente di Collaudo**: il fornitore di sviluppo acquisisce il codice nel suo Dominio, effettua gli sviluppi secondo la propria metodologia e il nuovo codice attraverso il push del ramo "feature" nel dominio del Cliente. Il codice viene quindi unito al ramo "main" attraverso una "pull request" approvata dallo stesso responsabile del fornitore di sviluppo
- **Rilascio in preproduzione:** a valle dei test previsti per l'ambiente di collaudo, viene rilasciata l'approvazione per il passaggio dell'artefatto in preproduzione. Non vengono effettuate ulteriori operazioni sul codice

- **Rilascio in Produzione:** a valle dei test previsti per l'ambiente preproduzione, viene rilasciata l'approvazione per il passaggio dell'artefatto nell'ambiente di produzione. Non vengono effettuate ulteriori operazioni sul codice
- **Hotfix:** qualora vengano riscontrati dei comportamenti errati dei sistemi o degli artefatti rilasciati in ambiente di produzione, viene clonato il repository dal fornitore di sviluppo all'interno del suo dominio. Il fornitore di sviluppo crea un proprio ramo a partire da quello di "main", a partire dal tag attualmente presente in ambiente di produzione, per lavorare le correzioni; dopo aver testato il corretto comportamento, si provvede a riportare le modifiche sul dominio Amministrazione attraverso le operazioni di push del ramo. A valle dell'accettazione della pull request da parte di utenti dell'Amministrazione, si effettua un rilascio in ambiente di preproduzione per verificare il corretto funzionamento dell'artefatto; completate le verifiche in ambiente di preproduzione si provvede ad effettuare il rilascio in ambiente di produzione.

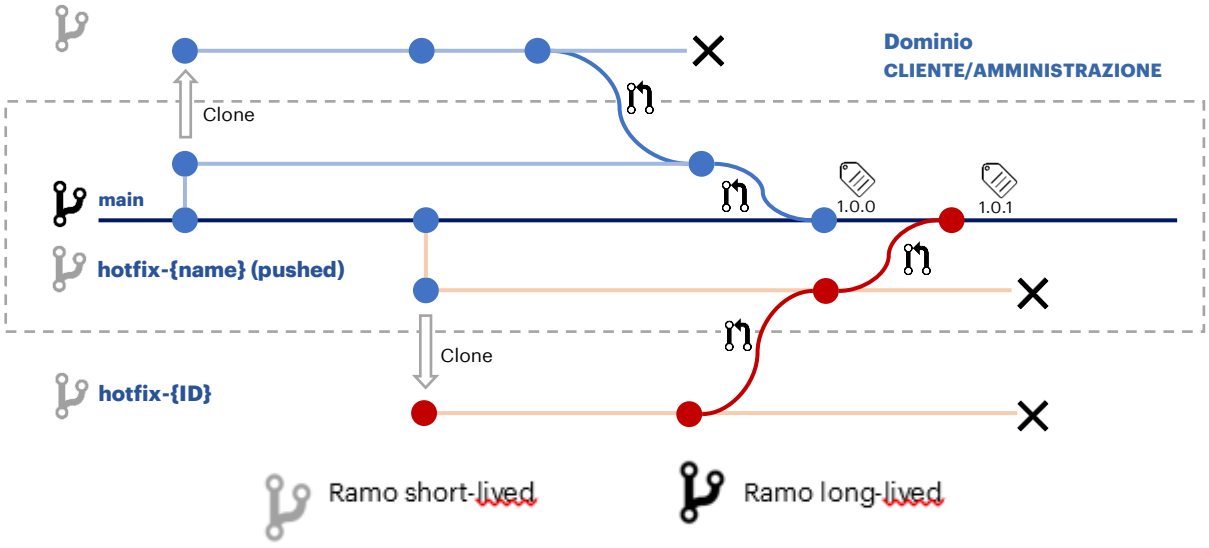


Figura 15 - Opzione Branched flow singolo ramo (1) (Fonte: documento Accenture)

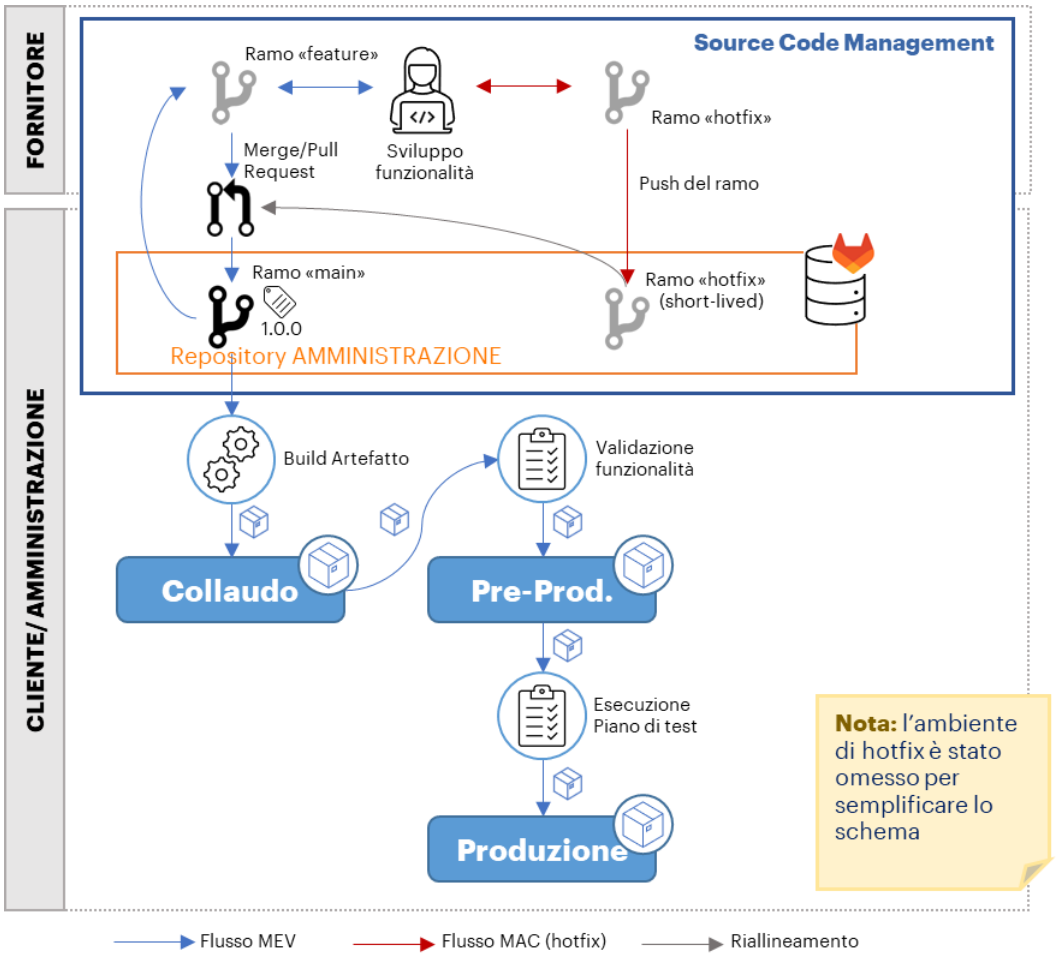


Figura 16 - Opzione Branched flow singolo ramo (2) (Fonte: documento Accenture)

OPZIONE 3 – GIT FLOW

L'Amministrazione detiene sia il ramo principale «main» che il ramo «coll», a cui sono associati due artefatti: uno per la produzione, l'altro per sviluppo/collaudo.

A livello di **branch strategy**, la presente opzione si può così delineare:

- **Ramo «main».** Contiene lo storico delle versioni
- **Ramo «pre-prod»**. Contiene la versione attualmente in fase di sperimentazione/test – (ambiente opzionale)
- **Ramo «coll»**. Contiene lo stato dell'arte del codice e viene utilizzato come sorgente per l'artefatto di collaudo
- **Rami short-lived**. Vengono utilizzati per gestire MEV e MAC.

Inoltre, per quanto concerne la **descrizione del flusso**:

- Questo flusso viene gestito come l'opzione Branched multi-ramo
- Vengono internalizzati tutti i rami short lived di feature e hotfix
- Non è prevista l'esternalizzazione degli sviluppi

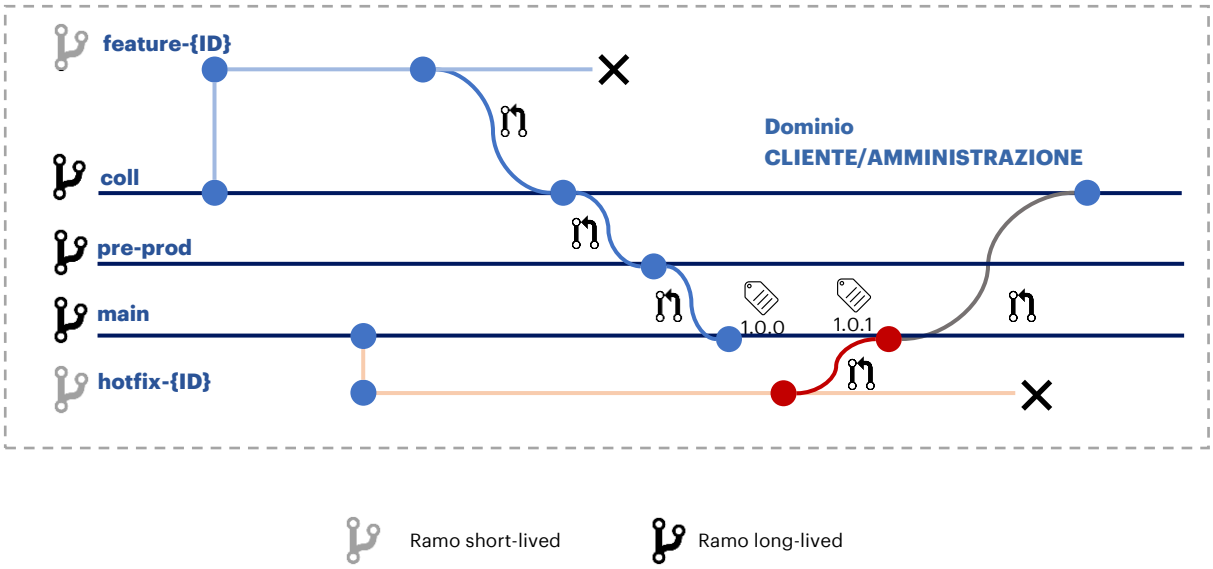


Figura 17 - Opzione Git Flow (1) (Fonte: documento Accenture)

