Alma Mater Studiorum · Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

DevOps e Continuous Integration per Sistemi IoT e Mobile - Un Caso di Studio basato su Android

Relatore:
Chiar.mo Prof.
Alessandro Ricci

Co-Relatore:
Dott. Ing.
Angelo Croatti

Presentata da:
Lisa Cattalani

Anno Accademico 2018-2019

A tutti coloro che mi hanno sostenuto durante questo cammino

Introduzione

Negli anni sia la tecnologia che le tecniche applicate per la produzione di software sono cambiate molto, si è passati da un approccio waterfall, per poi passare ad uno più innovativo, Agile, ed arrivare poi ad un approccio DevOps. In questa tesi si discuterà di come DevOps viene applicato ai vari settori, passando dal più comune It,all'Iot, al mobile per poi passare più nello specifico di Android. Nel primo capitolo si parlerà di come è nato DevOps, quali erano i precedenti approcci, di come sono evoluti in base al mercato, e di come può essere applicato DevOps nell'IT. Nel secondo capitolo si andrà più nello specifico di cos'è DevOps, quali fasi, metodologie e strumenti sono utilizzati in questo approccio, quali nuovi termini sono nati e cosa comporta questo approccio nell'immediato futuro. Nel terzo capitolo si parlerà invece dell'evoluzione delle tecnologie, dell'IoT, del mobile e di come applicare DevOps a queste nuove tecnologie. Nel quarto capitolo si andrà invece nello specifico di come applicare un approccio DevOps e la continuous integration ad un progetto Android, CERERE; nello specifico verrà considerato il server CI/CD Bitrise. Alcuni paragrafi iniziali sono state deliberatamente presi, adattati e citati, dal libro [1].

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

DevOps quadro generale

Il primo capitolo fornirà una breve definizione e introduzione a DevOps, partendo da quelli che sono stati i precursori di questo approccio e di come sono cambiati nel tempo gli obbiettivi e i requisiti che un software deve soddisfare.

1.1 DevOps definizione

Cosa si intende per DevOps? Vi sono diverse definizioni di DevOps, una comune e molto sintetica dice che:

"per DevOps si intende un insieme di pratiche intese a ridurre il tempo che intercorre tra il commit di una modifica in un sistema e l'avvenuta modifica in produzione, garantendo al contempo un'alta qualità.[1]"

La qualità della modifica distribuita in un sistema (di solito sotto forma di codice) è importante. Qualità significa adeguatezza all'uso da parte di varie parti interessate, inclusi utenti finali, sviluppatori o amministratori di sistema. Include anche la disponibilità, la sicurezza, l'affidabilità e altri "Requisiti non funzionali(Ilities)". Ci sono diversi metodi per garantire la qualità, tra questi abbiamo:

- avere una varietà di "*test cases*" automatizzati che devono dare esito positivo prima di portare le modifiche in produzione.

- Testare le modifiche in produzione con un numero limitato di utenti prima di renderle effettive a tutte le parti interessate.
- Monitorare attentamente il codice appena distribuito per un determinato periodo di tempo.

Non si specificherà come sia garantita la qualità, ma verrà richiesto che il codice prodotto sia di alta qualità. Questo implica inoltre che il delivery debba essere di alta qualità e che il meccanismo di delivery debba essere affidabile e ripetibile. Se il meccanismo di delivery fallisce regolarmente, il tempo richiesto aumenta. Se si verificano errori nel modo in cui avviene il delivery della modifica, la qualità del sistema distribuito ne risente, ad esempio, con una minore disponibilità ed affidabilità. Si identificheranno due periodi di tempo come importanti:

- Il primo è il momento in cui uno sviluppatore esegue il commit del codice appena sviluppato. Questo segna la fine dello sviluppo e l'inizio del processo di delivery.
- Il secondo è il delivery di quel codice in produzione.

Infatti vi è un periodo di tempo antecedente alla distribuzione del codice in produzione, dove quest'ultimo, viene testato attraverso live test e viene attentamente monitorato per trovare potenziali problemi. Se il codice supera i live test e il monitoraggio, viene considerato come *una parte del sistema di produzione*. Se una pratica ha lo scopo di ridurre il tempo che intercorre tra un commit di uno sviluppatore e la distribuzione in produzione, è una pratica DevOps che implica metodi, strumenti o forme di coordinamento Agili. Altre definizioni sottolineano gli strumenti utilizzati, senza menzionare l'obiettivo delle pratiche DevOps, il tempo impiegato o la qualità. In generale la comunità tecnologica utilizza una varietà di termini per descrivere l'essenza di DevOps. È una cultura, un movimento, una filosofia, un insieme di pratiche e l'atto di utilizzare strumenti (software) per automatizzare e migliorare i metodi di gestione di sistemi complessi. Indipendentemente da come lo si caratterizzano, DevOps è meglio definito dal **perché**: DevOps esiste perché lo sviluppo di software è un Asset strategico(entità materiale o immateriale suscettibile di valutazione economica per un certo soggetto, sono tutto ciò che all'interno di un sistema aziendale è monetizzabile e quindi può generare valore) per le organizzazioni

e le imprese di tutte le dimensioni nella moderna economia globale. Pertanto, i professionisti di Op cercano di creare, gestire sistemi complessi ricercando un miglioramento continuo e con una consapevolezza di dover soddisfare le esigenze di un modello software nel minor tempo possibile. La cultura DevOps è rafforzata dalle pratiche che prende in prestito dai principi Agile e Lean, con un'ulteriore attenzione al servizio e alla qualità. Progettando, costruendo, testando, implementando, gestendo applicazioni e sistemi in modo più rapido e affidabile, i professionisti di DevOps cercano di creare valore per i clienti (un vantaggio competitivo redditizio) e promuovere un flusso di lavoro gestibile rispetto al prodotto. Infine, gli obiettivi specificati nella definizione non limitano l'ambito delle pratiche DevOps ai test e alla distribuzione. Al fine di raggiungere tali obiettivi, è importante includere un modello operativo nella raccolta dei requisiti, ovvero molto prima dello sviluppo. Analogamente, la definizione non significa che le pratiche DevOps terminano con il rilascio in produzione; l'obiettivo è garantire l'alta qualità del sistema distribuito durante tutto il suo ciclo di vita. Pertanto, devono essere incluse anche le pratiche di monitoraggio che aiutano a raggiungere tali obiettivi.

1.2 I precursori dell'approccio DevOps: SDLC

Prima di parlare di DevOps e della sua nascita, si parlerà di cosa sia il Software development process, conosciuto sopratutto come **Software Development Life Cycle** (**SDLC**). "*SDLC è un processo continuo, che inizia nel momento in cui viene presa una decisione per iniziare un progetto, e finisce nel momento in cui il progetto è stato completamente sfruttato. Non c'è un singolo modello SDLC. I modelli sono divisi in vari gruppi, ognuno con le sue caratteristiche e debolezze.*"[2] Questi modelli hanno di solito 5-6 fasi principali che sono:

1. **Pianificazione e Analisi dei requisiti**(talvolta separati): la prima fase è quella più importante del ciclo. Viene eseguita dai membri senior del team prendendo in input i *desiderata* del cliente. Queste informazioni vengono utilizzate per pianificare l'approccio del progetto e per condurre studi di fattibilità del prodotto. Viene pianificata anche la qualità e vengono identificati i rischi associati al progetto. Il

risultato dello studio di fattibilità è definire i vari approcci tecnici che possono essere seguiti per attuare il progetto con successo e con il minor numero di rischi.

2. **Design**: in questa fase viene progettata l'architettura. Tutte le differenti questioni tecniche che possono venire fuori vengono discusse da tutte le parti interessate, anche con il cliente.Vengono anche definite le tecnologie utilizzate, il carico di lavoro, i limiti di tempi e il budget che vengono calcolati ed adottati in funzione del progetto.

3. **Implementazione**: I developer seguono le linee guida date dall'organizzazione, programmando e producendo il software richiesto attraverso appositi strumenti. Questa fase al suo interno include 4 fasi: la scelta dell'algoritmo di sviluppo, la scrittura, la compilazione e il debug del codice prodotto.Il linguaggio viene scelto in base alle esigenze dell'azienda e del software che sono emerse dalle fasi precedenti.

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

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

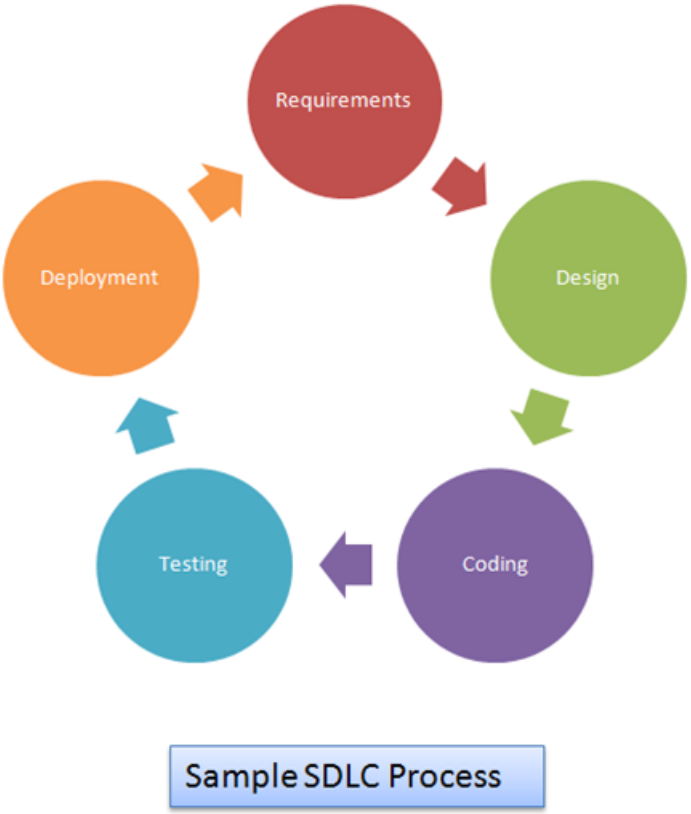

Figura 1.1: SDLC lifecycle. [43]

Da questi concetti di base di SDLC sono nati diversi principali modelli, i più comuni sono:

- il modello tradizionale **WaterFall**(comunemente detto *Modello a cascata*) degli anni 70.
- Modello **Iterative and incremental** (Iterativo ed Incrementale) sviluppato nel 1985.
- **Modello a spirale** sviluppato nel 1988.
- **V-Model**, detto anche V-shape Model, in quanto le varie fasi sono disposte a v, sviluppato negli anni 90.

- **Metodologia Agile**: Il Manifesto è del 2001; al suo interno troviamo anche l'XP (Extreme Programming), TDD (Test driven development), Scrum e Lean software development.

1.2.1 WaterFall

Il modello **Waterfall** è stato una delle prime pratiche applicate al mondo IT ed è stato proposto per la prima volta negli anni 70 da *Winston W. Royce*. Questo modello è diventato popolare perchè fornisce una linea guida pratica per sviluppare soluzioni software. Il suo nome deriva da specifiche strutturali. Al termine di ogni fase si passa alla successiva, e le attività possono essere divise in base alle fasi. L'ouput di una fase diventa l'input della fase successiva, si ha però la possibilità di rivisitare le fasi nel ciclo successivo.[3] L'utilizzo di un tale modello:

- incoraggia a specificare ciò che il sistema dovrebbe fare (cioè definire i requisiti) prima di costruire il sistema (cioè prima della progettazione);
- incoraggia a pianificare come devono interagire i componenti prima di costruirli (ovvero la codifica);
- consente ai Project manager di monitorare i progressi in modo più accurato e scoprire presto eventuali ritardi;
- richiede che il processo di sviluppo generi una serie di documenti che possano essere successivamente utilizzati per testare e mantenere il sistema;
- riduce i costi di sviluppo e manutenzione per via di tutti i motivi precedentemente elencati,
- aiuta l'organizzazione che svilupperà il sistema ad essere più strutturata e gestibile.[4]

Viene definito modello a cascata proprio perché ogni cascata rappresenta un incremento come illustrato nella figura 1.2.