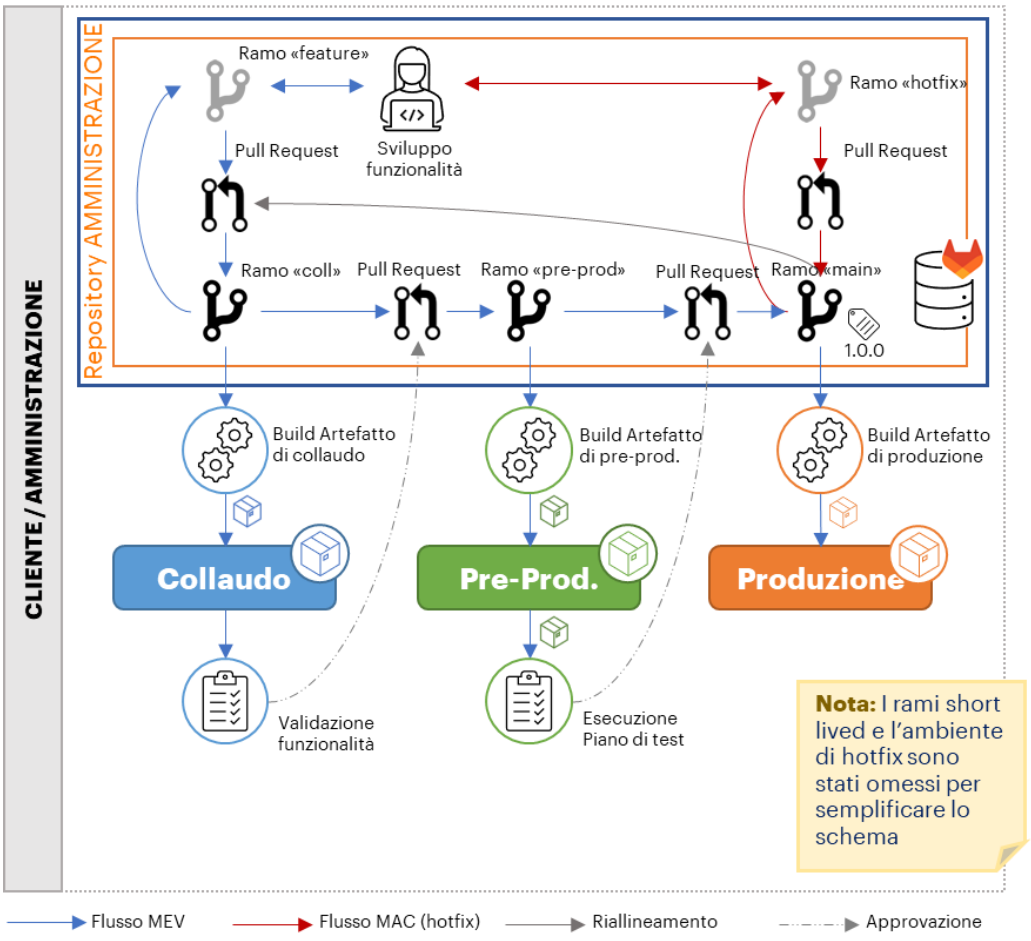


Figura 18 - Opzione Git Flow (2) (Fonte: documento Accenture)

6.1.5 Selezione della modalità di lavoro

Considerato il contesto multi-fornitore dell'Amministrazione, la proposta è di scartare il flusso GitFlow Standard (**OPZIONE 3**), in quanto molto rigido nella condivisione del codice. La scelta, dunque, ricadrà su una delle due opzioni rimaste (Branched flow singolo ramo e Branched flow multi-ramo). In questo paragrafo saranno analizzati i parametri che porteranno alla selezione della modalità di lavoro più opportuna.

La selezione della modalità di sviluppo deve essere guidata sulla base del contesto della specifica progettualità. Al fine di guidare la scelta sono stati definiti alcuni driver principali per la valutazione dei due flussi di sviluppo nel contesto dell'Amministrazione, di seguito elencati:

- **Semplicità di gestione Amministrazione:** indica la facilità nel gestire le operazioni di accettazione dei cambiamenti proposti dal fornitore di sviluppo sui vari rami
- **Parallelizzazione MEV e MAC:** indica il grado di semplicità nel lavorare in parallelo a evolutive e correttive
- **Adattabilità multi-fornitore:** indica quanto l'approccio proposto sia adattabile a un numero elevato di casistiche differenti, tipiche di uno scenario multi-fornitore.

Nella seguente figura, viene rappresentato graficamente il punteggio raggiunto da ciascuno dei due approcci descritti nel paragrafo precedente.

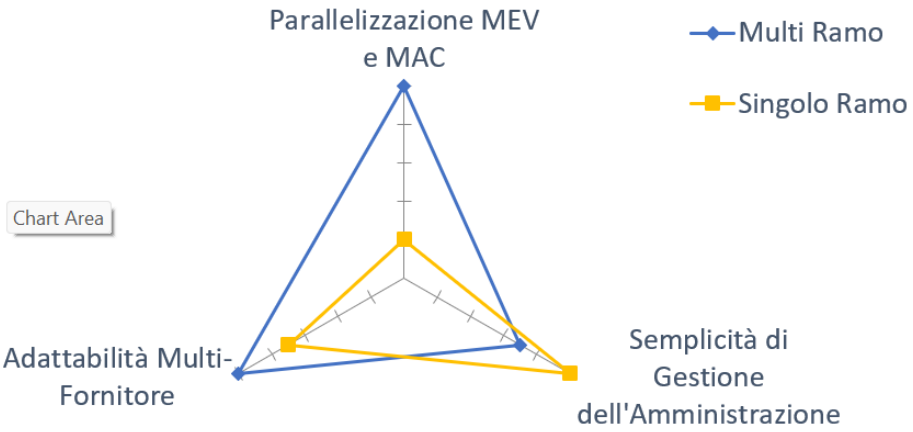


Figura 19 – Grafica del punteggio degli approcci seguiti (Fonte: documento Accenture)

I valori precisi sono riportati nella tabella seguente, su una scala normalizzata da 0 a 100.

Approccio	Parallelizzazione MEV e MAC	Semplicità di Gestione dell'Amministrazione	Adattabilità Multi-Fornitore
Multi Ramo	100	70	100
Singolo Ramo	20	100	70

Tabella 6 - Valori punteggio degli approcci seguiti

Dai dati riportati si determina che **la strategia Multi-Ramo è consigliata nella maggior parte dei casi**, poiché ha ottenuto il miglior punteggio globale. La strategia a singolo ramo andrebbe adottata solo in casistiche particolari, riportate di seguito:

- **Progetti non ancora rilasciati e/o molto giovani**: è possibile adottare la strategia a Singolo Ramo in quanto, mancando la complessità tipica dei progetti a regime, che spesso richiede interventi urgenti in parallelo con gli sviluppi, risulta più facilmente gestibile dal personale amministrativo
- **Progetti a basso impatto di business:** tendenzialmente, si tratta di progetti che rientrano in una delle seguenti casistiche:
    - Servizi interni consolidati e non esposti al cittadino;
    - Progettualità molto semplici;
    - Progetti statici e/o in dismissione.

I criteri appena esposti sono sintetizzati nel diagramma riportato nella figura di seguito. I numeri all'interno dei cerchi rappresentano la stima, in percentuale, di utilizzo delle metodologie specifiche nel quadrante di riferimento.

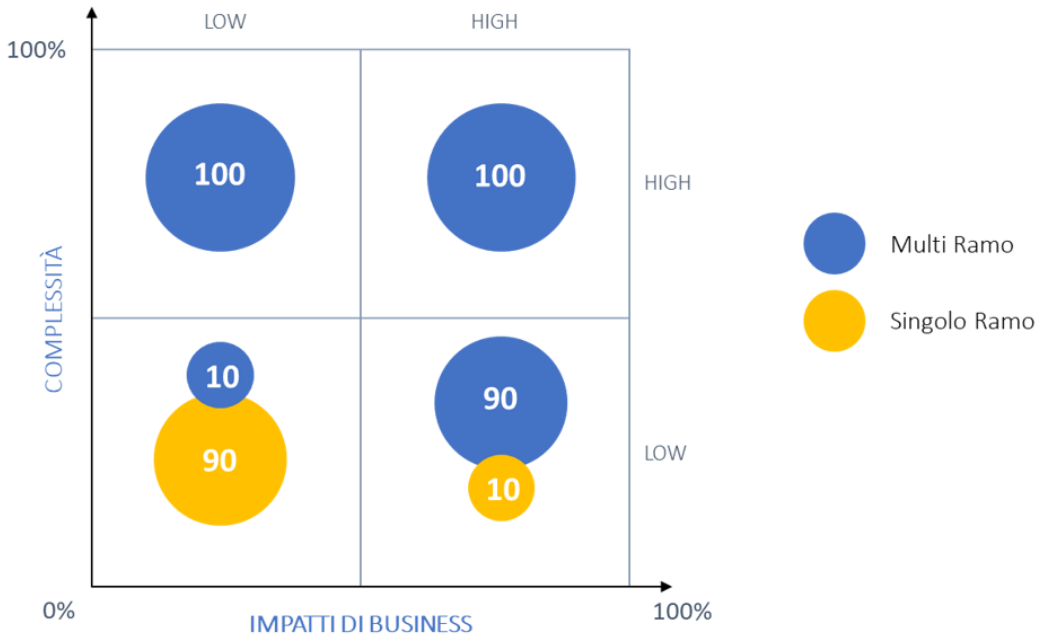


Figura 20 - Stima percentuale di utilizzo degli approcci seguiti in base a due caratteristiche di riferimento (Fonte: documento Accenture)

Le due opzioni sono state confrontate sulla base di quattro driver di valutazione. Sono stati anche evidenziati gli ambiti di applicabilità di ciascuna soluzione.

DRIVER	BRANCHED FLOW MULTI RAMO (OPZIONE 1)	BRANCHED FLOW SINGOLO RAMO (OPZIONE 2)
SEMPLICITA' DI GESTIONE DELL'AMMINISTRAZIONE	**Media**, dovuta alla presenza di più rami da tenere allineati	**Alta**, dovuta alla presenza di un singolo ramo
PARALLELIZZAZIONE MEV E MAC	**Alta**, grazie alla presenza di più linee di codice con due diversi stati di avanzamento	**Bassa**, a causa della presenza di un solo <<stato>> di avanzamento del codice
ADATTABILITA' MULTI FORNITORE	**Alta**, in quanto il codice può essere sviluppato esternamente	**Alta**, in quanto il codice può essere sviluppato esternamente
CONTROLLO DA PARTE DELL'AMMINISTRAZIONE	**Alto**, grazie alla presenza di step di approvazione sulle pull request	**Alto**, grazie alla presenza di step di approvazione sulle pull request
APPLICABILITA'	• Tutte le nuove progettualità • Progetti Legacy ad alto impatto di business • Progetti <<vivi>> con molte MEV in parallelo	• Fasi iniziali di sviluppo • Progettualità non strutturate con deadline stringenti

Tabella 7 - Driver di valutazione e ambiti di applicabilità delle due opzioni

A seguito di tutte queste considerazioni, la strategia Branched Multi Ramo (**OPZIONE 1**) si rivela la più adatta nella maggior parte delle situazioni, anche se la strategia Branched a Singolo Ramo (**OPZIONE 2**) rimane utile nelle progettualità non ancora strutturate grazie alla sua semplicità di gestione.

6.2 Seconda fase: Riprogettazione architetturale dell'infrastruttura abilitante

L'obiettivo prioritario di questa seconda fase di lavoro è quello di realizzare un high-level design ovvero una progettazione di alto livello. Questo, è stato possibile farlo delineando diversi step da seguire, riportati di seguito:

1. Raccolta dei requisiti funzionali e non funzionali
2. Progettazione architetturale
3. Presentazione di due opzioni riguardo la creazione e la manutenzione della propria infrastruttura IT: on-prem e Cloud. È stata, inoltre, eseguita una swot analysis tra le due opzioni, con la conseguente selezione dell'opzione ritenuta più adatta al contesto.

6.2.1 Raccolta dei requisiti

Per quanto riguarda il primo step della seconda fase di lavoro, si è focalizzata l'attenzione soprattutto sui requisiti non funzionali poiché, quelli funzionali corrispondono a quelle che sono le esigenze delle linee guida precedentemente trattate.
I **requisiti funzionali**, accompagnati da una descrizione sono i seguenti:

Requisiti funzionali	**Descrizione**
Internalizzazione del codice sorgente e profilazione degli utenti	Il codice deve essere ospitato nel dominio del cliente. Deve essere prevista la profilazione degli utenti autorizzati ad accedere. Le regole di assegnazione dei profili applicativi proposti sono: • RUP, DEC, assistenti DEC dei contratti, referenti applicativi e Dirigenti • Personale informatico interno addetto alla gestione del repository

	• Fornitori (solo Team Leader + backup)
Resilienza del sistema	Il sistema deve essere resiliente ad eventuali interruzioni di servizio di uno dei data center del cliente, sia temporaneo che permanente, consentendo ai fornitori di sviluppo di lavorare sfruttando le componenti dell'infrastruttura ancora operative
Compatibilità con le Linee Guida per la gestione del codice sorgente (SCM)	Il sistema deve garantire tutte le funzionalità richieste per l'implementazione del flusso di lavoro descritto nelle Linee Guida sulla Gestione del codice sorgente, trasversalmente su tutta l'organizzazione: • Flusso di lavoro a multi branch e branch singolo • Possibilità di proteggere i branch protetti con delle pull request • Push e pull dal ramo del dominio fornitore a quello primario nel dominio del cliente.