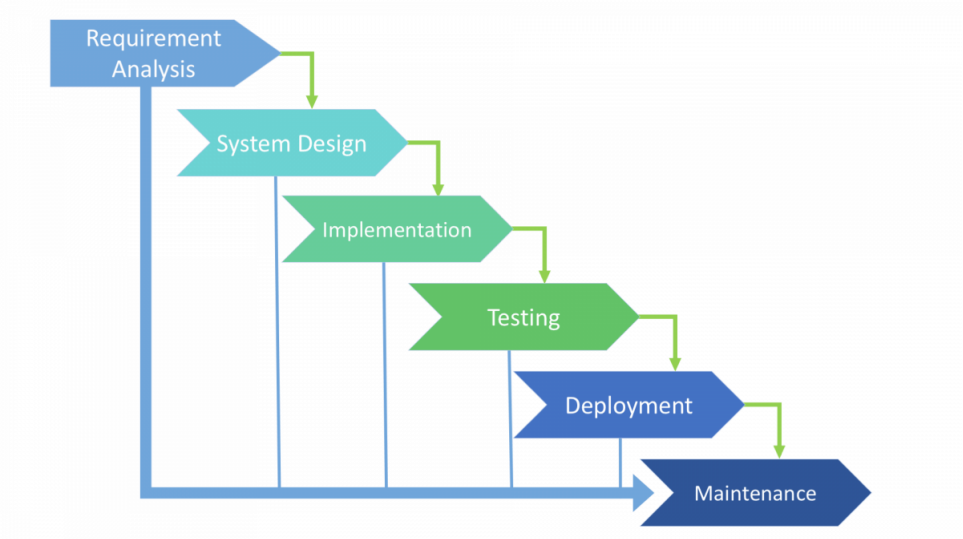

Figura 1.2: SDLC Waterfall. [45]

È indicato come approccio quando:

- la qualità è più importante del costo o della schedulazione temporale.
- I requisiti sono ben conosciuti, chiari e fissi.
- È necessaria una nuova versione di un prodotto esistente.
- È necessario eseguire il porting di un prodotto esistente in una nuova piattaforma.

Brevemente si elencheranno gli svantaggi e i vantaggi di questo approccio, nella tabella 1.1

Vantaggi	**Svantaggi**
Facile da capire e da utilizzare.	È molto difficile tornare indietro ad una qualsiasi fase dopo che questa è stata completata.
Utile per progetti piccoli,di routine e ripetiti dove i requisiti sono chiaramente definiti.	Inappropriato per progetti a lungo termine.
Ogni fase ha risultati specifici.	Costoso e richiede più tempo, in quanto è difficile stimare le risorse e i costi se non si è svolta la prima fase di analisi.
La verifica fase per fase garantisce l'individuazione precoce degli errori/incomprensioni.	Non è flessibile e quindi gli aggiustamenti di scope sono molto difficili e costosi.

Tabella 1.1: Modello Waterfall vantaggi e svantaggi

A partire dagli anni ottanta il modello è stato soggetto a profonde critiche e revisioni, soprattutto dovute all'evoluzione del software stesso e dei linguaggi di programmazione.

1.2.2 Modello Iterativo ed Incrementale

Questo modello mette insieme i principi del modello iterativo con quello incrementale sopperendo ai problemi del modello a cascata, viene comunemente chiamato "**Iterative and Incremental**" abbreviato in **IID**. La prima fase è quella di planning e termina con la fase di rilascio, tra le due fasi troviamo dei cicli ripetuti di iterazioni come mostrato nella figura 1.3.

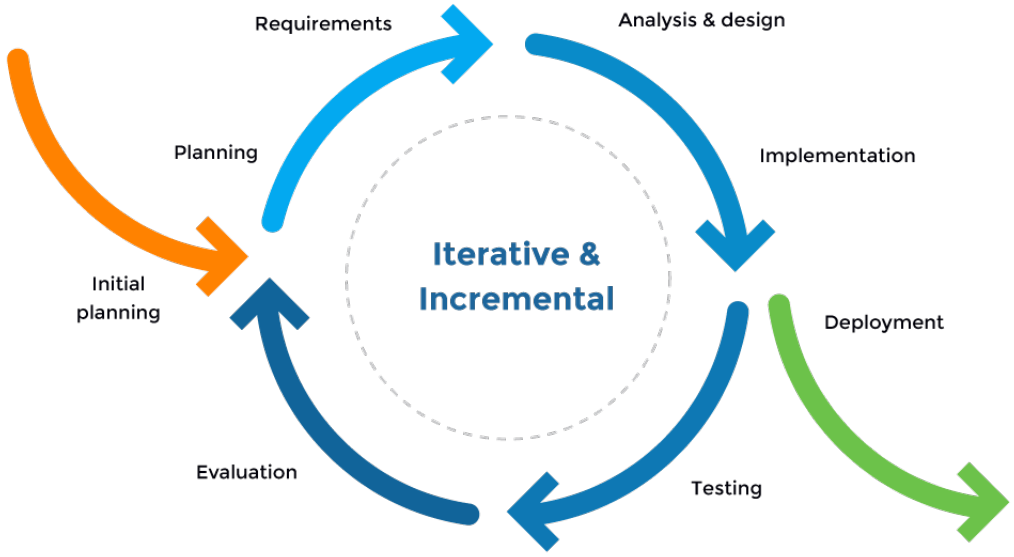

Figura 1.3: SDLC Iterative And Incremental. [46]

L'idea alla base di questo metodo è quella di sviluppare un sistema attraverso cicli ripetuti (iterativi) e sviluppare porzioni ogni volta più piccole (incrementali) consentendo agli sviluppatori di trarre vantaggio da ciò che è stato appreso durante lo sviluppo di parti o versioni precedenti del sistema. Può essere considerato come un mini Waterfall o mini V-Model.

È utile usare questo approccio:

- per progetti con rischio basso-medio.
- Per progetti che hanno delle schedule lunghe.
- Per progetti che fanno uso di nuove tecnologie, permettendo all'utente di suddividere il sistema in piccole fasi incrementali piuttosto che andare direttamente a creare un nuovo prodotto.
- Quando è altamente rischioso sviluppare interamente il sistema in una sola volta.[5]

Nella tabella 1.2 alcuni dei vantaggi e degli svantaggi apportati da questo modello.

Vantaggi	**Svantaggi**
Il business value viene prodotto più velocemente nella ciclo di vita di sviluppo.	Richiede una documentazione esaustiva.
Sono accettate modifiche durante le fasi incrementali.	Gli incrementi sono definiti e dipendenti in base alle funzioni e alle funzionalità.
Vi è un maggiore focus sul customer value rispetto all'approccio waterfall.	Richiede però un maggiore coinvolgimento da parte del cliente rispetto all'approccio Waterfall.
I problemi vengono rilevati più facilmente e le modifiche sono applicate più velocemente.	L'integrazione tra le varie interazioni può rappresentare un problema se non viene considerata durante la fase di sviluppo o durante la fase di planning.

Tabella 1.2: Modello IID vantaggi e svantaggi

1.2.3 Modello a Spirale

Proposta da Barry Boehm nel 1988, nel documento "**A Spiral Model of Software Development and Enhancement**"; rispetto ai precedenti approcci è più orientato al "*risc-driven*" che al "*document-driven*" o al "*code-driven*". Incorpora molti degli aspetti dei precedenti modelli e ne colma anche alcune lacune.[6] Il modello a spirale può ospitare al suo interno la maggior parte dei precedenti modelli come casi particolari e fornisce inoltre indicazioni su quale combinazione di modelli si adatta meglio ad una data situazione/tipologia di software. La dimensione radiale nella Figura [47] rappresenta il costo cumulativo sostenuto per eseguire ogni fase; la dimensione angolare rappresenta invece i progressi compiuti nel completare ciascun ciclo della spirale. Il modello riflette il concetto che ogni ciclo comporta una progressione che esegue la stessa sequenza di passaggi, per ogni porzione del prodotto e per ciascuno dei suoi livelli di elaborazione, da un concetto generale di documento operativo fino alla codifica di ogni singolo programma.

Ogni ciclo della spirale inizia con l'identificazione degli obiettivi della porzione di prodotto in elaborazione (prestazioni, funzionalità, capacità di adattamento al cambiamento, ecc.); i mezzi alternativi per implementare questa parte del prodotto (design A, design B, riutilizzo, acquisto, ecc.); e i vincoli imposti dalle alternative (costo, pianificazione, interfaccia, ecc.). Il passo successivo è valutare le alternative relative agli obiettivi e ai vincoli.Spesso, questo processo identificherà le aree di incertezza che sono fonti significative di rischio del progetto.In tal caso, il passaggio successivo dovrebbe comportare la formulazione di una strategia economica per la risoluzione delle fonti di rischio. Ciò può coinvolge la prototipazione, la simulazione, l'analisi comparativa, il controllo di riferimento, la gestione di questionari utente, la modellazione analitica o una combinazione di queste e altre tecniche di risoluzione dei rischi.Una volta valutati i rischi, il passaggio successivo è determinato dai rischi rimanenti relativi. Se i rischi relativi alle prestazioni o all'interfaccia utente influiscono fortemente sullo sviluppo del programma o sui rischi interni di controllo dell'interfaccia, il passaggio successivo può essere uno sviluppo evolutivo: uno sforzo minimo per specificare la natura generale del prodotto, un piano per il livello successivo di prototipazione e lo sviluppo di un prototipo più dettagliato per continuare a risolvere i principali problemi di rischio. Se questo prototipo è operativamente utile e abbastanza robusto da fungere da base a basso rischio per la

futura evoluzione del prodotto, i successivi passi guidati dal rischio sarebbero le serie in evoluzione di prototipi evolutivi che vanno verso destra come nella Figura [47].In questo caso, l'opzione di scrivere le specifiche verrebbe affrontata ma non esercitata. Pertanto, le considerazioni sui rischi possono portare a un progetto che implementa solo un sottoinsieme di tutti i potenziali passaggi del modello.D'altra parte, se i precedenti tentativi di prototipazione hanno già risolto tutti i rischi relativi alle prestazioni o all'interfaccia utente e dominano i rischi di sviluppo del programma o di controllo dell'interfaccia, il passaggio successivo segue l'approccio base a cascata (concetto di funzionamento, requisiti software, progettazione preliminare , ecc. come nella Figura [47]) ma modificato per incorporare lo sviluppo incrementale. Ogni livello di specifica del software presente nella figura è quindi seguito da una fase di convalida e dalla preparazione dei piani per il ciclo successivo. In questo caso, le opzioni per prototipare, simulare, modellare, ecc. sono indirizzati ma non esercitati, portando all'utilizzo di un diverso sottoinsieme di passaggi. Questa sotto-impostazione basata sui rischi delle fasi del modello a spirale consente al modello di adattarsi a qualsiasi combinazione che sia orientata: alle specifiche, al prototipo, alla simulazione, alla trasformazione automatica o ad un altro approccio allo sviluppo del software. Una caratteristica importante del modello a spirale, così come con nella maggior parte degli altri modelli, è che ogni ciclo è completato da una revisione che coinvolge le persone o le organizzazioni primarie interessate al prodotto.[7]

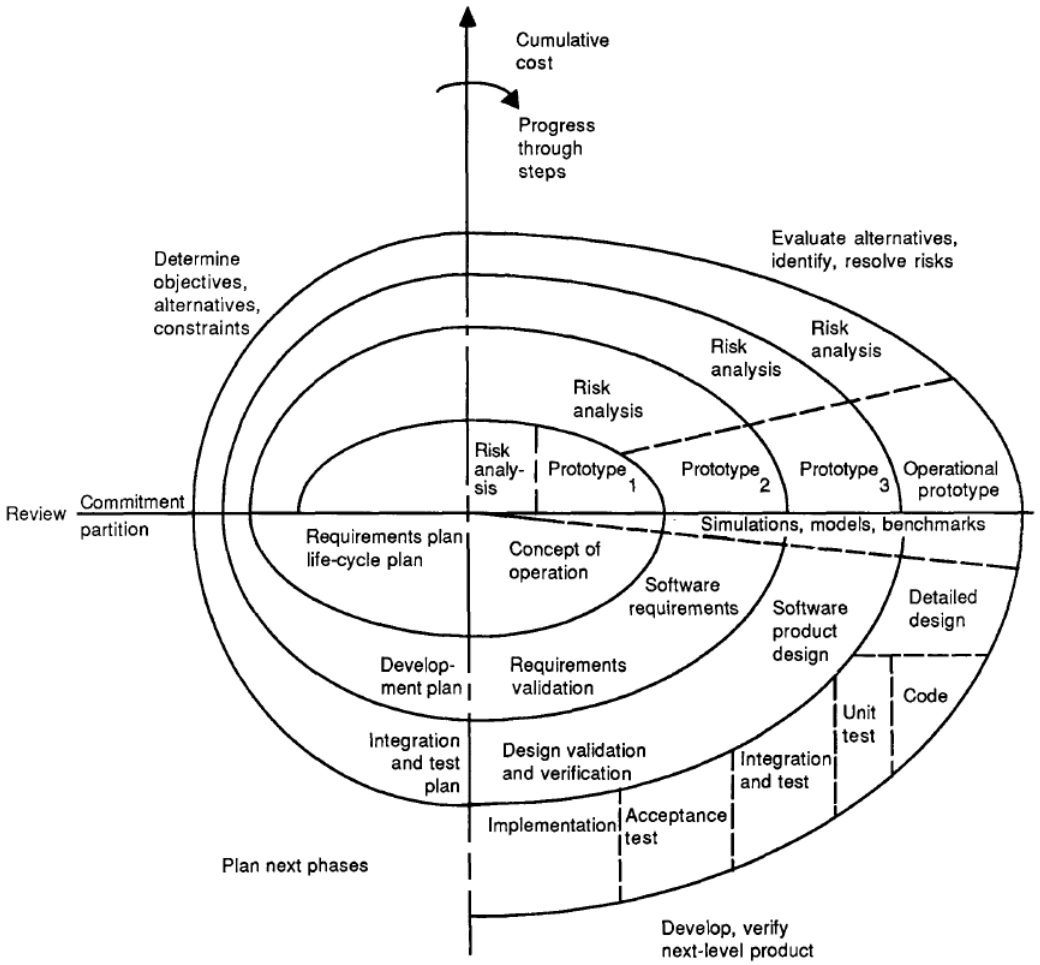

Figura 1.4: SDLC Modello Spirale. [47]

È un approccio consigliato:

- per progetti di medio-alto rischio.
- Quando la valutazione del rischio e del costo sono importanti.
- Quando ci si aspetta significativi cambiamenti al progetto.
- Quando gli utenti nono sono proprio sicuri di quello che desiderano.