Tabella 8 - Requisiti funzionali con relativa descrizione

In seguito all'identificazione dei tre requisiti funzionali, è stato possibile delineare diversi **requisiti non funzionali**:

Requisiti non funzionali	Descrizione
Accesso	Accesso consentito solo all'interno della Rete Unica tramite VPN (Virtual Private Network)
Carico Atteso	Numero di utenti contemporanei stimato inferiore a 200 unità
Disaster Recovery	Possibilità di ripristinare il sistema a valle della failure di uno dei nodi primari
Backup	Il backup dei dati deve essere effettuato su una regione associata, con RPO/RTO* giornalieri
Ridondanza	Sistema distribuito su almeno due data center.

Tabella 9 - Requisiti non funzionali con relativa descrizione

*Definizione alla fine del capitolo.

6.2.2 Progettazione architetturale

Nello step 2, sono state analizzate diverse architetture di riferimento (Reference Architecture) che si adattano ai requisiti dell'utente. Dopo aver analizzato e dettagliato i diversi requisiti, sulla base di questi ultimi, è stata scelta l'architettura più adatta a rappresentare il contesto nel quale ci troviamo e a soddisfare i requisiti dell'utente. Tale architettura è stata prevista solo all'interno della VPN aziendale, è stata integrata con l'AAD (Azure Active Directory) ed è stata personalizzata in base alle esigenze contestuali.
Sono state prodotte due diverse viste:

- La **prima vista** riguarda l'architettura delle componenti e delle connessioni
- La **seconda vista** ha a che fare con l'allocazione.

La **prima vista** riguarda, come già anticipato, l'architettura logica (delle componenti e delle connessioni). Di seguito, è riportata una breve analisi delle caratteristiche architetturali e delle componenti dei nodi GitLab, con la relativa figura che riporta l'architettura logica.

CARATTERISTICHE ARCHITETTURALI

- **Installazione ridondante su due nodi** (primario-secondario) in modalità attivo/passivo
- **Load balancer geografico (NSX ALB)** per distribuire il traffico in base alla geo localizzazione delle macchine
- **Routing interno (NSX)** per gestire e filtrare il traffico tra i due nodi
- **Accesso privato** attraverso una VPN point-to-site tra il PC dello sviluppatore e il data center
- **Integrazione con Azure AAD** per accesso alla piattaforma.

COMPONENTI DEI NODI GITLAB

- **Interfaces:** punti di accesso dell'istanza GitLab

- **Gitaly:** usa RPC per leggere e scrivere dati Git
- **Object Storage:** archiviazione di file condivisa, distribuita tramite integrazione con IBM COS
- **Work Horse:** reverse proxy che gestisce la replica dei dati tra le due istanze
- **PostgreSQL:** memorizza i metadati di GitLab
- **Altri:** gestiscono logiche aggiuntive legate al funzionamento interno di GitLab.

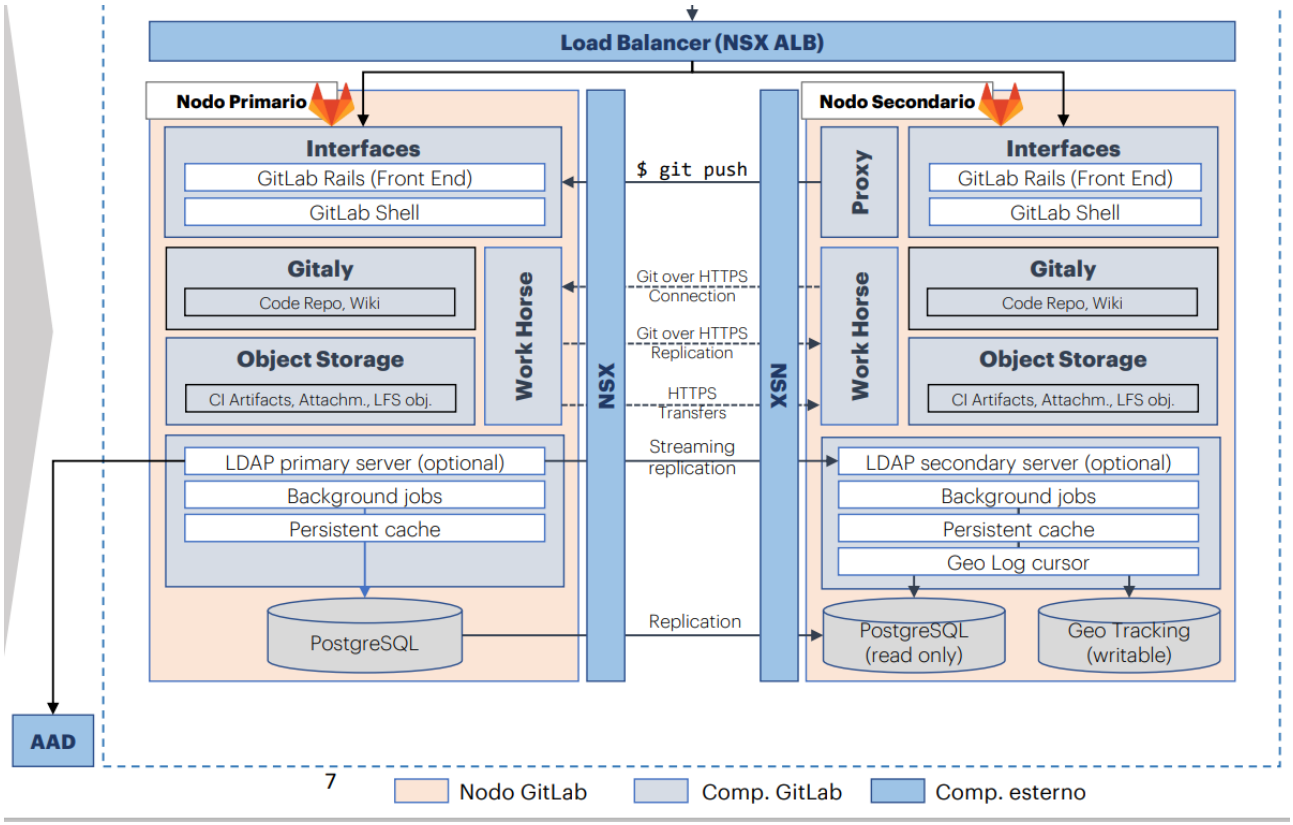


Figura 21 - Vista architetturale logica (delle componenti e delle connessioni) (Fonte: documento Accenture)

La **seconda vista**, invece, riguarda l'allocazione dell'architettura.

ALLOCAZIONE

- **Virtuale Machine (2x):** per l'allocazione dei nodi GitLab su ROMA (primario) e MILANO (secondario)

- **Istanza NSX (2x):** da configurare come punto di ingresso/uscita del traffico interno tra le VM VCF. Ciascuno è allocato nello stesso CED della relativa VM
- **Advanced Load Balancer (1x):** da configurare come punto di ingresso per gli utenti finali. Il componente è ridondato su ROMA e MILANO.

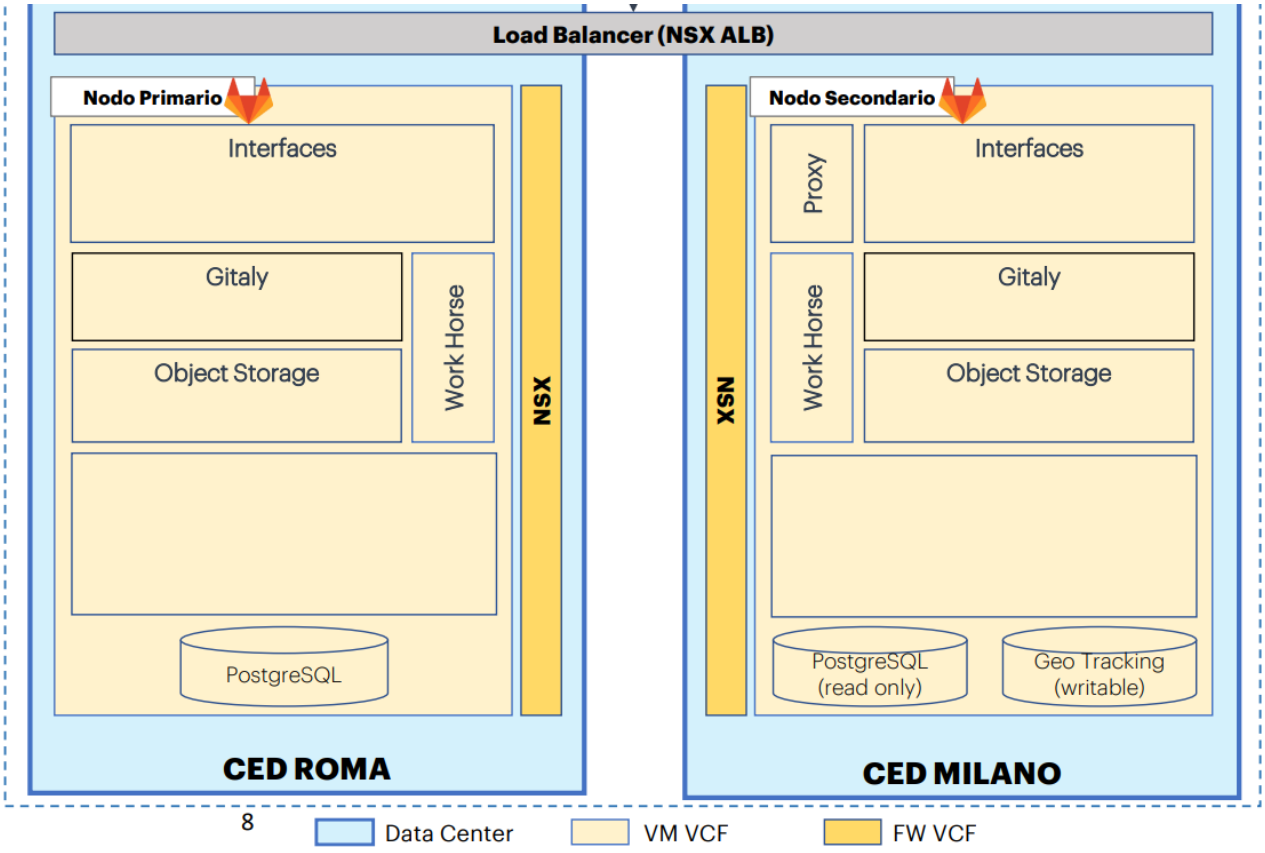


Figura 22 - Allocazione dell'architettura (Fonte: documento Accenture)

6.2.3 Opzioni di creazione e manutenzione dell'infrastruttura IT

Con il termine "software on premise" si fa riferimento alla fornitura di programmi informatici installati e gestiti attraverso computer locali. Deriva dall'inglese "**on the premises**": nelle sedi, nei locali (del titolare della licenza). Il concetto si contrappone all'erogazione di servizi software off premise, in modalità SAAS (software as a service) o in **Cloud computing**, dove

la fruizione del programma avviene attraverso l'accesso a un computer (o a un'architettura di hardware) in remoto, grazie ad una connessione internet.
Affinchè si potesse scegliere l'opzione più adatta tra le due presentate in precedenza, è stata eseguita una swot analysis, grazie alla quale, per ciascuna delle due opzioni, è stato possibile esaminare ed analizzare i punti di forza, i punti di debolezza, i punti di attenzione (minacce) e quelli di opportunità.