Di seguito i vantaggi e gli svantaggi apportati da questo modello, riportati nella tabella 1.3 sottostante.

Vantaggi	**Svantaggi**
L'attenzione iniziale è focalizzata sulle opzioni disponibili comportando il riutilizzo di software esistente.	La capacità di valutazione del rischio viene affidata agli sviluppatori e se inesperti, possono produrre una una valutazione diversa o possono non aver individuato il rischio.
Nel ciclo di vita del software viene considerata a priori l'evoluzione,la crescita e i cambiamenti che possono essere applicati al prodotto.	È inefficace per i piccoli progetti comportando costi e tempi elevati.
Appena il primo prototipo funzionante è pronto viene eseguito di modo che gli utenti possano evidenziarne gli eventuali difetti.	Ciò comporterà un gran numero di stadi intermedi con un eccessivo aumento della documentazione prodotta.

Tabella 1.3: Modello a Spirale vantaggi e svantaggi

1.2.4 V-Model

Viene considerato come un'estensione del modello Waterfall,viene detto a V in quanto invece di procedere in modo lineare, dopo la fase iniziale di programmazione risale con una tipica forma a V e ad ogni fase ne viene associata una di test. È un modello dove l'inizio di ogni fase è strettamente collegata alla fine della fase precedente. Viene chiamato anche modello di "**Validation e Verification**" (Convalida e verifica). I test sono molto enfatizzati,più che nel modello a cascata. Le procedure di test sono sviluppate all'inizio del ciclo di vita, prima che venga eseguita qualsiasi codifica e durante ciascuna delle fasi precedenti l'implementazione. I requisiti danno inizio al ciclo proprio come nell'approccio waterfall. Prima di iniziare lo sviluppo, viene creato un piano di test del sistema. Il piano di test si concentra sul rispetto delle funzionalità specificate nella fase di raccolta dei requisiti. La fase di progettazione si concentra sull'architettura e sulla progettazione del sistema. In questa fase viene creato un piano di test d'integrazione per testare le capacità dei sistemi software di lavorare insieme. Tuttavia,la fasi di progettazione di basso livello si trovano dove vengono progettati i componenti software effettivi e anche in questa fase vengono creati degli unit test. La fase di implementazione è la fase in cui avviene tutta la codifica. Una volta completata la codifica, il percorso di esecuzione continua sul lato destro della V dove vengono poi utilizzati i piani di test sviluppati in precedenza.[6]

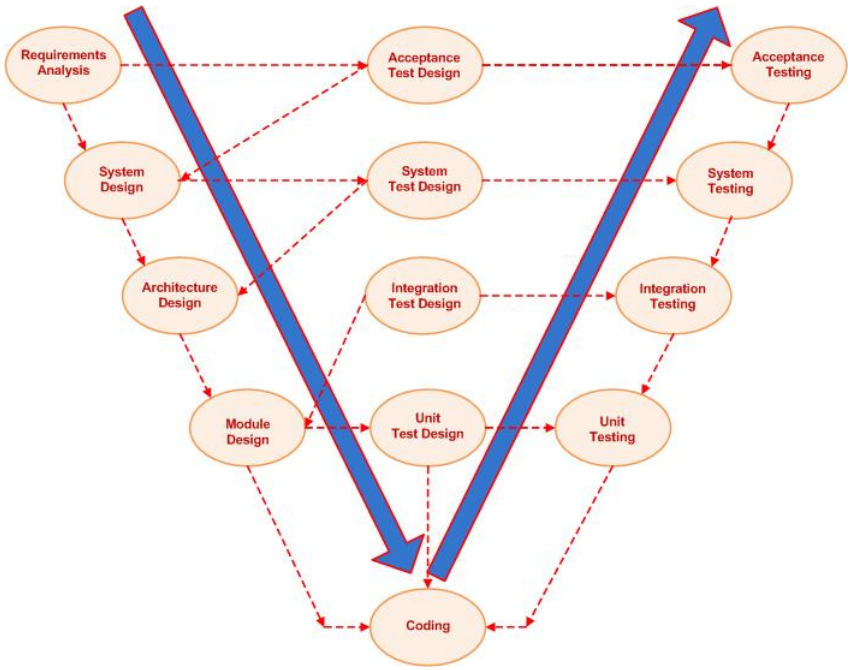

Figura 1.5: SDLC V-Model. [48]

Di seguito i vantaggi e gli svantaggi apportati da questo modello, riepilogati nella tabella 1.4.

Vantaggi	Svantaggi
Utile per piccoli progetti, dove i requisiti sono chiari e statici	Rischi relativamente elevati, essendo molto rigido e poco flessibile.
Facile da gestire grazie alla rigidità del modello. Ogni fase ha dei risultati specifici e un processo di revisione.	Pessimo modello per progetti lunghi e già avviati.
Le attività di test come la pianificazione e la progettazione dei test avvengono molto prima della codifica. Questo permette di risparmiare molto tempo, ed ha anche una maggiore probabilità di successo rispetto al modello Waterfall.	Il software viene sviluppato durante la fase di implementazione, quindi non vengono prodotti dei prototipi prima del completamento del software.

Tabella 1.4: Modello V vantaggi e svantaggi

1.2.5 Metodologia Agile

Nel modello waterfall si ritiene che i sistemi software possano essere realizzati in maniera lineare, cioè per fasi. In tale processo lineare, inizialmente vengono raccolti e analizzati tutti i requisiti ed il completamento della progettazione costituisce il passo successivo. Infine, il risultato di tale progettazione diventa direttamente il software che sarà portato in produzione, passando per una lunga fase di integrazione e test. Nonostante che questo i tradizionali approcci, sembrano validi, non sono in grado di affrontare in modo efficace i problemi legati ai costi e al tempo.

Da circa una decina di anni è, tuttavia, emersa una nuova filosofia di sviluppo che è alla base di una vasta raccolta di nuovi processi nota come **Agile Software Development**. Tali nuovi processi si concentrano maggiormente: sulle interazioni tra le persone e su un rapido sviluppo di codice piuttosto che sulla documentazione e sulla pianificazione. Già dai primi anni 90, iniziano a nascere nuove metodologie innovative quali: eXtreme Programming, Scrum, Feature Driven Development, DSDM, Crystal o Lean Software Development, che verranno poi considerate come metodologie agili, in quanto consentono di poter cambiare continuamente le specifiche durante lo sviluppo mediamente un attivo coinvolgimento con il committente. Nel 2001, 17 software developer si sono incontrati in un resort, per discutere i metodi di sviluppo, dando origine al ***Manifesto Agile***[1] (contenente 12 punti fondamentali), che tuttora rimane il riferimento principale,ufficiale e la fonte di ispirazione per qualsiasi iniziativa che intenda qualificarsi come agile. Il significato letterale della parola Agile è "*caratterizzato da rapidità, leggerezza e facilità di movimento*". Questo significa quindi che il processo di sviluppo agile consiste in una distribuzione rapida di software caratterizzata da una maggiore facilità e flessibilità di sviluppo. Utilizzando uno dei tanti punti del manifesto, si può dire che: "*La nostra massima priorità è soddisfare il cliente rilasciando software di valore, fin da subito e in maniera continua.*", ciò significa creare rapidamente dei prototipi funzionanti da poter condividere con il cliente. In generale per Metodologie agili, si intendono tutte quelle metodologie non tradizionali, cioè che non prevedono una strutturazione lineare dello sviluppo software ma che pongono maggiore attenzione sul cliente, il quale diventa l'elemento principale su cui è incentrato lo sviluppo; nella figura [49] sono riepilogate le fasi principali di questo approccio.

[1] https://agilemanifesto.org/iso/it/manifesto.html

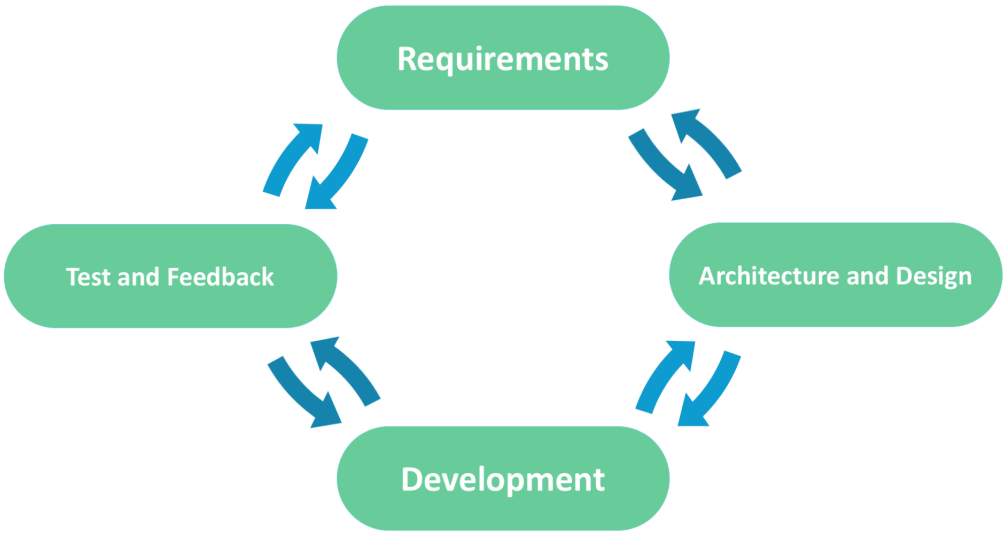

Figura 1.6: SDLC Agile Model. [49]

Di seguito alcuni dei vantaggi e degli svantaggi apportati da questo modello riepilogati nella tabella 1.5.

Vantaggi	Svantaggi
I prototipi, e i rilasci in generale, vengono realizzati più frequentemente e più velocemente.	Le continue correzioni possono far slittare notevolmente la data di consegna stimata.
Maggiore flessibilità e minore rischi.	I nuovi requisiti possono andare in conflitto con l'architettura presente.
Feedback continuo da parte del cliente finale, riducendo al minimo i rischi di progetto.	Il team deve essere molto professionale ed orientato al cliente.

Tabella 1.5: Modello Agile vantaggi e svantaggi

1.2.6 Altri approcci dopo la metodologia Agile

Dopo il manifesto agile del 2001, troviamo due altri approcci fondamentali: Domain-Driven Design del 2004 e Lean software development del 2008.

1.2.6.1 Domain Driven Design

Il **Domain Driven Design**, detto anche DDD è un approccio allo sviluppo del software che risolve problemi complessi connettendo l'implementazione ad un modello in evoluzione.È diventato famoso grazie al libro pubblicato da Eric Evans nel 2004, intitolato "*Domain-Driven Design: Tackling Complexity in the Heart of Software*". La sua premessa è:

- porre come obiettivo principale del progetto il dominio delle entità e della loro logica.

- Basare concetti complessi su un modello.

- Avviare una collaborazione creativa tra esperti tecnici e di dominio per restringere in modo iterativo il problema ed arrivare direttamente al suo nucleo concettuale.