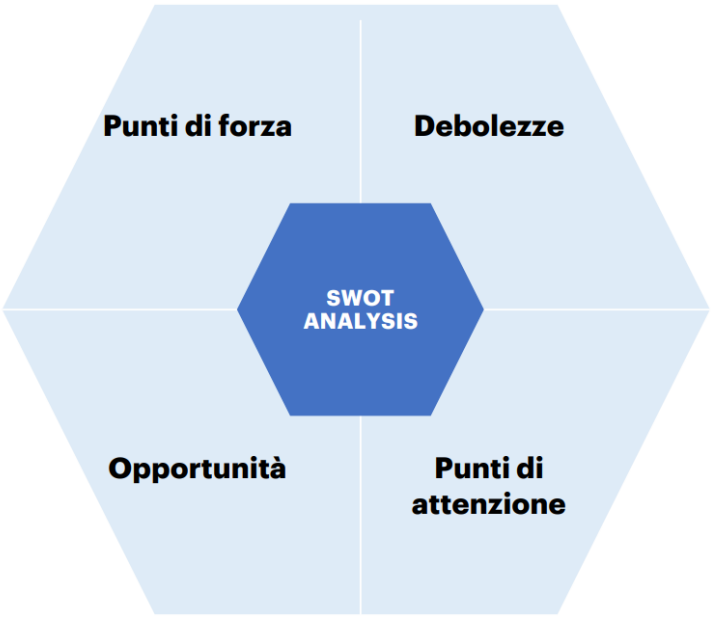


Figura 23 - Swot Analysis (Fonte: documento Accenture)

GITLAB INSTALLATO ON-PREMISES

Punti di forza:

1. **Controllo completo:** Le organizzazioni hanno il controllo completo sull'ambiente in cui GitLab è installato, inclusi l'hardware, il software e la sicurezza

2. **Personalizzazione:** È possibile personalizzare l'installazione on-premises per soddisfare le esigenze specifiche dell'azienda, come l'integrazione con altri sistemi interni o la configurazione delle politiche di sicurezza
3. **Conformità e sicurezza:** Per alcune aziende, mantenere i dati e le applicazioni all'interno delle proprie strutture può soddisfare i requisiti di conformità e fornire un maggiore controllo sulla sicurezza
4. **Connessione di rete:** L'infrastruttura on-premises può sfruttare connessioni di rete dedicate e ad alte prestazioni per garantire un'esperienza utente ottimale.

Debolezze:

1. **Costi iniziali elevati:** L'implementazione di GitLab on-premises richiede un investimento iniziale significativo in hardware, software e risorse umane per la gestione e la manutenzione dell'infrastruttura
2. **Complessità della gestione:** La gestione dell'infrastruttura on-premises può essere complessa e richiedere competenze tecniche specializzate per garantire prestazioni elevate, sicurezza e disponibilità
3. **Scalabilità limitata:** La capacità di scalare l'infrastruttura on-premises può essere limitata dalle risorse hardware disponibili e dai costi associati all'aggiunta di nuovi server o capacità di storage.

Opportunità:

1. **Integrazione con sistemi interni:** L'installazione on-premises offre opportunità per integrare GitLab con altri sistemi interni e applicazioni aziendali
2. **Controllo completo sulla roadmap:** Le organizzazioni hanno il controllo completo sulle decisioni di aggiornamento e sulle nuove funzionalità da implementare, permettendo una maggiore personalizzazione e adattamento alle esigenze aziendali specifiche.

Minacce:

1. **Rischi di sicurezza:** L'infrastruttura on-premises è soggetta a minacce di sicurezza, inclusi attacchi informatici, perdite di dati e violazioni della sicurezza, che richiedono una gestione e una protezione costante
2. **Obsolescenza tecnologica:** L'infrastruttura on-premises potrebbe diventare obsoleta nel tempo, richiedendo aggiornamenti hardware e software per rimanere al passo con l'evoluzione tecnologica
3. **Costi operativi elevati:** I costi operativi associati alla gestione dell'infrastruttura on-premises, inclusi manutenzione, aggiornamenti e gestione dei backup, possono aumentare nel tempo e diventare un onere finanziario significativo per l'azienda.

GITLAB INSTALLATO IN CLOUD

Punti di forza:

1. **Scalabilità:** GitLab in cloud offre una maggiore scalabilità, consentendo alle aziende di aumentare o ridurre rapidamente le risorse in base alle esigenze senza dover investire in hardware aggiuntivo
2. **Agilità:** L'installazione in cloud semplifica il processo di provisioning delle risorse e l'implementazione di nuove funzionalità, consentendo alle organizzazioni di essere più agili e reattive
3. **Costi operativi ridotti:** Il modello di pagamento basato sull'uso consente alle aziende di pagare solo per le risorse effettivamente utilizzate, riducendo i costi operativi complessivi rispetto all'infrastruttura on-premises
4. **Aggiornamenti automatici:** I fornitori di servizi cloud gestiscono gli aggiornamenti del software e la manutenzione dell'infrastruttura, riducendo il carico di lavoro operativo per l'azienda.

Debolezze:

1. **Dipendenza dai fornitori di servizi cloud:** Le aziende che utilizzano GitLab in cloud sono dipendenti dai fornitori di servizi cloud per la disponibilità, le

prestazioni e la sicurezza del servizio, e possono essere soggette a interruzioni di servizio o problemi di connettività

2. **Conformità e sicurezza:** Alcune aziende potrebbero avere preoccupazioni riguardo alla conformità e alla sicurezza dei dati ospitati nel cloud, specialmente se riguardano settori altamente regolamentati.

Opportunità:

1. **Innovazione tecnologica:** GitLab in cloud offre l'opportunità di sfruttare le più recenti innovazioni tecnologiche e funzionalità offerte dai fornitori di servizi cloud
2. **Flessibilità geografica:** Le organizzazioni possono sfruttare la distribuzione geografica dei data center dei fornitori di servizi cloud per fornire servizi a livello globale e raggiungere nuovi mercati.

Minacce:

1. **Sicurezza dei dati:** Esistono preoccupazioni riguardo alla sicurezza dei dati nel cloud, inclusi potenziali attacchi informatici, violazioni della sicurezza e perdite di dati
2. **Costi a lungo termine:** Se non gestiti correttamente, i costi a lungo termine dell'utilizzo di GitLab in cloud possono aumentare nel tempo, specialmente se non viene monitorato attentamente l'utilizzo delle risorse e i costi associati
3. **Vincoli di prestazioni:** Le prestazioni del servizio GitLab in cloud possono essere influenzate da fattori come la connettività di rete e la condivisione delle risorse con altri utenti sulla piattaforma cloud.

Alla luce di tutte queste considerazioni, si può affermare che l’opzione “in cloud” risponde già di sua natura ai requisiti utente. L’architettura “on prem”, invece, andrebbe progettata in maniera tale da soddisfare i requisiti.

Nonostante questo, la scelta è ricaduta sull'opzione "**on prem**" poiché, il cliente, come esigenza di business, ha espresso la volontà di mantenere il server all'interno del suo stesso data center e non lasciarlo su dei server in cloud.

6.3 Obiettivi raggiunti

L'adozione e implementazione del modello DevOps nel settore della Pubblica Amministrazione, può portare ad aziende operanti in tale contesto numerosi vantaggi. Con particolare riferimento all'azienda oggetto di studio del caso applicativo trattato precedentemente, i benefici portati dall'adozione della metodologia DevOps sono stati parecchi, ed essi corrispondono con gli obiettivi che, come team, ci siamo prefissi di raggiungere. Lo scopo di questo paragrafo è proprio quello di illustrare questi obiettivi e giustificare il loro raggiungimento.

- **Riduzione dei tempi di rilascio.** Per ridurre i tempi di rilascio, sono state implementate pipeline di integrazione continua (CI) e distribuzione continua (CD). Queste pipeline hanno automatizzato la costruzione, il test e il deployment del software, eliminando ritardi manuali. Adottando una metodologia Agile, il team ha lavorato in iterazioni brevi e rilasciato frequentemente piccoli aggiornamenti. Inoltre, utilizzare strumenti come Jenkins, GitLab CI, o Azure DevOps ha notevolmente facilitato questo processo, garantendo un flusso di lavoro continuo e veloce
- **Miglioramento della qualità del software.** Per migliorare la qualità del software è stata implementata una suite di test automatizzati che comprende unit test, integration test e end-to-end test. Questo ha assicurato che ogni modifica al codice sia stata verificata immediatamente
- **Aumento dell'efficienza operativa.** Per aumentare l'efficienza operativa, sono state automatizzate le attività ripetitive e i processi manuali. L'uso di infrastruttura come codice tramite strumenti come Terraform ha permesso di gestire e scalare

l'infrastruttura IT in modo programmatico. Implementare strumenti di gestione della configurazione come Ansible ha contribuito a ridurre significativamente il tempo necessario per configurare nuovi ambienti, migliorando la coerenza e l'efficienza

- **Maggiore sicurezza e compliance.** Integrare la sicurezza nel ciclo di vita dello sviluppo del software (DevSecOps) si è rivelato cruciale così come l'aver implementato scanner di sicurezza automatizzati, nelle pipeline CI/CD ha aiutato a identificare e correggere le vulnerabilità fin dall'inizio. Inoltre, sono state stabilite politiche di sicurezza e compliance e sono stati utilizzati strumenti di monitoraggio e auditing, i quali assicurano che tutte le modifiche siano tracciabili e conformi alle normative
- **Miglioramento della soddisfazione dei clienti.** Raccogliere e analizzare costantemente il feedback degli utenti tramite sondaggi, recensioni e strumenti di analytics, come Google Analytics ha aiutato a comprendere meglio le loro esigenze. Implementare cicli di rilascio rapidi e iterativi ha consentito di apportare modifiche e miglioramenti in tempi brevi, rispondendo alle richieste dei clienti
- **Gestione efficace delle emergenze.** È stato possibile implementare sistemi di monitoraggio in tempo reale, che hanno consentito di rilevare problemi e anomalie rapidamente. Sono stati, inoltre, predisposti piani di business continuity e disaster recovery (come si può riscontrare nel caso applicativo), inclusi back-up regolari e test periodici, i quali assicurano una risposta rapida ed efficace alle emergenze
- **Trasparenza e responsabilità.** Sono stati utilizzati sistemi di monitoraggio e reporting, per fornire visibilità sulle performance delle applicazioni e sui progressi dei progetti. L'aver creato strumenti "trasparenti" e soprattutto accessibili a tutti gli stakeholder ha sicuramente facilitato la comunicazione delle metriche chiave e degli indicatori di performance. Stabilendo politiche di governance che includono responsabilità chiare è stato possibile assicurare che le operazioni fossero tracciabili e conformi alle normative, migliorando la fiducia e la trasparenza.

6.4 Drivers di valutazione e interpretazione dei risultati

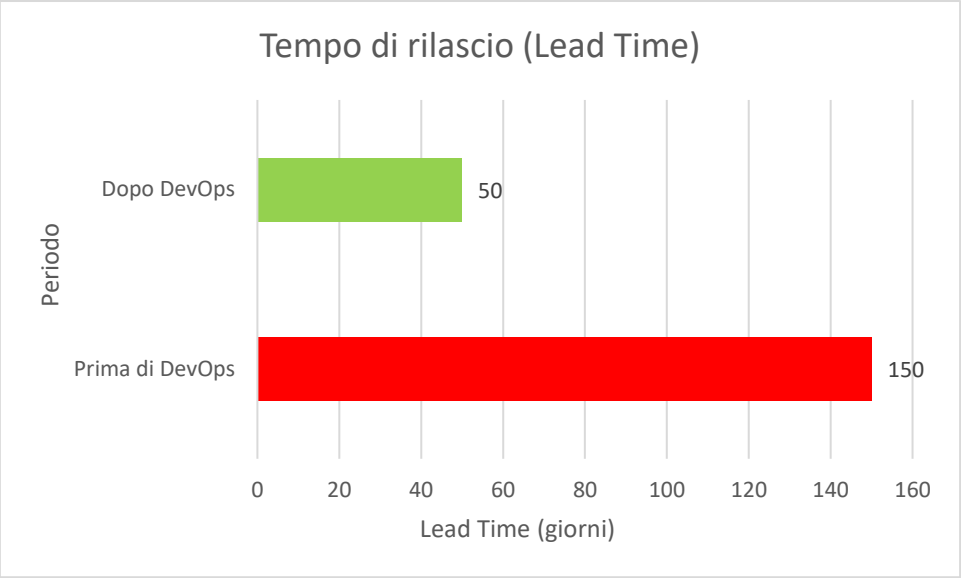
Come illustrato in questo capitolo e anche nei precedenti, i vantaggi che la metodologia DevOps offre alle aziende che decidono di introdurla sono numerosi. Con riferimento al caso applicativo in questione, infatti, sono stati presi in considerazione diversi drivers di valutazione, i quali indicano e dimostrano a livello quantitativo i benefici del metodo DevOps.

All'interno di questo paragrafo verranno elencati e descritti i drivers di valutazione, si mostreranno i risultati ottenuti, i quali verranno rapportati con i risultati precedenti all'adozione di DevOps.

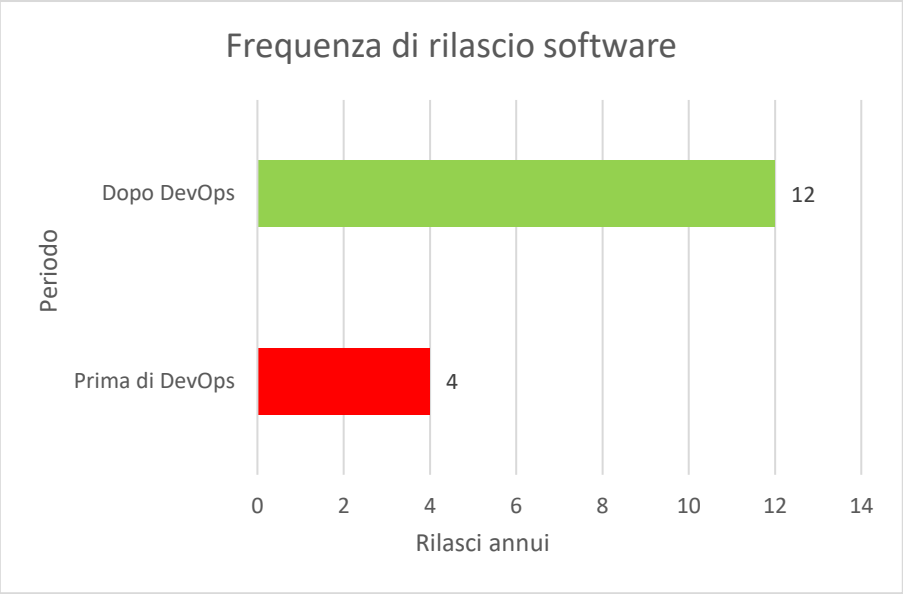
È necessario specificare che i risultati mostrati si riferiscono a due semestri differenti. Infatti, il periodo di riferimento precedente all'adozione DevOps è quello relativo al semestre Marzo-Settembre 2023. L'adozione della metodologia DevOps, invece, è relativa al mese di Ottobre 2023; di conseguenza i risultati che mostrano l'efficacia di tale modello sono stati raccolti nel semestre Ottobre 2023-Marzo 2024.

Di seguito i drivers presi in considerazione nell'analisi:

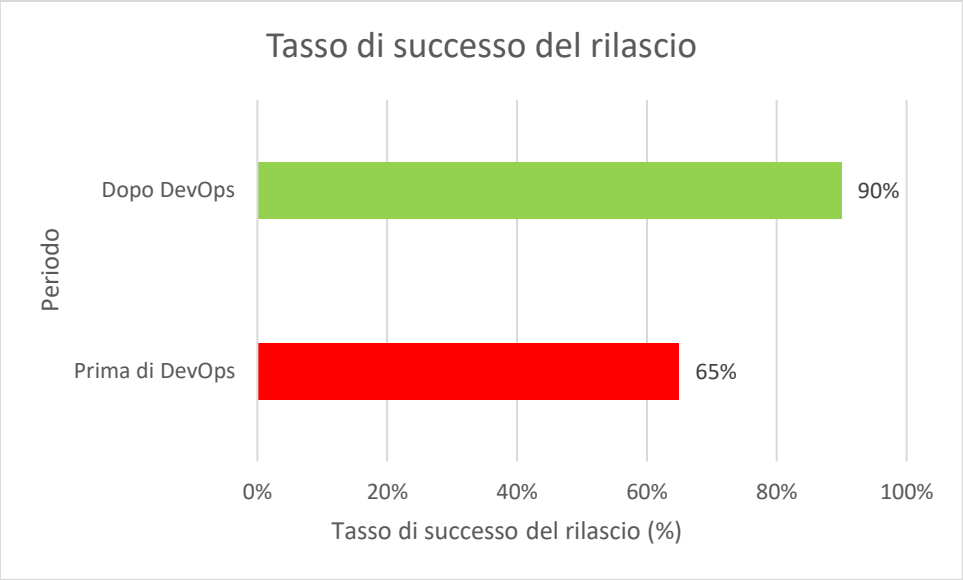
1. **Tempo di rilascio (lead time):** Metrica cruciale per valutare l'efficienza e la velocità di un processo di sviluppo software. Esso si riferisce specificatamente al tempo che passa dall'inizio dello sviluppo di una funzionalità o una modifica fino alla sua effettiva distribuzione in produzione. I risultati ottenuti sono significativi, in quanto, in seguito all'adozione di DevOps, si è registrata una riduzione del lead time di 100 giorni. Con l'adozione del DevOps, infatti, i processi sono automatizzati e semplificati, riducendo significativamente il tempo necessario per sviluppare, testare e rilasciare nuove funzionalità o aggiornamenti.



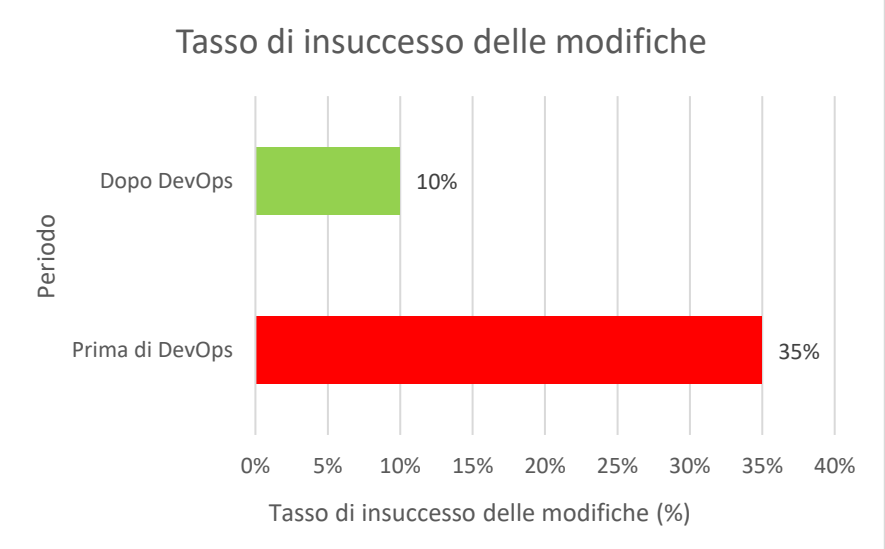
2. **Frequenza di rilascio.** Si riferisce a quanto spesso nuove versioni del software, aggiornamenti o funzionalità vengono distribuite agli utenti. Questa metrica è cruciale per comprendere l'agilità e la reattività del team di sviluppo software nel rispondere alle esigenze del cliente, correggere bug e introdurre nuove funzionalità. Prima dell'adozione del DevOps, è stata pianificata e coordinata manualmente una cospicua parte di rilasci e questo portava a cicli di rilascio lunghi e complessi. Con l'implementazione di CI/CD e test automatizzati, il team ha automatizzato gran parte del processo, permettendo rilasci più frequenti e riducendo il tempo necessario per passare dall'implementazione di una nuova funzionalità alla sua disponibilità per gli utenti. Questo ha portato ad un aumento della frequenza dei rilasci su base annua (da 4 a 12).



3. **Il tasso di successo di rilascio** (o release success rate) è una metrica che misura l'efficacia e l'affidabilità del processo di rilascio del software. Questa metrica indica la percentuale di rilasci (o versioni) di software che sono stati completati con successo senza introdurre problemi significativi, errori critici o malfunzionamenti che richiedono interventi correttivi immediati. Un rilascio è considerato di successo se il software funziona come previsto una volta distribuito, senza causare interruzioni o gravi malfunzionamenti agli utenti finali. Si calcola dividendo il numero di rilasci di successo per il numero totale di rilasci effettuati in un determinato periodo di tempo, moltiplicato per 100 per ottenere una percentuale. L'implementazione di CI/CD e test automatizzati ha permesso di ridurre significativamente gli errori, aumentando la qualità e la stabilità dei rilasci. Nello specifico, si è osservato un aumento significativo di tale tasso dal 65% al 90% in seguito all'adozione della metodologia DevOps.

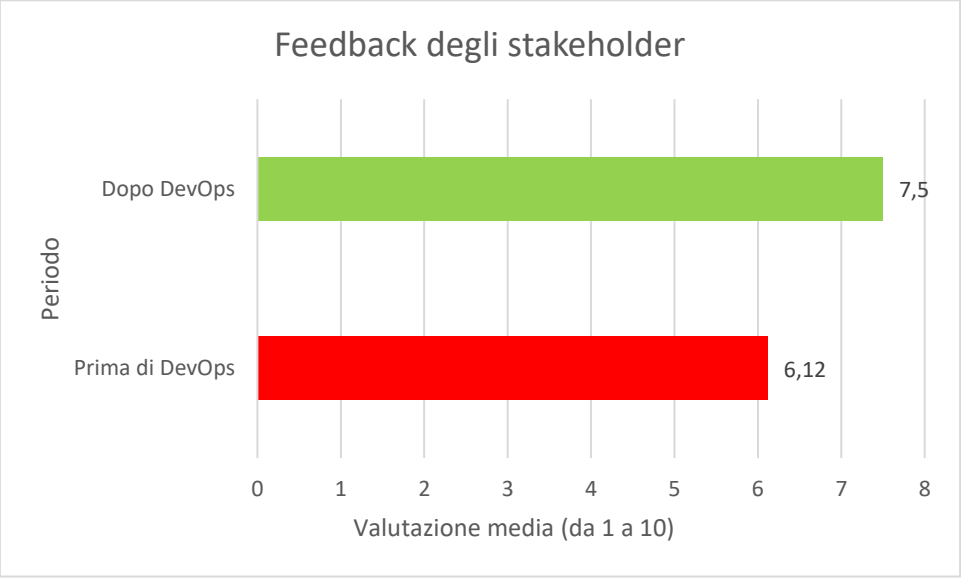


4. **Tasso di insuccesso delle modifiche.** Il tasso di insuccesso delle modifiche misura la percentuale di modifiche al codice, distribuzioni o aggiornamenti che portano a un fallimento, un errore o richiedono un intervento correttivo. Questo tasso si è ridotto sensibilmente in seguito all'adozione DevOps (dal 35% al 10%) grazie all'adozione di diverse pratiche e strategie. Una su tutte il DevSecOps, che ha permesso di integrare la sicurezza nel ciclo di vita del software, eseguendo scansioni di sicurezza automatizzate e verifiche di conformità per rilevare vulnerabilità prima che il codice raggiungesse la produzione. È stata, inoltre, adottata una gestione delle versioni rigorosa per tracciare le modifiche e facilitare i rollback in caso di problemi.