La premessa è semplice, ma è difficile applicarla al mondo reale. Richiede nuove competenze,una determinata disciplina e un approccio sistematico.La progettazione guidata dal dominio non è una tecnologia o una metodologia. DDD fornisce una struttura di pratiche e terminologia per prendere decisioni di progettazione che focalizzano e accelerano i progetti software che si occupano di domini complicati.[8]
Perchè viene chiamato Domain Driven Design? Il termine **domain** indica in questo contesto e nello sviluppo in generale, "*A sphere of knowledge or activity*"(Una sfera di conoscenza e attività),quindi DDD è un'estensione dell'applicazione del dominio,ed è applicato allo sviluppo del software.[9]
Di seguito alcuni dei vantaggi e degli svantaggi apportati da questo modello presenti nella tabella 1.6.

Vantaggi	**Svantaggi**
Facile comunicazione	Adatto a progetti altamente tecnici.
Maggiore flessibilità	Incoraggia le pratiche iterative.
Enfatizza il dominio sull'interfaccia.	Richiede un team esperto del dominio.

Tabella 1.6: Domain Driven Design vantaggi e svantaggi

1.2.2.2 Lean StartUp

Dal metodo Lean software development deriva la **Lean StartUp**,elaborata nel 2008 dal giovane imprenditore Eric Ries che propone un nuovo approccio al lancio di prodotti e servizi innovativi. Lean StartUp è un sistema per lo sviluppo di un'azienda o di un prodotto nel modo più efficiente possibile atto a ridurre il rischio di fallimento, considerando tutte le idee di prodotto e di business come ipotesi che devono essere convalidate da una rapida sperimentazione sul mercato. L'approccio si basa sulla sperimentazione scientifica, sui rilasci iterativi dei prodotti e sul feedback dei clienti per generare un apprendimento validato.Simile ai precetti della lean management, la filosofia lean startup

cerca di eliminare le pratiche dispendiose e aumentare le pratiche di produzione di valore durante la fase di sviluppo del prodotto in modo che le startup possano avere maggiori possibilità di successo senza richiedere grandi quantità di finanziamenti esterni, piani aziendali elaborati o il raggiungimento del prodotto *perfetto*. La ricerca della perfezione nello sviluppo del prodotto, i piani aziendali a lungo termine in un mercato incerto o sconosciuto o ipotesi non convalidate sono tutti approcci evitati dai professionisti delle startup lean.[10] Di seguito alcuni dei vantaggi e degli svantaggi apportati da questo modello presenti nella tabella 1.7.

Vantaggi	**Svantaggi**
Permette di creare un prodotto basandosi sulle necessità del cliente riducendo tempi,costi e minimizzando, i rischi di mercato.	Se l'idea non funziona, prevede subito di fare pivoting(cioè di tornare indietro sulle decisioni prese), non dando il tempo di capire se il non funzionamento era un falso positivo.
Maggiore innovazione, ottenuta dai continui test eseguiti.	L'enfasi su una costante convalida, testando continuamente le proprie ipotesi e perfezionando la propria idea può facilmente portare a stanchezza ed esaurimento.
Minimum Viable Product (si intende un numero minimo di caratteristiche che deve possedere il nuovo prodotto affinché sia sostenibile).	Poca attenzione su tecnologia e architettura.
Maggiore prototipizzazione del prodotto.	Ottenere il feedback dei clienti sul prodotto incompiuto non è sempre possibile.[11]

Tabella 1.7: Lean StartUp vantaggi e svantaggi

1.2.2.3 Come l'approccio DDD ha influito su DevOps

Grazie al cambio di mentalità portato da DDD, che DevOps ha ereditato, si è compresa l'importanza di un modello nella progettazione guidata dal dominio, dove esistono tre usi di base che determinano la scelta di un modello:

1. Il modello determina la forma del design della parte centrale(*"cuore"*) del software. È il legame stretto tra il modello e l'implementazione che rende rilevante il modello e garantisce che l'analisi che vi è stata applicata si applica al prodotto finale, un programma in esecuzione. Questa associazione di modello e l'implementazione aiuta anche la manutenzione e lo sviluppo continuo perché il codice può essere interpretato in base alla comprensione del modello.

2. Il modello è il fondamento di una lingua utilizzata da tutti i membri del team. Grazie al legame tra modello e implementazione, gli sviluppatori possono parlare del programma in questa *lingua*. Loro possono comunicare con esperti di dominio senza traduzione, e, poiché la lingua si basa sul modello, le nostre capacità possono essere rivolte a perfezionare il modello stesso.

3. Il modello è il modo concordato del team di strutturare il dominio di conoscenza e distinguere gli elementi di maggior interesse. Un modello cattura il modo in cui pensiamo al dominio mentre selezioniamo i termini, suddividiamo i concetti e li mettiamo in relazione.

Grazie a questa definizione di modello, si è compreso che l'importanza del software e del suo sviluppo, non sta nelle tecnologie utilizzate, ma proprio nella comprensione di quest'ultimo, non bisogna per forza avere conoscenze approfondite del progetto che si vuole sviluppare, bensì comprendere il modello da progettare. Per farlo bisogna prima partire dal modellare il dominio in modo comprensibile sia agli altri membri del team che al committente, così da poter continuare con un *linguaggio* comune ad entrambi, senza entrare nello specifico dei tecnicismi. Se poi a ciò si aggiunge un prototipo, con direttamente un feedback dell'utente finale, si è riusciti nell'intento. Ciò ci fa capire il concetto base: non è importante il software o il linguaggio scelto, ma il modo di approcciarsi al progetto che fa la differenza. Il feedback continuo con il committente ci

garantisce uno sviluppo in linea con i requisiti del committente. Però per poter avere una discussione con il cliente che porti ad un risultato positivo, bisogna:

- Decidere un modello che sia vincolante e implementarlo nel minor tempo possibile, in modo tale da avere un prototipo (non per forza funzionante o completo), che dia inizio poi a ciò che sarà il progetto finale e che si possa utilizzare tale prototipo durante la fase iniziale di analisi con il cliente.

- Oltre ad un modello ed un prototipo, la fase successiva sarà parlare lo stesso "*linguaggio*", ciò non significa che il cliente deve capire il linguaggio di programmazione o che lo sviluppatore deve conoscere, ad esempio i principi della fisica o della meccanica, ma che ci sia un dialogo dove i termini assumono lo stesso significato per entrambi le parti e si possano applicare al modello che si vuole implementare. Pian piano quindi si va a creare un modello sempre più complesso ma in linea con i requisiti del cliente, risolvendo quindi anche i problemi o le incomprensioni che sono emerse durante la stesura del modello o di alcuni concetti che sono poi stati inseriti nel modello.

Nel vecchio modello a cascata, erano gli esperti del business che parlavano con gli analisti, quest'ultimi a loro volta rimodellavano il contenuto e lo passavano direttamente ai programmatori. Questo approccio però è risultato essere fallimentare in quanto manca completamente di feedback con il committente, oltre al fatto che gli analisti avevano la piena responsabilità della creazione del modello basato esclusivamente sul contributo degli esperti aziendali, senza avere l'opportunità di imparare dai programmatori o acquisire esperienza con le prime versioni. Oltre a ciò, molte altre cose possono incidere sulla riuscita di un grande progetto: la burocrazia, obiettivi poco chiari, mancanza di risorse, solo per citarne alcuni, ma è l'approccio alla progettazione che determina in gran parte come il software può diventare complesso. Quando la complessità sfugge di mano il software non può più essere compreso, modificato, o esteso; solo un buon design può aiutare nella modellazione delle caratteristiche ritenute complesse. Nella progettazione alcuni di questi fattori di complessità sono di carattere tecnico, ma per molte applicazioni non lo sono, la complessità è nel dominio stesso. Quando la complessità di dominio non viene affrontata nella progettazione, non importa se la tecnologia infrastrutturale sia ben

concepita, un design di successo deve affrontare sistematicamente questo aspetto centrale del software.[12] C'è però un problema da tenere in considerazione, infatti quando diversi membri del team lavorano sullo stesso contesto, c'è una forte probabilità che il modello si frammenti. Più grande è il team più grande sarà il problema, ma sono sufficienti anche tre membri di un team per avere seri problemi. Inoltre dividendo il sistema in contesti sempre più piccoli si andrà perdendo un livello importante di integrazione e coerenza. È perciò fondamentale utilizzare un sistema di merging del codice che lavori in maniera frequente assieme a test automatizzati; è altrettanto importante istruire i team a mantenere ed utilizzare una semantica universale, coscienti del fatto che i concetti spesso si evolvono diversamente nella mente delle persone. Tutti i team hanno ruoli specifici, ma l'eccessiva separazione delle responsabilità per l'analisi, la modellazione, la progettazione e la programmazione interferisce con il Design del modello. Quando chi modella il dominio è separato dal processo di implementazione, perde rapidamente la percezione dei vincoli dell'implementazione. Il vincolo di base dell'approccio DDD è che il modello supporta un'implementazione efficace e riassume le informazioni chiave nel dominio è stato rimosso e i modelli risultanti saranno poco pratici. Se a ciò si aggiunge che le persone che scrivono il codice non si sentono responsabili del modello o non capiscono come far funzionare il modello per un'applicazione, il modello sarà completamente diverso rispetto al software. Se gli sviluppatori non si rendono conto che la modifica del codice impatta anche il modello, il loro refactoring indebolirà il modello anziché rafforzarlo. Infine, le conoscenze e le capacità di designer esperti non saranno trasferite ad altri sviluppatori se la divisione del lavoro impedisce il tipo di collaborazione che trasmette le sottigliezze della codifica di un design basato sul modello. Con DDD, e, di conseguenza con DevOps, una parte del codice è un'espressione del modello, cambiando quel codice si cambia il modello. Quindi è meglio impostare il progetto in modo che i programmatori facciano un buon lavoro di modellazione. Il risultato è un software che offre funzionalità avanzate basate su una comprensione fondamentale del dominio principale.

1.2.7 L'evoluzione delle metodologia e la nascita di DevOps

I progressi nell'informatica e lo slancio economico del settore tecnologico statunitense all'inizio del XXI secolo hanno fornito alle aziende (che non avevano mai collegato

le operazioni IT esistenti come Asset strategico) un nuovo vantaggio competitivo. Nelle precedenti espansioni economiche, le innovazioni tecnologiche erano riservate alla produzione di prodotti aziendali di base e allo sviluppo di servizi a supporto dei clienti. L'interno capitale umano, la tecnologia,i software e i sistemi avanzati hanno mantenuto stabile l'infrastruttura e i riflettori accesi per le vendite e il marketing. Aziende come Google, Netflix e Amazon hanno dimostrato come creare valore sfruttando lo sviluppo software interno e le operazioni IT per innovare e rispondere rapidamente alla crescente domanda dei clienti di applicazioni affidabili, sicure e ricche di funzionalità. Per farlo, i principi e le pratiche tradizionali delle aziende per lo sviluppo e le operazioni software dovevano adattarsi a una nuova realtà: dovevano commercializzare il software agli utenti più velocemente migliorando al contempo i livelli di servizio e la sicurezza. Diverse filosofie, come lo sviluppo Agile, sono cresciute riconoscendo che il vecchio metodo non era più sufficiente. La crescita dei principi Agile della gestione di progetti/prodotti, unita ai progressi della tecnologia informatica (come l'infrastruttura basata su cloud), ha evidenziato i conflitti tra i team di sviluppo e le operazioni IT: i loro obiettivi opposti hanno reso impossibile la distribuzione frequente di sistemi affidabili e sicuri. Presi da una maggiore domanda di mercato, che richiedeva soluzioni migliori e nel minor tempo possibile, i professionisti si sono accorti di aver bisogno di una nuova soluzione.

Il movimento DevOps è uno sforzo per trasformare il modo in cui le varie parti interessate interagiscono, comunicano e lavorano insieme nel ciclo di vita dello sviluppo software. John Willis, ex direttore dell'ecosistema di sviluppo presso Docker e autore di sei libri sulla gestione dei sistemi aziendali, *sottolinea che il motivo per cui le persone ignorano DevOps come qualcosa di nuovo o degno di attenzione è la mancanza di una* "definizione canonica". *A differenza di Agile, non esiste un* "manifesto DevOps" *per associare il movimento a un luogo di nascita e un tempo specifico o guidare coloro che cercano un insieme specifico di valori, metodi, pratiche e strumenti. Il disprezzo per l'istituzione formale del movimento (cioè la mentalità* "abbiamo fatto DevOps per anni") *deriva dal fatto che DevOps riflette i principi familiari dei processi aziendali creati nel XX secolo. È una combinazione di gestione filosofica e movimenti di produzione.*[13] Principi snelli (Lean Startup)[41], la Toyota Production Management (Toyota Kata)[42] e lo sviluppo di software Agile (integrazione continua/consegna continua, automazione dei

test software, ecc.), si sono fusi con idee che si stavano diffondendo attraverso conferenze Agile, Velocity e DevOpsDays.