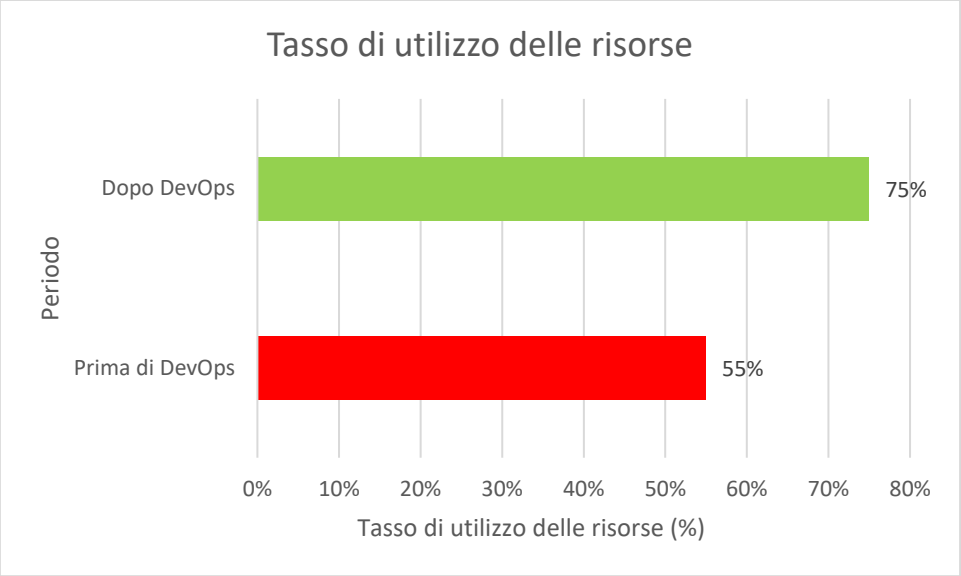
5. **Feedback degli utenti.** Utilizzando come strumento Microsoft Forms, è stato possibile somministrare sondaggi e questionari periodici a diversi stakeholder (come clienti e team di sviluppo) per raccogliere feedback sulla metodologia DevOps. Tali sondaggi e questionari specifici sono stati creati per diverse aree del processo DevOps (ad esempio CI/CD, automazione dei test, deployment) in modo da ottenere feedback dettagliati per ciascun aspetto. I risultati raccolti, espressi in media su una scala numerica da 1 a 10, evidenziano come, in relazione allo scenario antecedente all'adozione di DevOps, la soddisfazione degli stakeholder coinvolti nel processo è notevolmente aumentata.

   In particolar modo, come si può notare nel grafico successivo, la soddisfazione media degli utenti coinvolti è aumentata da una media di 6,12 su 10 a 7,5 su 10.



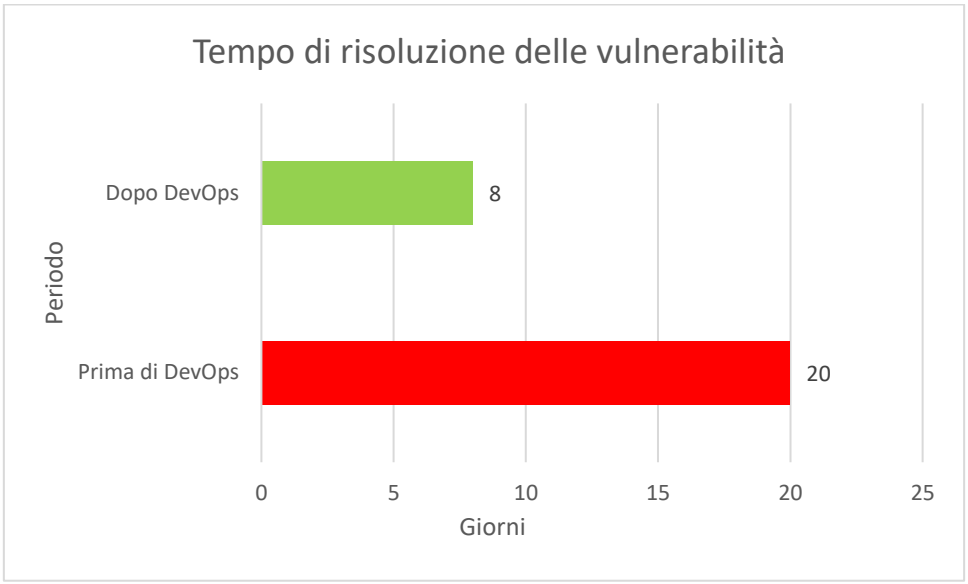
6. **Utilizzo delle risorse.** Il DevOps ha contribuito nel migliorare l'utilizzo medio delle risorse attraverso diverse pratiche e strumenti che hanno promosso l'efficienza, l'automazione e la scalabilità. Questo è stato possibile grazie al Continuous Integration/Continuous Delivery (CI/CD). Infatti, automatizzando il processo di build, test e deploy, si è ridotto il tempo di inattività e si è assicurato l'utilizzo efficiente delle risorse. Queste ultime sono state ridotte o aumentate automaticamente in base alla

domanda (auto-scaling) e questo ha assicurato che le risorse non fossero sottoutilizzate durante i periodi di bassa domanda e che ci fosse abbastanza capacità durante i picchi. Per ultimo, ma non di minore importanza, ha svolto un ruolo fondamentale il Cloud computing. Sfruttando, infatti, le capacità del cloud, è stato possibile ridimensionare le risorse in base alla domanda, pagando solo per ciò che si utilizza e riducendo gli sprechi. In definitiva, come si può evincere dai grafici sottostanti, si è registrato un incremento dell'utilizzo delle risorse dal 55% al 75%.

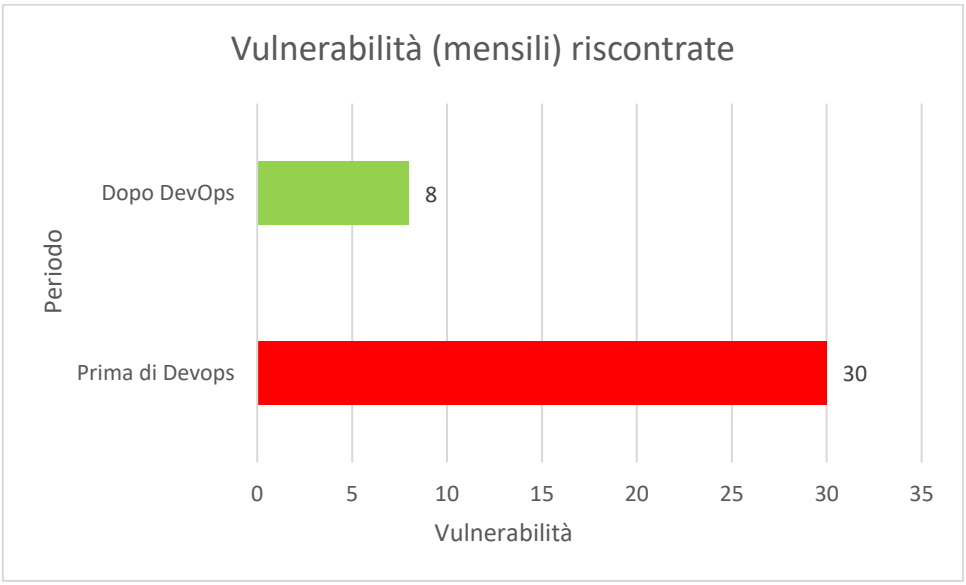


7. **Sicurezza e compliance.** Questo driver di valutazione si focalizza sul tempo di risoluzione delle vulnerabilità. È necessario specificare che quest'ultimo varia ampiamente in base alla gravità della vulnerabilità, alla complessità del sistema, alla disponibilità delle risorse e all'efficienza dei processi di gestione delle vulnerabilità. Si è riscontrato, però, che, in seguito all'adozione della metodologia DevOps, tale parametro sia drasticamente diminuito. Infatti, nello scenario pre-adozione DevOps il tempo associato alla risoluzione delle vulnerabilità è risultato essere mediamente di 20 giorni. Dopo l'adozione DevOps, invece, si è registrato un tempo di risoluzione medio pari a 8 giorni.

Con DevSecOps, la sicurezza è integrata nel ciclo di vita del software. Questo ha consentito di ridurre il numero di vulnerabilità e accelerare i tempi di risoluzione grazie a una rilevazione e risposta più rapide.



Altro dato interessante è quello relativo al numero delle vulnerabilità riscontrate. Anche in questo caso, è stata riscontrata una netta diminuzione rispetto allo scenario antecedente all'adozione della metodologia DevOps. Nello specifico, le vulnerabilità riscontrate mensilmente sono passate da 30 ad 8.



CAPITOLO 7 - CONCLUSIONI

L’ adozione della metodologia DevOps nella pubblica amministrazione ha rappresentato un significativo passo avanti nella modernizzazione dei processi e nella qualità dei servizi offerti ai cittadini. Attraverso questa esperienza, sono emersi numerosi benefici che hanno rafforzato la convinzione dell'importanza di un approccio DevOps per il futuro.

Innanzitutto, l'automazione dei processi di sviluppo, test e rilascio ha notevolmente ridotto i tempi di consegna dei progetti, consentendo una risposta più rapida alle esigenze dei cittadini. Questo miglioramento ha non solo aumentato l'efficienza operativa, ma ha anche incrementato la capacità della pubblica amministrazione di adattarsi ai cambiamenti e alle nuove richieste normative in maniera tempestiva.

La collaborazione e la comunicazione tra i vari team coinvolti sono state notevolmente migliorate. L'approccio DevOps ha promosso una cultura di integrazione e collaborazione, riducendo i silos organizzativi e favorendo una maggiore condivisione delle conoscenze. Questo ha portato a un ambiente di lavoro più coeso e produttivo, dove il feedback continuo ha giocato un ruolo cruciale nell'ottimizzazione dei processi e nella risoluzione dei problemi.

Inoltre, l'implementazione di pratiche DevSecOps ha garantito che la sicurezza fosse integrata in tutte le fasi del ciclo di vita del software. Questo ha aumentato la fiducia nella sicurezza dei servizi offerti e ha assicurato la conformità alle normative vigenti, proteggendo tutti i dati sensibili.

L'esperienza ha anche evidenziato l'importanza della formazione continua e dello sviluppo delle competenze del personale. Gli investimenti in programmi formazione e apprendimento hanno permesso ai dipendenti di acquisire nuove competenze tecniche e gestionali, fondamentali per il successo dell'approccio DevOps.

Infine, l'adozione di tecnologie cloud e architetture a microservizi ha migliorato la scalabilità e la flessibilità delle infrastrutture IT, rendendo possibile una gestione più efficace delle risorse e una maggiore agilità nella risposta alle esigenze operative.