Il termine **DevOps** è attribuito a *Patrick Debois*, *Andrew Shafer*, dal loro lavoro su Agile Infrastructure in una conferenza Agile del 2008. DevOps è divenuto popolare nel 2009 grazie ad una presentazione di *John Allspaw* e *Paul Hammond* alla conferenza Velocity, chiamata: ***10+Deploy Per Day: Dev and Ops Cooperation presso Flickr*** [2]. Debois riteneva che DevOps fosse una risposta all'inflessibilità e alla mentalità "noi contro loro" che le organizzazioni hanno creato mettendo "contro" sviluppatori di software, tester, manager, amministratori di database, tecnici di rete, amministratori di sistema, ecc. La tecnologia ha contribuito a creare una convergenza armonica dei principi di gestione DevOps e degli strumenti software in grado di introdurre procedure DevOps critiche, come la collaborazione, l'automazione, il monitoraggio, la registrazione e la distribuzione di software. Utilizzando una combinazione di strumenti software e una cultura di collaborazione e miglioramento costante, è nato il movimento DevOps per lo sviluppo di software.[14]

1.3 DevOps : Developer and Operations

DevOps è in realtà l'acronimo di *Developer and Operations* ed è un termine coniato per descrivere l'approccio che viene seguito utilizzando strumenti, metriche e processi per aiutare il team a offrire esperienze di alta qualità, utilizzando l'automazione per ridurre al minimo gli errori e massimizzare l'efficienza nella consegna dei prodotti. Si concentra sul consentire agli sviluppatori di utilizzare e gestire i processi e l'infrastruttura delle operazioni software senza la necessità di un team operativo separato.

Però proprio come DevOps non include Dev, non include Ops. Per comprendere DevOps, tuttavia, è importante essere consapevoli del contesto da cui provengono le persone in Ops o Dev. Una caratterizzazione di Ops è data dalla biblioteca dell'infrastruttura informatica (ITIL). ITIL si basa sul concetto di "servizi" e il compito di Ops è supportare la progettazione, l'implementazione, il funzionamento e il miglioramento di questi servizi nel contesto di una strategia globale.[15] Nello specifico, l'Ops è responsabile:

[2] https://www.youtube.com/watch?v=LdOe18KhtT4

- del provisioning di hardware e software;
- del fatto che il personale abbia competenze specifiche (skill);
- della specifica e monitoraggio degli SLA;
- della capacità di pianificazione;
- della business continuity;
- e della sicurezza delle informazioni.

Molte di queste responsabilità hanno aspetti che sono inclusi sia all'interno che all'esterno di Processi DevOps. Se si vuole parlare di quali aspetti di Ops devono essere inclusi in DevOps bisogna prendere in considerazione tutte le attività che l'Ops attualmente svolge come attività funzionali, le sue competenze personale e le sue disponibilità. Gli aspetti di queste attività che incidono su DevOps sono:

1. *Hardware provisioning.* L'hardware virtualizzato può essere allocato da un team di sviluppo o da un'applicazione con maggiore automazione.
2. *Software provisioning.* Il software sviluppato internamente sarà rilasciato dal Dev team. Altri software possono essere forniti dal team Ops.
3. *IT function provision.* Nella misura in cui un team di sviluppo è responsabile della gestione degli incidenti e degli strumenti di distribuzione, deve disporre di persone con esperienza per eseguire queste attività.
4. *Specifica e monitoraggio degli SLA.* Per quegli SLA specifici di una particolare applicazione, il Dev team è responsabile del monitoraggio, della valutazione e della risposta agli incidenti.
5. *Capacity planning(capacità di produzione).*Il Dev team è responsabile del capacity planning per le singole applicazioni e l'Ops team sarà responsabile del capacity planning complessivo.

6. *Business continuity.* Il Dev team è responsabile di quegli aspetti della continuità aziendale che coinvolgono l'architettura dell'applicazione e l'Ops team è responsabile del resto. L'Ops può fornire servizi e politiche per la Business continuity, che a loro volta vengono utilizzati dal Dev team.

7. *Informazioni di sicurezza.* Il Dev team è responsabile di quegli aspetti della sicurezza delle informazioni che coinvolgono una particolare applicazione e l'Ops team è responsabile del resto.

8. *Il numero di persone coinvolte* in DevOps dipenderà dai processi adottati dall'organizzazione. Un'organizzazione stima che il 20% del team Ops e il 20% del team Dev siano coinvolti nei processi DevOps.

Alcuni fattori che incidono sull'ampiezza del coinvolgimento dei diversi team sono:

- Il fatto che Dev diventa l'aiutante primario in caso di problemi.

- Se esiste un gruppo DevOps separato responsabile degli strumenti utilizzati nella pipeline di distribuzione continua. [16]

1.3.1 Developer and Operations: un'unica realtà

Come già detto, prima di DevOps, il team di sviluppo e il team operazionale lavoravano in modo completamente distaccato. Test e implementazione erano attività isolate che venivano eseguite solo dopo la progettazione e quindi consumavano più tempo rispetto ai cicli attuali di build. Senza utilizzare DevOps:

- i team di sviluppo e il team operazionale avevano tempistiche distinte e non sincronizzate causando ulteriori ritardi. Nello specifico Developer e Operations hanno i loro processi, strumenti e basi di conoscenza indipendenti. L'interfaccia tra loro nell'era pre-DevOps era di solito un sistema di ticket: i team di sviluppo richiedevano l'implementazione di nuove versioni del software e il personale operativo gestiva manualmente quei ticket. In questo accordo però, i team di sviluppo cercavano continuamente di spingere le nuove versioni in produzione, mentre il personale

operativo tentava di bloccare queste modifiche per mantenere la stabilità del software. Teoricamente, questo modus operandi offre una maggiore stabilità per il software in produzione però in pratica comportava lunghi ritardi tra gli aggiornamenti del codice e la distribuzione, oltre a processi di risoluzione dei problemi inefficaci.

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

- Nessuna pratica di programmazione estrema (XP), ad esempio, comprendeva la distribuzione, di conseguenza, il passaggio alla produzione tendeva ad essere un processo stressante nelle organizzazioni, infatti venivano svolte attività manuali soggette a errori e, anche, correzioni dell'ultimo minuto.[17]

Vi è anche una maggiore richiesta per aumentare la velocità di rilascio del software da parte delle parti interessate(stackholder). Secondo lo studio di consulenza Forrester però, solo il 17% dei team utilizza il software pronto per il rilascio abbastanza velocemente. Questo evidenza un punto dolente che può essere risolto utilizzando un approccio DevOps.[**?**]

1.3.2 La differenza tra DevOps e l'approccio tradizionale IT

Veranno messi a confronto il modello tradizione IT con quello DevOps per capire le modifiche apportate da DevOps. Si partirà da un esempio pratico, preso dal saggio [18] in un cui un'applicazione è programmata per essere pubblicata nelle 2 settimane a venire e la scrittura del codice è all'80%. Si presuppone che l'applicazione sia nuova e il processo di acquisto del server per la distribuzione del codice sia appena iniziata; le differenze verranno mostrate tramite la tabella 1.8.

Vecchio processo	DevOps
Dopo aver effettuato l'ordine per i nuovi server, il team di sviluppo lavora ai test. Il team operazionale si occupa delle pratiche burocratiche come richiesto dalle imprese per distribuire l'infrastruttura.	Dopo aver effettuato l'ordine per i nuovi server, il team di sviluppo e operazionale lavorano insieme sui documenti per configurare i nuovi server. Ciò si traduce in una migliore visibilità nei requisiti dell'infrastruttura.
Le proiezioni su failover, ridondanza, ubicazioni dei data center e requisiti di archiviazione sono distorte poiché non sono disponibili input da sviluppatori che hanno una profonda conoscenza dell'applicazione.	Le proiezioni su failover, ridondanza, disaster recovery, ubicazioni dei data center e requisiti di archiviazione sono piuttosto precise per via degli input degli sviluppatori.
Il team operazionale non ha idea dei progressi del team di sviluppo. Il team operazionale sviluppa un piano di monitoraggio secondo la propria comprensione.	In DevOps, il team operazionale è completamente consapevole dei progressi che gli sviluppatori stanno avendo. Il team operazionale interagisce con gli sviluppatori e sviluppa congiuntamente un piano di monitoraggio adatto alle esigenze IT e aziendali. Utilizzano anche strumenti avanzati di monitoraggio delle prestazioni delle applicazioni (Application Performance Monitoring).
Prima di andare in produzione, il test di carico fa arrestare in modo anomalo l'applicazione. Il rilascio è ritardato.	Prima di andare in produzione, il test di carico rallenta un po' l'applicazione. Il team di sviluppo risolve rapidamente i colli di bottiglia creati. L'applicazione è rilasciata in tempo.

Tabella 1.8: Approccio tradizionale IT vs DevOps

1.3.3 La differenza tra DevOps e l'approccio tradizionale Agile

Si tende a pensare che Agile e DevOps siano due metodologie distinte, quando in realtà DevOps è un miglioramento recente della metodologia Agile. La principale differenza che DevOps apporta è la tra Developer, solitamente orientati esclusivamente alla creazione di cambiamenti e all'aggiunta o alla modifica di funzionalità, e Operation, solitamente orientati allo stabilimento e miglioramento dei servizi, in tutte le fasi del ciclo di vita del prodotto, con l'obbiettivo che questa collaborazione porti ad un enorme valore sia per quanto riguarda la qualità del prodotto finale, sia per la velocità con cui questo viene realizzato. DevOps estende i principi della metodologia Agile che si fondano su valori, metodi e pratiche, concentrandosi sull'intero servizio,ed aggiunge anche gli strumenti. Secondo il Manifesto Agile infatti, gli strumenti assumono una minima importanza, ma questo ha portato nel tempo a trascurarli notevolmente con conseguenze spesso non in linea con l'obiettivo del cliente. Un altro modo di vedere le differenze tra i due metodi è che Agile, risolve i problemi della tecnologia, mentre DevOps cerca di risolvere un problema di business attraverso un insieme di pratiche, cultura e valori che portano ad avere un software più solido e che viene prodotto più velocemente, con meno problemi durante il ciclo di vita. Infatti DevOps accumuna l'obiettivo di Developers e Operators, cioè puntare all'ottimizzazione dell'intero sistema invece che alle ottimizzazioni locali, ciò implica che al raggiungimento degli obbietti di business, sarà risolto il problema del cliente, sia esso un problema di Developer o di Operation.

Capitolo 2

DevOps Perchè viene usato

Come spiegato nel capitolo precedente, se i principi di Devops vengono adottati nel ciclo di vita del rilascio si può davvero ottimizzare il rilascio del software. Le organizzazione che stanno adottando i principi DevOps avranno sicuramente un vantaggio rispetto alle organizzazioni che non lo utilizzano. I processi devono cambiare con il tempo e devono essere in linea con le esigenze del mercato in quanto quest'ultimo è altamente mutevole. DevOps infatti consente all'organizzazione di lanciare prodotti più rapidamente sul mercato,[19] e favorisce:

1. **Prevedibilità**: con questo termine si intende che DevOps offre un tasso di fallimento significativamente inferiore per le nuove releases.

2. **Riproducibilità**: qualsiasi cosa è versionata così che la versione precedente possa essere ripristinata in qualsiasi momento.

3. **Manutenibilità**: il processo di recupero deve essere immediato in caso di arresto anomalo di una nuova versione o disabilitazione del sistema corrente.

4. **Time to market**(il tempo che intercorre dall'ideazione di un prodotto alla sua effettiva commercializzazione): come già detto, DevOps riduce il time to market quasi alla metà attraverso un rilascio del software semplificato. Questo sopratutto per le applicazioni digitali e mobili.

5. **Maggiore qualità**: DevOps aiuta il team a fornire una migliore qualità dello sviluppo delle applicazioni in quanto considera anche i problemi legati all'infrastruttura.

6. **Rischio ridotto**: DevOps incorpora aspetti di sicurezza nel ciclo di vita del rilascio del software. Aiuta a ridurre i problemi durante il ciclo di vita.

7. **Resilienza**: lo stato operazionale del sistema software è più stabile, sicuro e le modifiche sono verificabili.

8. **Efficienza dei costi**: DevOps offre efficienza dei costi nel processo di sviluppo del software, che è sempre auspicabile nella gestione delle aziende IT.

9. **Suddivide codice molto grande in porzioni più piccole**: DevOps si basa sul metodo di programmazione agile. Pertanto, consente di suddividere blocchi di codice più grande in blocchi più piccoli e gestibili.