Nonostante siano decisamente inferiori ai benefici, sono state riscontrate anche diverse criticità. Queste riguardano prettamente:

- **Resistenza al cambiamento culturale**. Difatti, diversi dipartimenti della Pubblica Amministrazione hanno ancora difficoltà a recepire i cambiamenti e le novità introdotte dalla metodologia DevOps, in quanto questi implicano l'introduzione di pratiche Agili in un contesto abituato a gestire i progetti adottando il modello tradizionale (Waterfall). Un ulteriore ostacolo è stato rappresentato dalle strutture rigide e gerarchiche che presentano i dipartimenti della Pubblica Amministrazione. Dunque, la promozione di una cultura della collaborazione e dell'innovazione ha richiesto tempi e sforzi significativi
- **Formazione e competenze**. La mancanza di competenze specifiche nel personale della Pubblica Amministrazione è un altro ostacolo rilevante. L'adozione di DevOps richiede una formazione continua e l'acquisizione di nuove competenze tecniche e operative. Per ovviare a questo problema, si è ritenuto necessario costituire diversi programmi di formazione e sviluppo professionale con lo scopo di far familiarizzare l'intero personale con le pratiche DevOps
- **Integrazione di sistemi legacy**. È stato riscontrato che all'interno del ministero appartenente al cliente la maggior parte dei sistemi legacy sono obsoleti e non si integrano facilmente con le pratiche DevOps. La migrazione o l'integrazione di questi sistemi ha rappresentato una sfida tecnica complessa che ha rallentato il processo di adozione
- **Compliance e sicurezza.** La Pubblica Amministrazione deve necessariamente rispettare rigorosi standard di sicurezza e compliance, che hanno complicato l'adozione di pratiche DevOps. Infatti, una sfida cruciale è stata quella di garantire che le nuove metodologie rispettino le normative vigenti senza compromettere la sicurezza dei dati.

Questo lavoro di tesi è il risultato di un periodo di tirocinio svolto all'interno del Team Architetture di Accenture. Ha rappresentato un'esperienza formativa fondamentale, permettendomi di applicare le conoscenze teoriche acquisite durante il mio percorso di studi a contesti reali e dinamici del mondo del lavoro. Questo periodo di stage ha evidenziato l'importanza dell'integrazione tra teoria e pratica, nonché il valore del lavoro di squadra e della collaborazione interfunzionale. Con riferimento al progetto trattato dettagliatamente nel caso applicativo, mi è stato possibile, in stretta collaborazione con i membri del team, svolgere differenti attività così organizzate:

- **Redazione di documenti che forniscono delle linee guida per l'amministrazione nell'ambito del ciclo di vita del software**. Tali documenti, indirizzati principalmente a personale informatico interno all'amministrazione e sviluppatori e progettisti esterni all'amministrazione hanno come obiettivo quello di fornire delle linee guida per l'amministrazione nell'ambito del ciclo di vita del software con particolare riferimento al Source Code Management (SCM). Il mio contributo in questa fase ha riguardato prevalentemente la stesura di una parte dei documenti, nonché la loro pubblicazione e revisione. Per quanto riguarda quest'ultimo aspetto, prima della pubblicazione ufficiale di un documento questo è soggetto a più revisioni, realizzate in collaborazione con diversi comitati e società di software quali il Comitato Interuniversitario Nazionale per l'Informatica (CINI) e Red Hat. A questo scopo, il team Architetture di cui faccio parte ha organizzato durante l'intera durata del progetto due meeting settimanali della durata di un'ora ciascuno entrambi finalizzati al confronto e all'avanzamento di proposte, idee, soluzioni o limitazioni relative all'avanzamento dell'attività di redazione delle linee guida;

- **Raccolta, formalizzazione e validazione dei requisiti funzionali e non funzionali.** In questa fase ho contribuito partecipando attivamente assieme all'intero team ad incontri e focus group appositamente dedicati all'individuazione delle esigenze di business del cliente. Sulla base di queste, è stato possibile individuare una serie di

requisiti funzionali che descrivono e definiscono il comportamento del sistema (ciò che il sistema deve fare). Per la loro identificazione sono stati necessari i feedback ricevuti dagli stakeholder durante la redazione delle linee guida per il Source Code Management. La raccolta dei requisiti funzionali è avvenuta anche grazie alla somministrazione di appositi questionari agli stakeholder coinvolti. Dopo questa fase, si è passati all'analisi di tali requisiti volta a identificare conflitti, duplicazioni ed ambiguità. Tutto questo ha portato alla definizione dei requisiti non funzionali che l'architettura deve fornire al fine di supportare le esigenze di business. Essi definiscono i criteri che giudicano il funzionamento di un sistema, ovvero come il sistema deve comportarsi e hanno incluso aspetti come prestazioni, sicurezza, usabilità, affidabilità, scalabilità;

- **Validazione dei requisiti formalizzati.** Per quanto riguarda questa attività, ho partecipato a meeting prima interni (peer review), in cui assieme al team di lavoro ho effettuato una puntuale esamina dei requisiti con lo scopo di identificare errori, ambiguità e eventuali omissioni e successivamente esterni (technical review), in cui diversi esperti tecnici hanno valutato la fattibilità dei requisiti, fornendoci conseguentemente i loro preziosi feedback. Inoltre, l'utilizzo di liste di controllo ha permesso di verificare costantemente che tutti i requisiti rispettassero le dovute caratteristiche, ovvero che essi fossero chiari, completi, consistenti, corretti, verificabili, tracciabili e modificabili;

- **Progettazione delle specifiche funzionali partendo dai requisiti definiti.** Questa attività riguarda il processo di traduzione dei requisiti raccolti in una descrizione dettagliata di come il sistema software deve comportarsi per soddisfare tali requisiti. Questa fase è cruciale perché fornisce una guida chiara e specifica per gli sviluppatori e assicura che il prodotto finale corrisponda alle aspettative degli stakeholder. Il mio apporto in questa attività ha riguardato prettamente la revisione dettagliata del relativo documento e, dunque, la rilevazione di errori e incongruenze. Sono state,

inoltre, identificate le interdipendenze e le priorità tra i requisiti e stabilito un formato che includesse sezioni come ambito, scopo, descrizione dettagliata delle funzionalità, vincoli e criteri di accettazione;

- **Selezione della modalità di lavoro.** Dopo aver individuato le esigenze di business mi è stato possibile identificare la strategia di branching adeguata. Questo è stato possibile farlo, come spiegato dettagliatamente nel caso applicativo, individuando alcuni driver principali per la valutazione dei due flussi di sviluppo nel contesto dell'Amministrazione e assegnando a ciascun driver un punteggio in corrispondenza di ciascuna strategia di branching. È stato possibile assegnare tali punteggi in base all'esperienza dei membri nel team con progetti simili e con domande mirate e interviste effettuate settimanalmente con il cliente;

- **Riprogettazione architetturale dell'infrastruttura abilitante.** Per quanto concerne questa attività, ho partecipato a diversi meeting che il Team Architetture ha effettuato in stretta collaborazione con il fornitore GitLab. Il prodotto di tali incontri sono state le viste architetturali riguardanti i componenti dei nodi GitLab, le connessioni e l'allocazione. Infatti, dopo aver analizzato e dettagliato i diversi requisiti, sulla base di questi ultimi, è stata scelta l'architettura più adatta a rappresentare il contesto nel quale ci troviamo e a soddisfare i requisiti dell'utente;

- **Swot Analysis sulla creazione dell'infrastruttura IT.** Ho avuto l'opportunità di effettuare una swot analysis per quanto concerne la creazione e manutenzione dell'infrastruttura IT. Le alternative erano realizzare GitLab on-premises oppure installarlo in Cloud. Per ciascuna delle due opzioni ho analizzato punti di forza, punti di debolezza, minacce e opportunità rappresentandole prima testualmente e poi graficamente. Ciò è stato permesso grazie a ricerche mirate sul web e in base alle esigenze manifestate dal cliente.

In definitiva, l'adozione di DevOps può consentire alla Pubblica Amministrazione di offrire servizi più efficienti, trasparenti e orientati al cittadino, contribuendo così a una governance più efficace e all'avanzamento della società digitale.

Sebbene l'adozione della metodologia DevOps nella Pubblica Amministrazione, come evidenziano i risultati raggiunti sia molto promettente, è necessario identificare le sfide ed esplorare le potenziali aree di miglioramento: proprio per le caratteristiche del settore, l'applicazione di DevOps e pratiche agili in questo contesto presenta molte sfide ancora aperte. Questo elaborato ha cercato di contribuire in questa direzione, esaminando le applicazioni, i benefici, le sfide e le considerazioni associate al DevOps. Nel prossimo futuro è evidentemente necessario continuare a monitorare, documentare e interpretare le rapide evoluzioni che si presentano anche considerando il crescente interesse sull'argomento. In conclusione, il DevOps ha dimostrato come un approccio innovativo e collaborativo possa trasformare radicalmente i processi interni e la qualità dei servizi offerti. Questo percorso di trasformazione rappresenta un modello di riferimento per future iniziative di modernizzazione e digitalizzazione nella pubblica amministrazione, ponendo le basi per un'amministrazione più efficiente, sicura e orientata al cittadino, nonché all'avanzamento della società digitale.

TABELLE ACRONIMI E DEFINIZIONI

Per una maggiore chiarezza e per offrire al lettore una più ampia comprensione, in questo paragrafo viene riportato, in forma tabellare, un elenco degli acronimi riportati all'interno di questo elaborato, con la rispettiva definizione (prima tabella) e dei termini tecnici che necessitano di una descrizione (seconda tabella).

Acronimo	***Definizione***
AAD	Azure Active Directory
API	Application Programming Interface
BPMN	Business Process Modeling Notation
CED	Centro Elaborazione Dati
DBMS	Database Management System
DEC	Direttore dell'Esecuzione del Contratto
MAC	Manutenzione correttiva
MEV	Manutenzione evolutiva
RPC	Remote Procedure Call
RPO	Recovery Point Objective
RTO	Recovery Time Objective
RUP	Responsabile Unico di Progetto
TDD	Test-driven Development
VPN	Virtual Private Network.

Tabella 10 - Tabella degli acronimi

Termine	**Descrizione**
Application Programming Interface (API)	Insieme di definizioni e protocolli per la creazione e integrazione di applicazioni software
Back-up	Replicazione su un supporto di memorizzazione di materiale informativo, archiviato nella memoria di massa dei computer, al fine di prevenire la perdita definitiva dei dati in caso di eventi malevoli accidentali o intenzionali
Bug	Anomalia che porta al malfunzionamento di un software, per esempio producendo un risultato inatteso o errato
Build	Preparazione, costruzione, produzione del pacchetto da distribuire, cioè il software eseguibile
Business Process Modeling Notation (BPMN)	Metodo di notazione per delineare un processo aziendale
Cloud Computing	Tecnologia che consente di usufruire, tramite server remoto, di risorse software e hardware, il cui utilizzo è offerto come servizio da un provider
Commit o Git Commit	Comando Git per salvare i cambiamenti del codice sorgente con allegato commento esplicativo
Database Management System (DBMS)	Sistema di gestione della base di dati. Sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione di una o più basi di dati in modo corretto ed efficiente
Data center	Centro di elaborazione dati. Funzione all'interno di un'organizzazione che coordina e mantiene le apparecchiature ed i servizi di gestione delle risorse informatiche a servizio di uno o più fruitori
Focus group	Intervista rivolta ad un gruppo omogeneo di persone per approfondire un tema o particolari aspetti di un argomento
Front-end	Interfaccia grafica utente con cui gli utenti possono interagire direttamente, come menù di navigazione, elementi di design, pulsanti, immagini e grafici
Hotfix	Sviluppo software legato alla risoluzione di bug
Message broker	Modulo software che permette l'integrazione asincrona tramite lo scambio di messaggi
Middleware	Software che funge da intermediario tra applicazioni, strumenti e database per fornire agli utenti servizi unificati
Peer review	Valutazione critica che un lavoro o una pubblicazione riceve da parte di specialisti aventi competenze analoghe a quelle di chi ha prodotto l'opera
Pipeline	Servizio o insieme di azioni che elaborano i dati in sequenza. Ciò significa che i risultati o l'output di un segmento del sistema diventano input per il successivo

Product owner	Leader o autorità che guidano l'avanzamento del prodotto e gestiscono il team verso il successo
Provisioning	Processo di creazione e configurazione di un'infrastruttura IT, che comprende le procedure necessarie per gestire l'accesso di utenti e sistemi a varie risorse
Pull Request	Meccanismo di accettazione e approvazione per le evolutive e correttive eseguite
Push Request	Meccanismo che generalmente consente ad uno sviluppatore di notificare l'avvenuto completamento di una funzionalità
Repository	Archivio digitale centralizzato che gli sviluppatori utilizzano per apportare e gestire le modifiche al codice sorgente di un'applicazione
Rollback	Operazione che permette di riportare la base di dati a una versione o stato precedente
RPO	Recovery Point Objective, ovvero tempo massimo previsto per il ripristino di un sistema aziendale, così da renderlo di nuovo utilizzabile dal cliente finale
RTO	Recovery Time Objective, ovvero tempo massimo che intercorre tra la produzione di un dato e la sua messa in sicurezza (ad esempio attraverso backup)
Sandbox	Ambiente sterile costituito da una macchina virtuale isolata dal resto del computer, in cui è possibile lanciare software potenzialmente dannosi, senza rischiare di danneggiare il computer o la rete
Script	Tipo particolare di programma, scritto in una particolare classe di linguaggi di programmazione
Service design	Approccio alla progettazione che si occupa di definire come si svolge la relazione tra un utente e un'organizzazione, generando un'esperienza di qualità per entrambe le parti coinvolte e agevolando il raggiungimento del risultato desiderato
Smoke test	Controllo preliminare del software dopo una complicazione e prima di un rilascio. Aiuta gli sviluppatori di software a capire se il programma funziona come dovrebbe e se è pronto a passare alla fase successiva di test
Techincal review	Processo sistematico e strutturato di valutazione della qualità di determinati requisiti di un prodotto o codice software da parte di un team di esperti
Test-driven Development (TDD)	Sviluppo guidato dai test. Metodologia che scrive i test prima di sviluppare il codice in produzione

Time to market	Tempo necessario per lo svolgimento del processo che va dall'ideazione e sviluppo di un prodotto o servizio fino ad arrivare alla sua commercializzazione
Trunk Flow	Approccio di sviluppo con Git che prevede l'utilizzo di un'unica linea di sviluppo principale (detta "trunk" o "mainline") su cui vengono integrati tutti i cambiamenti, che costituisce il punto centrale di sviluppo e distribuzione del software.

Tabella 11 - Tabella dei termini

INDICE FIGURE

INDICE TABELLE

BIBLIOGRAFIA E SITOGRAFIA

Figure:

- [1] L.Ardito, F.Corno, M.Torchiano, “Sistemi Informativi Aziendali”, 2022 – v 0.9.1
- [2] (s.d.) Tratto da http://wildiritto.pbworks.com/w/page/130739487/LA%20PUBBLICA%20AMMINISTRA ZIONE-
- [3] (s.d.) Tratto da https://www.bucap.it/news/trasformazione-digitale-pubblica-amministrazione.htm
- [4] (s.d.) Tratto da https://tc1019softwareengineering.wordpress.com/2016/10/03/waterfall-model/
- [5] (s.d.) Tratto da https://monade.io/blog/agile/scrum-come-funziona-la-metodologia-agile-piu-diffusa/
- [6] (s.d.) Tratto da https://learn.microsoft.com/it-it/devops/what-is-devops
- [7] (s.d.) Tratto da https://herequality.com/theory-of-constraints-toc-is-spirituality-in-business/
- [8] (s.d.) Tratto da https://www.aditinet.it/tecnologia/devops-e-gestione-del-bilanciamento-applicativo-allinterno-della-pipeline-di-ci-cd/
- [9] (s.d.) Tratto da https://blog.hubspot.com/website/continuous-integration-tools

Testo:

- L.Ardito, F.Corno, M.Torchiano, “Sistemi Informativi Aziendali”, 2022 – v 0.9.1
- K.Beck, M.Beedle, A.Van Bennekum, J.Grenning, J.Highsmith, A.Hunt, R.C. Martin, S.Mellor, K. Schwaber, W.Cunningham, M. Fowler, J.Kern, B.Marick, J.Sutherland, D.Thomas, “Manifesto Agile”, 2001
- (s.d.) Tratto da https://agilemanifesto.org/iso/it/manifesto.html
- (s.d.) Tratto da https://universeit.blog/sistema-informativo-aziendale/
- (s.d.) Tratto da https://vitolavecchia.altervista.org/componenti-di-un-sistema-informativo-per-la-gestione-delle-informazioni/

- (s.d.) Tratto da https://www.diritto.it/la-pubblica-amministrazione-definizione-principi-struttura-e-profili-di-criticita/
- (s.d.) Tratto da https://www.agendadigitale.eu/cittadinanza-digitale/egovernment-in-italia-quadro-normativo-e-tecniche-di-valutazione-degli-interventi/
- (s.d.) Tratto da http://qualitapa.gov.it/sitoarcheologico/www.urp.it/sito-storico/www.urp.it/allegati/6_e-government.pdf
- (s.d.) Tratto da https://innovazione.gov.it/italia-digitale-2026/il-piano/digitalizzazione-della-pa/
- (s.d.) Tratto da https://www.agid.gov.it/it/agenzia/piano-triennale
- (s.d.) Tratto da https://www.bucap.it/news/trasformazione-digitale-pubblica-amministrazione.htm#:~:text=Il%20Modello%20strategico%20di%20evoluzione%20del%20sistema%20informativo%20della%20pubblica,gli%20obiettivi%20del%20Piano%20triennale
- (s.d.) Tratto da https://www.agid.gov.it/it/agenzia/attuazione-misure-pnrr
- (s.d.) Tratto da https://temi.camera.it/leg18/temi/riordino_ministeri.html#:~:text=Il%20decreto%2Dlegge%2022%2F2021%20istituisce%20il%20Ministero%20della%20transizione,del%20Ministero%20dello%20sviluppo%20economico.
- (s.d.) Tratto da https://asana.com/it/resources/waterfall-project-management-methodology
- (s.d.) Tratto da https://www.atlassian.com/it/agile
- (s.d.) Tratto da https://www.qrpinternational.it/blog/agile/agile-vs-waterfall/
- (s.d.) Tratto da https://it.smartsheet.com/comprehensive-guide-values-principles-agile-manifesto
- (s.d.) Tratto da https://it.smartsheet.com/understanding-agile-software-development-lifecycle-and-process-workflow
- (s.d.) Tratto da https://www.humanwareonline.com/project-management/center/devops-approccio-agile/
- (s.d.) Tratto da DEVOPS Che cos'è DevOps? - Azure DevOps | Microsoft Learn

- (s.d.) Tratto da https://learn.microsoft.com/en-us/training/modules/introduce-foundation-pillars-devops/6-summary
- (s.d.) Tratto da https://www.atlassian.com/it/git/tutorials/source-code-management
- (s.d.) Tratto da https://vitolavecchia.altervista.org/cose-importanza-e-fasi-della-continuous-integration-nello-sviluppo-software/
- (s.d.) Tratto da Continuous delivery: introduzione a CI/CD | Atlassian Continuous Delivery
- (s.d.) Tratto da https://www.investopedia.com/terms/c/continuous-operations.asp
- (s.d.) Tratto da https://www.realdocumentsolution.it/blog/soluzione-on-premise-o-cloud-quale-scegliere/#:~:text=La%20differenza%20principale%20tra%20le,ospitato%20sul%20server%20del%20fornitore
- (s.d.) Tratto da https://www.zerounoweb.it/techtarget/searchdatacenter/cose-il-devsecops-e-quali-sono-i-4-pilastri-che-lo-caratterizzano/
- (s.d.) Tratto da https://www.cybersecurity360.it/cultura-cyber/security-by-design-strumenti-e-metodologie-per-lo-sviluppo-sicuro-del-software/
- (s.d.) Tratto da https://www.bigdata4innovation.it/big-data/data-base-management-system/
- (s.d.) Tratto da https://www.talend.com/it/resources/what-is-middleware/
- (s.d.) Tratto da https://modi2018.readthedocs.io/it/latest/doc/doc_02_cap_05.html
- (s.d.) Tratto da https://miro.com/it/diagramma/cosa-e-bpmn/
- (s.d.) Tratto da https://www.google.com/search?q=cloud+computing+definizione&rlz=1C1GCEA_enIT1074IT1074&oq=cloud+computing+definizione&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDUyMzZqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://www.google.com/search?q=front-end+definizione+informatica&rlz=1C1GCEA_enIT1074IT1074&oq=front-end+definizione+informatica&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDU1NzRqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8

- (s.d.) Tratto da https://it.wikipedia.org/wiki/Centro_elaborazione_dati
- (s.d.) Tratto da https://aws.amazon.com/it/what-is/repo/#:~:text=Un%20repository%2C%20o%20repo%2C%20%C3%A8,durante%20lo%20sviluppo%20del%20software.
- (s.d.) Tratto da https://www.redhat.com/it/topics/api/what-are-application-programming-interfaces#:~:text=Il%20termine%20API%2C%20acronimo%20di,l'integrazione%20di%20applicazioni%20software.
- (s.d.) Tratto da https://www.proofpoint.com/it/threat-reference/sandbox#:~:text=Con%20il%20termine%20sandbox%2C%20nel,il%20computer%20o%20la%20rete.
- (s.d.) Tratto da https://www.google.com/search?q=service+design+definizione+informatica&rlz=1C1GCEA_enIT1074IT1074&oq=service+design+definizione+informatica&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDY0NTRqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://it.wikipedia.org/wiki/Bug#:~:text=Il%20bug%20(pronuncia%20inglese%20%2Fb%CA%8Cg,del%20codice%20sorgente%20di%20un
- (s.d.) Tratto da https://www.google.com/search?q=time+to+market+definiziuone&rlz=1C1GCEA_enIT1074IT1074&oq=time+to+market+definiziuone&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDM3MjBqMGo0qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://www.google.com/search?q=product+owner&rlz=1C1GCEA_enIT1074IT1074&oq=product+owner&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDI2NDhqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://www.google.com/search?q=provisioning+informatica+definizione&rlz=1C1

GCEA_enIT1074IT1074&oq=provisioning+informatica+definizione&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDU3OThqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8

- (s.d.) Tratto da https://www.google.com/search?q=test+driven+development&rlz=1C1GCEA_enIT1074IT1074&oq=test+driven+development&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDQyNjZqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://www.google.com/search?q=build+significato+informatica&rlz=1C1GCEA_enIT1074IT1074&oq=build+significato+informatica&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDU2OTZqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://it.wikipedia.org/wiki/Script
- (s.d.) Tratto da https://www.google.com/search?q=pipeline+informatica+definizione&rlz=1C1GCEA_enIT1074IT1074&oq=pipeline+informatica+definizione&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDY4MDJqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://www.google.com/search?q=smoke+test+informatica+definizione&rlz=1C1GCEA_enIT1074IT1074&oq=smoke+test+informatica+definizione&gs_lcrp=EgZjaHJvbWUyBggAEEUYOdIBCDc2MjVqMGo5qAIAsAIA&sourceid=chrome&ie=UTF-8
- (s.d.) Tratto da https://it.wikipedia.org/wiki/Rollback#:~:text=Il%20rollback%2C%20in%20informatica%2C%20%C3%A8,la%20precedente%20%C3%A8%20detta%20revert
- (s.d.) Tratto da https://www.uniba.it/it/docenti/toma-ernesto/attivita-didattica/focus-2023-24-ascc.pdf
- (s.d.) Tratto da https://it.wikipedia.org/wiki/Revisione_paritaria
- (s.d.) Tratto da https://fastercapital.com/it/contenuto/Revisione-tecnica--come-eseguire-una-revisione-tecnica-per-il-codice-software.html.