2.1 DevOps Lifecicle

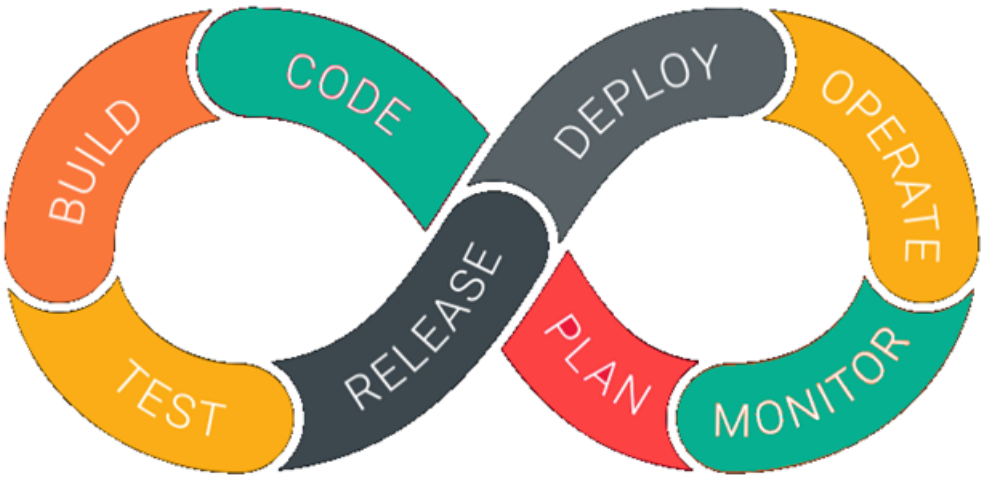

Figura 2.1: DevOps lifecycle. [50]

DevOps è una profonda integrazione tra sviluppo e operazioni. Comprendere DevOps non è possibile senza conoscere il ciclo di vita di DevOps. Ecco un breve riepilogo sul ciclo di via di DevOps (DevOps lifecycle) presente in figura 2.1:

1. **Plan**. Inizialmente va pianificato il tipo di applicazione che è necessario sviluppare. Serve per fare un quadro generale del processo di sviluppo.

2. **Code**: Scrivere il codice dell'applicazione secondo i requisiti del cliente,se i requisiti sono già stati definiti, seguire la precedente fase di plan.

3. **Build**: In questa fase, le nuove funzionalità sono integrate con il codice esistente sviluppato nella fase precedente così che poi avrà luogo il test. Lo sviluppo continuo è possibile solo grazie alla continua integrazione e test.In questa fase DevOps, lo sviluppo del software avviene costantemente. L'intero processo di sviluppo è separato in piccoli cicli di sviluppo. Ciò avvantaggia il team DevOps: accelerare lo sviluppo del software e il processo di delivery.

4. **Test**: Questo è il punto centrale dell'applicazione. Bisogna provare l'applicazione che si è creata, e riscrivere l'applicazione se necessario. Il team di controllo qualità utilizzerà strumenti come *Selenium* per identificare e correggere i bug rilevati nel nuovo codice.

5. **Releases**: Se la fase di test va a buon fine, è il momento di rilasciare l'applicazione in produzione(Live).

6. **Deploy**: Eventualmente distribuire il codice in cloud per un ulteriore utilizzo. In questa fase, il processo di distribuzione si svolge in modo continuo. Viene eseguito in modo tale che qualsiasi modifica apportata in qualsiasi momento nel codice, non influisca ad esempio sul funzionamento di un sito Web ad alto traffico.

7. **Operate**: Eseguire l'operazione sul codice, se presente.

8. **Monitor**: In questa fase, il team si prenderà cura del comportamento inappropriato del sistema o dei bug rilavati in produzione. Monitorare le prestazioni dell'applicazione secondo i requisiti del cliente.Apportare eventuali modifiche per soddisfare

i clienti. E se non si riesce a soddisfare un requisito, apportare modifiche in quella particolare area per soddisfare il cliente.

In questo modo si ripete continuamente il DevOps lifecycle.

2.1.1 Microservizi,virtualizzazione e cloud

L'implementazione efficace di strumenti e procedure DevOps per testare, distribuire, monitorare e modificare il codice e i sistemi complessi di aziende quali Netflix, Amazon, Microsoft e Google spesso richiede un'architettura software nota come microservizi. Sam Guckenheimer di Microsoft Visual Studio e autore di quattro libri sulle procedure DevOps e Agile descrive i microservizi come: "*il modello architetturale di composizione di un'applicazione distribuita da servizi distribuibili separatamente che eseguono funzioni aziendali specifiche e comunicano tramite interfacce Web.I team DevOps incapsulano singoli pezzi di funzionalità nei microservizi e costruiscono sistemi più grandi componendo i microservizi come elementi costitutivi. I microservizi applicano un esempio del principio aperto / chiuso: sono aperti per l'estensione (usando le interfacce che espongono) e chiusi per modifica (in quanto ciascuno è implementato e aggiornato in modo indipendente).*"[20] Consigliando di usare un'architettura basata sui microservizi in combinazione con le procedure DevOps per: ridurre il tempo necessario per distribuire le modifiche del codice o della configurazione,evitare SPOF(Single Point Of Failure). Per SPOF si intende indirettamente un'infrastruttura o un servizio, che contiene diverse funzionalità o servizi, e nel caso in cui un singolo servizio o la macchina che lo ospita risulterebbe non funzionante, la ripercussione si estenderebbe a tutti i servizi presente e/o ai servizi ospitata dall'infrastruttura. Un esempio pratico è un'azienda che ha una singola macchina dove al suo interno troviamo: sia numerosi software backend/frontend di una determinata azienda,sia il suo database;nel caso in cui la macchina si rompesse o necessitasse di un riavvio, l'azienda sarebbe impossibilitata a fornire qualsiasi tipo di servizio fino al completo riavvio della macchina o risoluzione del problema.

Quando invece si implementa l'architettura basata su cloud e sui microservizi, la tecnologia implica che spesso siano utilizzati dei contenitori per isolare, creare pacchetti e distribuire il codice. I contenitori sono il prossimo passo evolutivo nella tecnologia di virtualizzazione. La virtualizzazione si riferisce all'atto di utilizzare una risorsa virtuale (ad

esempio, un server, un desktop, un sistema operativo o una rete) per rendere scalabili i processi di elaborazione.I contenitori contrariamente alle macchine virtuali forniscono un metodo semplificato per configurare la gestione dei file. L'infrastruttura basata su cloud consente l'architettura dei microservizi. Le procedure DevOps critiche, come il monitoraggio e la registrazione delle modifiche del codice in produzione, richiedono piattaforme di hosting cloud scalabili, sicure e stabili.

2.1.2 Infrastruttura

Come già visto in parte nel precedente paragrafo, l'infrastruttura ha un ruolo chiave nei processi di implementazioni e rilascio, infatti sia a livello software che a livello hardware deve seguire i seguenti requisiti:

- *I membri del team devono poter lavorare su diverse versioni del sistema in modo concorrente.*
- *Il codice sviluppato da un team non deve sovrascrivere accidentalmente il codice implementato da un altro team.*
- *Il lavoro non deve essere perso se un membro dovesse improvvisamente lasciare il team.*
- *Il codice di un membro di un team deve poter essere testato facilmente.*
- *Il codice di un team deve essere facilmente integrato con il codice prodotto da un altro membro dello stesso team.*
- *Il codice prodotto da un team deve essere facilmente integrato con il codice prodotto da un altro team.*
- *Una versione integrata del sistema deve essere facilmente rilasciata in vari ambienti(test, sviluppo o produzione).*
- *Una versione integrata del sistema deve essere facilmente testata senza apportare modifiche alla versione di produzione del sistema.*

- *Una versione rilasciata recentemente del sistema deve essere attentamente monitorata.*

- *Vecchie versioni del codice devono essere disponibili nel caso in un cui ci siano problemi con la nuova versione presente in produzione.*

- *Può essere fatto il rollback in caso di problemi.*[21]

2.1.3 Principi DevOps

L'infrastruttura e il ciclo di vita dei processi DevOps o in generale se si vuole seguire un approccio DevOps bisogna seguire questi 6 principi fondamentali:

1. **Azione incentrata sul cliente**: il team DevOps deve intraprendere un'azione incentrata sul cliente per cui deve costantemente investire in prodotti e servizi.

2. **Responsabilità end-to-end**: il team DevOps deve fornire supporto alle applicazioni fino a quando queste non smettono di funzionare. Ciò migliora il livello di responsabilità e la qualità dei prodotti progettati.

3. **Miglioramento continuo**: la cultura DevOps si concentra sul miglioramento continuo per ridurre al minimo gli sprechi. Accelerare continuamente il miglioramento del prodotto o dei servizi offerti.

4. **Automatizzare tutto**: l'automazione è un principio vitale del processo DevOps. Questo non si riferisce solo allo sviluppo del software, ma anche all'intera infrastruttura.

5. **Lavorare come una squadra**: nella cultura DevOps il ruolo di progettista, sviluppatore e tester è già definito. Tutto quello che devono fare è lavorare come un'unica squadra in totale collaborazione.

6. **Monitorare e testare tutto**: è molto importante per il team DevOps disporre di solide procedure di monitoraggio e test.

2.1.4 Ruoli,responsabilità e competenze di un DevOps

I DevOps lavorano a tempo pieno. Sono responsabili della produzione e della manutenzione continua della piattaforma di un'applicazione software. Di seguito sono riportati alcuni ruoli, responsabilità e competenze di un DevOps:

- È in grado risolvere i problemi del sistema e i problemi tra piattaforme e domini di applicazioni.
- Gestisce il progetto in modo efficace attraverso piattaforme aperte e basate su standard.
- Aumenta la visibilità del progetto e la tracciabilità.
- Migliora la qualità e riduce i costi di sviluppo attraverso la collaborazione.
- Analizza, progetta e valuta script e sistemi di automazione.
- Garantisce la risoluzione critica dei problemi del sistema utilizzando i migliori servizi di soluzioni di sicurezza cloud.
- Un DevOps dovrebbe avere le competenze trasversali di risolutore di problemi e apprendimento rapido.

Questi concetti sono riepilogati in modo sintetico anche in una delle tante definizioni di DevOps: *DevOps è uno sforzo collaborativo e multidisciplinare all'interno di un'organizzazione per automatizzare la consegna continua di nuove versioni di software, garantendo al contempo la loro correttezza e affidabilità.*[22]

2.1.5 Strumenti di automazione DevOps

È fondamentale automatizzare tutti i processi di test e configurarli per ottenere velocità e agilità. Questo processo è noto come automazione DevOps. La difficoltà incontrata dal team DevOps che mantiene un'infrastruttura IT di grandi dimensioni può essere brevemente classificata in sei diverse categorie.

1. Automazione dell'infrastruttura.

2. Gestione della configurazione.

3. Automazione della distribuzione.

4. Gestione delle prestazioni(Perfomarce Management).

5. Gestione dei log.

6. Monitoraggio.

Si vedranno alcuni strumenti in ciascuna di queste categorie e in che modo risolvono le correlate problematiche.

2.1.5.1 Automazione dell'infrastruttura

Amazon Web Services (AWS): Essendo AWS un servizio cloud non è necessario essere fisicamente presenti nel data center. Inoltre si possono facilmente scalare le risorse su richiesta. Non ci sono costi hardware iniziali. Può essere configurato per eseguire automaticamente il provisioning di più server in base al traffico.

2.1.5.2 Gestione della configurazione

Chef: È un utile strumento DevOps per raggiungere velocità, scalabilità e coerenza. Può essere utilizzato per semplificare attività complesse come la gestione delle differenti configurazioni. Con questo strumento, il team DevOps può evitare di apportare manualmente modifiche ad un gran numero di server in quanto basterà apportare la modifica in un'unica posizione che si rifletterà automaticamente negli altri server.

2.1.5.3 Automazione della distribuzione

Jenkins: Questo strumento facilita l'integrazione e il test continuo. Consente di integrare più facilmente le modifiche al progetto individuando rapidamente i problemi non appena viene distribuita una nuova modifica.

2.1.5.4 Gestione delle prestazioni

App Dynamic: È lo strumento DevOps che offre il monitoraggio delle prestazioni in tempo reale. I dati raccolti da questo strumento consentono agli sviluppatori di eseguire il debug quando si verificano problemi.

2.1.5.5 Gestione dei log

È sempre più diffuso l'utilizzo dei log piuttosto che l'esecuzione del debug dell'applicazione anche se prima i programmatori utilizzavano il debug piuttosto che i log, così da non avere file di log e la gestione che ne comporta. Da un punto di vista DevOps però, i log sono importanti per tener traccia dell'applicazione. I log hanno diversi livelli, si passa per esempio da quelli più importanti come: fatal,error,warn, per arrivare a quelli di normale consultazione come: info, trace o debug. In molti linguaggi la gestione dei log è direttamente presente senza bisogno di librerie esterne, ad esempio in Java si può usare "*java.util.logging*". Oppure si può usare anche il framework **Log4j**, che è presente anche in altri linguaggi. Molti strumenti di log offrono un servizio basato sul cloud per raccogliere dati dalle applicazioni, piattaforme, e servizi in real time usando standard quali HTTP o syslog. Sono facili da installare e possiedono anche delle dashboard consultabili per operazioni di ricerca. Un strumento open source interessante è **Graylog2**: analizza i log, permette la storicizzazione dei log e la ricerca degli errori. È facile da utilizzare grazie alla sua interfaccia Web, gestisce anche una grande quantità di formati di dati e si possono aggiungere anche dei plug-in.

2.1.5.6 Monitoraggio

Nagios: È importante che il team riceva una notifica quando le infrastrutture e i servizi correlati non sono raggiungibili(down). Nagios è uno di quei strumenti che notifica il team DevOps a trovare e correggere i problemi. *Nello specifico Nagios monitora l'intera infrastruttura IT per garantire il corretto funzionamento di sistemi, applicazioni, servizi e processi aziendali. In caso di guasto, Nagios può avvisare il personale tecnico del problema, consentendogli di avviare i processi di riparazione prima che le interruzioni incidano sui processi aziendali, sugli utenti finali o sui clienti.*[23]

2.2 Come applicare DevOps alle varie fasi del rilascio del software

Ogni fase che include DevOps, è sempre preceduta dall'aggettivo "*Continuous*", nello specifico sono:

- ***Il Continuous Planning***: I businesses plans devono essere Agili, cioè in grado di adattarsi rapidamente alle continue evoluzioni del mercato. Bisogna sempre modificare/regolare i propri piani aziendali in base al mercato. Ciò comporta un'ulteriore difficoltà di adattamento ai cambiamenti per i team di sviluppo/test. *DevOps invece consente di applicare velocemente questi cambiamenti riprioritizzando i product backlog, attraverso un canale di feedback continuo con il cliente. È un processo continuo per pianificare piccole porzioni, per reagire di conseguenza ai feedback ottenuti e aggiustare il piano di lavoro se necessario, questo processo è un processo continuo.*[24]

- ***La Continuous Integration*** (abbreviato in CI): è una pratica che consiste nell'automatizzare l'integrazione delle modifiche del codice fatte da diversi contributori in un unico progetto software. Il processo CI comprende strumenti automatici che testano la correttezza del nuovo codice prima di integrarlo.Un sistema di versioning rappresenta la parte fondamentale del processo CI ed è inoltre integrato con altri controlli quali: test della qualità del codice,strumenti di revisione della sintassi e altri. Un feedback sulla qualità del codice può essere fornito da strumenti di analisi come SonarQube e Analizo. SonarQube classifica i problemi di codice e valuta le metriche di copertura (coverage metrics), nonché i debiti tecnici(le possibili complicazioni che subentrano in un progetto, qualora non venissero adottate adeguate azioni volte a mantenerne bassa la complessità) in diversi linguaggi di programmazione tramite i suoi plugin. Analizo supporta l'estrazione di una varietà di metriche del codice sorgente per i linguaggi C, C++, C# e Java. In generale, gli strumenti di analisi del codice sorgente possono segnalare problemi nel codice sorgente, come non conformità allo stile standard, problemi di manutenibilità, rischio di bug e persino violazioni della vulnerabilità. Gli strumenti di analisi statica del codice

sorgente variano nei linguaggi di programmazione supportati e nel modo in cui vengono forniti. Possono essere forniti come servizio, ad esempio Code Climate, o persino all'interno dell'ambiente di sviluppo tramite strumenti come PMD per l'IDE Eclipse.[25]

- ***Il Continuous Delivery e Continuous Deployment*** (entrambi abbreviati in CD): rappresentano la parte centrale del processo DevOps, il tempo impiegato per rilasciare una nuova funzionalità ha un forte impatto su tutto il sistema, più velocemente viene rilasciata una nuova funzionalità, più velocemente saranno effettive le modifiche e l'eventuali correzioni, oltre al fatto che ciò non deve comportare disservizi. Se il processo di deploy, fosse manuale, ci sarebbe un alto rischio di errori umani senza considera il fatto che dovrebbe essere seguito da una persona togliendo quindi tempo alle sue attività. Oltre all'errore, anche il deploy stesso sarebbe più lento, in quanto non automatizzato e dettato da tempi "*umani*".

- ***Il Continuous Testing***: il prerequisito per il continuous testing è automatizzare ogni test case. Non è pensabile un approccio in cui ogni nuova funzionalità viene provata attraverso un test eseguito manualmente dal programmatore e mai ripetuto. I test devono essere definiti a priori, ed eseguiti ogni volta che viene implementata una nuova funzionalità, in questo modo, il programmatore può continuare a programmare, in quanto i test, essendo automatici e quindi senza una gestione manuale, saranno veloci garantendo un rapido rilascio del codice.Ogni linguaggio di programmazione ha anche alcuni framework di Unit test che forniscono agli sviluppatori un feedback vitale sulla correttezza del software. Alcuni esempi sono JUnit e Mockito per Java, RSpec per Ruby e NUnit per .NET. Test più complessi che automatizzano il comportamento dell'utente finale per le applicazioni Web sono possibili con strumenti di automazione della navigazione, come Selenium.

- ***Il Continuous Monitoring***: grazie alla capacità di poter testare velocemente le nuove funzionalità e l'impatto che queste hanno sull'intero sistema, i test possono essere applicati anche all'ambiente di produzione, ciò permette di monitorare continuamente i vari ambiente e poter agire di conseguenza, oltre ad avere anche una panoramica sulle prestazioni.

Le varie fasi sono visibili nell'immagine [54].

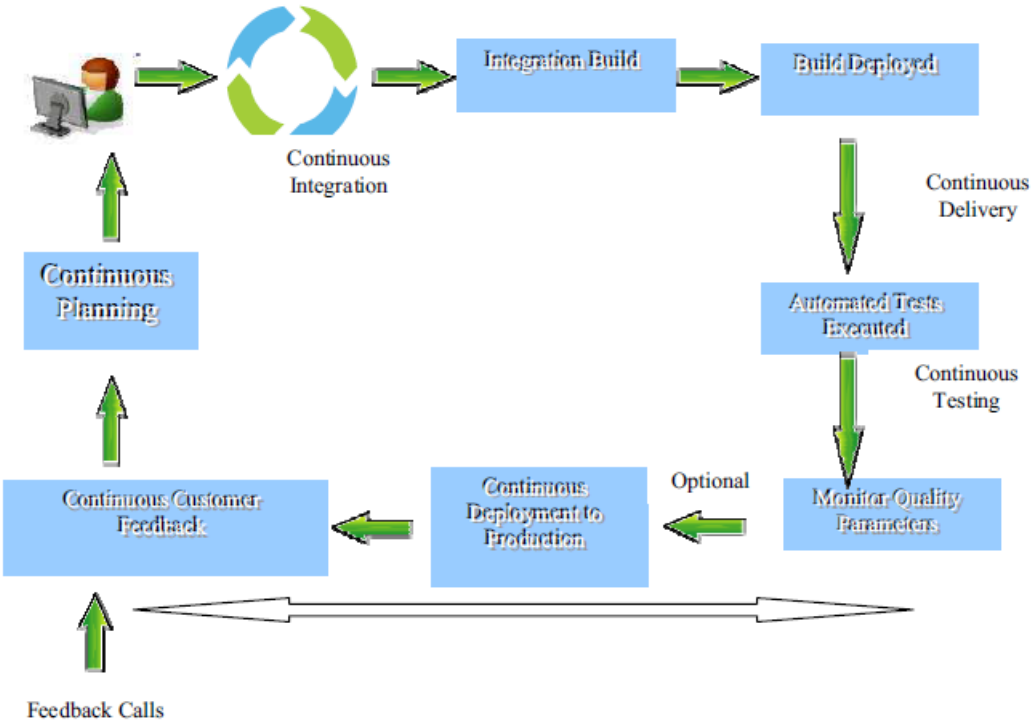

Figura 2.2: Continuous DevOps. [54]

2.2.1 CI e CD

Per quanto riguarda la CI e il CD che sono state precedentemente introdotte,l'immagine [55] chiarisce meglio quale sia l'ordine e il compito delle tre fasi.

Figura 2.3: CI e CD Flow. [55]

In generale la Continuous Integration e il Continuous Delivery/Deployment sono processi dove il proprio team di sviluppo apporta frequenti modifiche al codice che vengono inserite nel branch principale garantendo allo stesso tempo che non abbiano alcun impatto sulle modifiche apportate dagli sviluppatori che lavorano in parallelo su altri branch. Lo scopo è ridurre la possibilità di errori e conflitti durante l'integrazione del progetto.

2.2.1.1 Definizione e storia

Cosa si intende per Continous integration? È una pratica che si applica in contesti dove lo sviluppo del software avviene attraverso un sistema di versioning. Consiste nell'allineamento frequente (ovvero "molte volte al giorno") dagli ambienti di lavoro degli sviluppatori con l'ambiente condiviso (mainline). Il concetto è stato originariamente proposto nel contesto dell'extreme programming (XP), come contromisura preventiva per il problema dell'"*integration hell*" (le difficoltà dell'integrazione di porzioni di software sviluppate in modo indipendente su lunghi periodi di tempo e che di conseguenza potrebbero essere significativamente divergenti). La CI è stata originariamente concepita per essere complementare rispetto ad altre pratiche, sopratutto quelle legate al Test Driven Development (sviluppo guidato dai test). In particolare, si suppone generalmente che siano stati predisposti test automatici che gli sviluppatori possono eseguire immediatamente prima di rilasciare le loro modifiche verso l'ambiente condiviso, in modo da garantire che le modifiche non introducano errori nel software esistente. Per questo motivo, la CI viene spesso applicata in ambienti dove sono presenti sistemi di build automatici e/o esecuzioni automatiche di test, come ad esempio *Jenkins*.

2.2.1.2 Perchè si utilizza

Prima di effettuare una modifica sul codice sorgente, uno sviluppatore scarica dal repository il branch dov'è presente il codice. Man mano che gli altri sviluppatori fanno delle modifiche sul repository principale, la copia che ha lo sviluppatore, smette di rappresentare il codice del repository. Le modifiche effettuate non si riflettono unicamente sul codice, ma anche sulle librerie ed altre risorse che possono creare dipendenze e potenziali conflitti. Più è lungo il tempo durante il quale un branch di codice rimane scaricato, modificato e non committato, maggiore è il rischio di conflitti di integrazio-

ne quando il branch viene reintegrato nel repository. Quando gli sviluppatori vogliono eseguire il Push del codice, devono prima aggiornare il proprio branch locale con i cambiamenti che sono stati fatti al repository principale(Pull) dal momento in cui loro hanno scaricato la copia locale. Più modifiche sono state apportate e più lavoro è necessario fare prima di eseguire il Push delle proprie modifiche. È possibile che il repository sia diventato così diverso dalla copia locale dello sviluppatore che il tempo necessario per integrare le proprie modifiche con quelle fatte dagli altri supera il tempo necessario ad eseguire le modifiche. Questo fenomeno viene descritto come *inferno dell'integrazione* (**integration hell**) o *inferno dei merge* (**merge hell**). In uno scenario estremamente negativo, gli sviluppatori potrebbero dover annullare le proprie modifiche e doverle rifare da capo.

L'integrazione continua è un processo che consiste nell'integrare presto e spesso, limitando le possibilità dell'inferno dell'integrazione. La pratica cerca di minimizzare il lavoro inutile e risparmiare tempo.

2.2.1.3 Pratiche comuni

Di seguito verranno riportati i principî generali che permettono di svolgere l'integrazione continua.

1. Mantenere un repository del codice sorgente: questo elemento è propedeutico a tutti gli altri principi, poichè senza avere un repository del codice è impossibile automatizzare il build ed i test. Per questo motivo va utilizzato un repository Git, come ad esempio: **Github**,**GitLab**, **Bitbucket**, etc.

2. Automatizzare il build: per build si intende il processo che trasforma il codice sorgente in un artefatto. Deve essere possibile eseguire la compilazione e la creazione dei pacchetti da mettere sui server in modo completamente automatizzato, senza alcun intervento umano.Per questo si possono utilizzare : **Maven**, **Apache Ant**, **Gradle** principalmente per progetti Java, invece abbiamo Pip per Python, RubyGems per Ruby, NuGet per .NET. Gradle facilita l'implementazione di attività personalizzate durante la compilazione, come anche GNU Make, che viene spesso usato per pacchetti GNU / Linux, o Rake per Ruby. Alcuni degli strumenti citati,

come Maven, sono anche responsabili della produzione del pacchetto distribuibile, come i file WAR per gli ambienti Java. Tuttavia, per alcuni linguaggi, come Python e Ruby, non è necessario produrre un pacchetto distribuibile come artefatto a file singolo. È anche possibile usare strumenti di pacchetto più generici accoppiati all'ambiente di destinazione, come i pacchetti Debian.[27]

3. Rendere il build auto-testante: ogni volta che il codice sorgente viene buildato ed impacchettato, è importante che vengano eseguiti dei test sul sorgente affinché la qualità del codice venga tenuta sotto controllo ed eventuali bug vengano scoperti il prima possibile. Per questo si può utilizzare: **Jenkins, Buddy, Bamboo, TeamCity, CircleCI** o altri.

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

5. Ogni commit fa partire una build. Ogni modifica al codice sorgente condiviso potrebbe generare dei bug e quindi compilare e testare subito dà la possibilità di intervenire immediatamente su eventuali bug. Per questo veranno utilizzate le funzionalità di **Jenkins e di Github**. Questo principio è importante a seguito di un altro principio, detto in inglese *fail fast*, ovvero se un commit di uno sviluppatore è tale da far fallire un test, bisogna intervenire immediatamente prima che si crei una sequenza di errori successivi alla modifica apportata.

6. Fare in modo che il build sia veloce: un build troppo lento può rendere poco agevole il far partire i build automatici dopo i commit, e potrebbe costringere gli sviluppatori ad aspettare la fine del build prima di effettuare un commit o di verificarne l'esito. Per questo i test verranno eseguiti da **Jenkins** e non durante il build del progetto.

7. Eseguire i test in un clone dell'ambiente di produzione: se l'ambiente su cui vengono eseguiti i test non è assolutamente identico all'ambiente di produzione, c'è il rischio che qualcosa che è già stato testato generi dei bug inspiegabili una volta rilasciato

in produzione, con conseguenti danni economici, è sempre bene che i due ambienti siano allineati il più possibile.

8. Fare in modo che sia facile prendere le ultime versioni dei pacchetti: questo principio serve quando si hanno tanti ambienti che devono essere tenuti allineati. I pacchetti saranno sempre versionati e disponibili tramite l'apposito repository, i più comuni sono **Jitpack**, **Nexus**, **Maven repository**.

9. Tutti possono vedere i risultati dell'ultimo build: La trasparenza permette di stabilire il livello di qualità raggiunto da ogni modulo e capire quali sono i moduli che hanno bisogno di maggiore attenzione. Essendo Jenkins un tools, qualsiasi utente sarà informato dell'esito del build, oltretutto se si dispone di un account di un server di posta si può anche configurare l'invio di una notifica tramite email dell'esito negativo/positivo dell'ultimo build.

2.2.2 Continuous integration

La CI, come già detto, è sia un approccio che un insieme di pratiche attraverso cui gli sviluppatori apportano modifiche frequenti al proprio codice, validandole e integrandole ogni volta. Viene detta continua in quanto è un'attività che continuerà sempre ad essere applicata durante tutta la durata di vita del software.

2.2.2.1 Come avviene la CI

Un modo per definire l'integrazione continua è disporre di trigger automatici tra una fase e la successiva, fino ai test di integrazione. Cioè, se la compilazione da esito positivo, vengono attivati i test di integrazione. In caso contrario,allo sviluppatore viene notificato l'errore commesso. Il Continuous Delivery è definito come con trigger automatico fino al sistema di gestione temporanea. L'acronimo UAT, presente in un riquadro della figura 5.1 sta per User Acceptance Test(test di accettazione dell'utente)/staging/performance test(test delle prestazioni).Verrà utilizzato il termine *Staging* per queste varie funzioni. Il Continuos deployment significa che anche il passaggio successivo all'ultima fase (ovvero la distribuzione nel sistema di produzione) è automatizzato. Una volta che un servizio è distribuito in produzione, viene attentamente monitorato per un certo periodo di tempo

e poi viene promosso nella normale produzione. In questa fase finale, il monitoraggio e i test continuano ad esserci, ma il servizio non risulta diverso dagli altri servizi in considerazione.

2.2.2.2 Software/tools maggiormente utilizzati per progetti IT: esempio di utilizzo

Elencheremo brevemente una serie di tools che vengono maggiormente utilizzati insieme per eseguire la Continuous Integration e il Continuous Delivery/Deployment applicati ad un semplice progetto IT con Web Services.

- **Docker**: è un progetto open-source che automatizza il deployment (consegna o rilascio al cliente, con relativa installazione e messa in funzione o esercizio, di una applicazione o di un sistema software tipicamente all'interno di un sistema informatico aziendale) di applicazioni all'interno di container software, fornendo un'astrazione aggiuntiva grazie alla virtualizzazione a livello di sistema operativo. Docker utilizza le funzionalità di isolamento delle risorse del kernel Linux come ad esempio cgroups e namespaces per consentire a "*container*" indipendenti di coesistere sulla stessa istanza di Linux, evitando l'installazione e la manutenzione di una macchina virtuale:

- **GIT**: è un Software di controllo versione distribuito utilizzabile da interfaccia a riga di comando. Serve per mantenere un progetto di sviluppo distribuito.

- **Github**: è un servizio di hosting per progetti software. Il nome "GitHub" deriva dal fatto che GitHub è una implementazione dello strumento di controllo versione distribuito Git.

- **Jenkins**: è lo strumento open source di Continous integration, che in un certo modo unifica tutti i vari software quali Git, maven, junit in un unico tools. Viene eseguito lato server all'interno di una pagina web ed è totalmente configurabile dall'utente.

- **JitPack**: è package repository che viene usato per scaricare e caricare le librerie necessarie. Grazie all'uso combinato di Github è possibile reperire i jar che sono stati caricati su Github.

- **Maven**: è un software usato per la gestione di progetti Java e build automation. Per funzionalità è similare ad Apache Ant, ma basato su concetti differenti. Usa un costrutto basato su *POM*(Project Object Model): un file XML che descrive le dipendenze fra il progetto e le varie versioni di librerie necessarie nonché le dipendenze fra di esse. In questo modo si separano le librerie dalla directory di progetto utilizzando questo file descritto per definirne le relazione. Maven effettua automaticamente il download di librerie Java e plug-in Maven dai vari repository definiti all'interno del POm, scaricandoli in locale(nella cartella .m2) o in un repository centralizzato lato sviluppo. Questo permette di recuperare in modo uniforme i vari file JAR e di poter spostare il progetto indipendentemente da un ambiente all'altro avendo la sicurezza di utilizzare sempre le stesse versioni delle librerie.

- **MySql**: è un Relational database management system (RDBMS) composto da un client a riga di comando e un server. È disponibile sia per sistemi Unix e Unix-like sia per Windows; le piattaforme principali di riferimento sono Linux e Oracle Solaris.

- **Wildfly**: è l'application server dove viene eseguito il deploy di un progetto, precedentemente era noto con il nome Jboss, in generale implementa le specifiche di JAVA EE.

2.2.2.3 I passi della continous integration in ambito IT: esempio di utilizzo

Si son visti gli strumenti per la Continous Integration, ora si andrà ad analizzare come usarli insieme. Prendendo come spunto un *"semplice"* progetto che interroga un database: inserendo,modificando, eliminando o semplicemente prendendo i dati al suo interno. Perchè per un progetto del genere si usa la Continous Integration? Principalmente perchè :

- il database potrebbe essere condiviso e usato anche da altri progetti.

- Per essere sempre sicuri e consapevoli delle modifiche che vengono apportate al progetto.
- Per non dover perdere ulteriore tempo nel testare il proprio codice.

Si partirà dal *primo punto*; cosa significa che il database può essere usato anche da altri progetti? Semplicemente è bene che il progetto che esegue il mapping degli oggetti presenti nel database in Java Entities, sia comune a tutti e usabile come una qualsiasi libreria Java. Per quanto riguarda il *secondo punto*: non esiste Continous Integration senza la definizione di test, questo serve per essere sicuri che il nostro deploy non sia solo giusto a run time o a Maven compile, ma che anche il nostro stesso codice sia sempre "*come ci aspettiamo che sia*". Il *terzo punto* ovviamente comprende il fatto di usare strumenti quali Jenkins, dove sono stati configurati dei test per cui se dovessero insorgere errori, lo sviluppatore sarebbe subito informato dell'esito;ciò permette anche di non dover perdere ulteriore tempo in test manuali, che oltre a poter generare errori umani di digitazione e/o altri, rallenta anche il tempo di esecuzione dei test che non sarebbe più lanciati da un meccanismo automatico ma da un click dell'utente.

2.2.3 Continuous Delivery e Continuos Deployment

Le pratiche associate al Continuous Delivery hanno lo scopo di abbreviare il tempo tra un commit di uno sviluppatore in un repository e, grazie alla distribuzione continua, automatizzare il rilascio del codice convalidato in un repository.Ne consegue che, affinché il processo di distribuzione continua sia efficace, è importante che la CI sia già integrata nel flusso di sviluppo. Il continuous delivery enfatizza anche i test automatizzati per aumentare la qualità del codice che verrà portato in produzione.L'acronimo UAT, presente in un riquadro della figura 2.4 sta per User Acceptance Test(test di accettazione dell'utente)/staging/performance test(test delle prestazioni).Si userà il termine *Staging* per queste varie funzioni.

Il Continuos deployment significa che anche il passaggio successivo all'ultima fase (ovvero la distribuzione nel sistema di produzione) è automatizzato. Una volta che un servizio è distribuito in produzione, viene attentamente monitorato per un certo periodo di tempo e poi viene promosso nella normale produzione. In questa fase finale, il

monitoraggio e i test continuano ad esistere, ma il servizio non risulta diverso dagli altri servizi in considerazione.

2.2.3.1 Deployment pipeline

Una pipeline di distribuzione, come mostrato nella Figura 2.4, è costituita dai tutti i passaggi che si susseguono tra il commit del codice sviluppato da uno sviluppatore e il codice che è stato effettivamente portato in produzione, garantendo alta qualità.

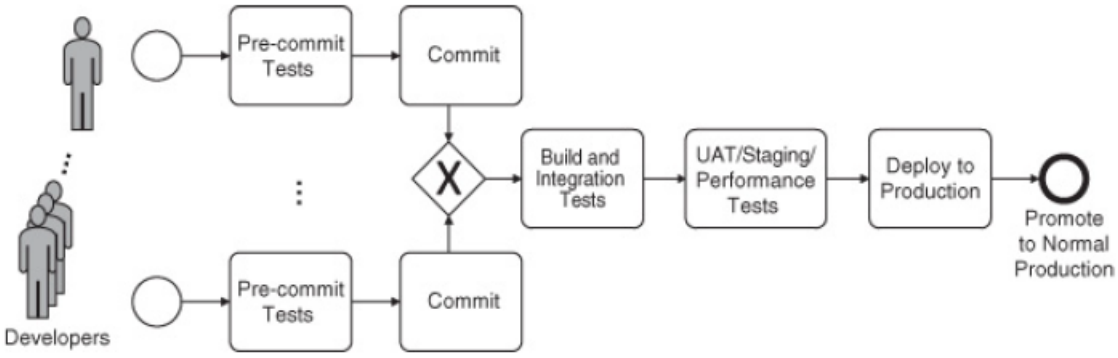

Figura 2.4: Deployment pipeline. [52]

La pipeline di distribuzione inizia quando uno sviluppatore esegue il commit del proprio codice in un sistema di versioning. Prima di eseguire questo commit, lo sviluppatore avrà eseguito una serie di test sul proprio ambiente locale;ovviamente se questo genera degli errori nella fase di test, non verrà fatto il commit di questo codice. Un commit innesca quindi una build di integrazione del servizio sviluppato. Questa build è testata da dei test di integrazione. Se questi test danno un esito positivo, la build viene promossa in un ambiente di "*quasi produzione*" detto l'ambiente di gestione temporanea: dove viene nuovamente testato. Dopo un altro periodo di supervisione ed un esito positivo, verrà promosso alla normale produzione. I compiti specifici possono variare un po' a seconda delle organizzazioni. Ad esempio, una piccola azienda può non avere un ambiente di gestione temporanea o supervisione speciale per una nuova versione distribuita. Una società più grande può invece avere diversi ambienti di produzione per scopi diversi. Ciò è attuabile principalmente se i membri del team lavorano con lo stesso sistema operati-

vo, con le stesse librerie, e lo stessa versione dello stesso IDE. Oltre a ciò, l'architettura del sistema dovrebbe essere strutturata in modo tale che non ci siano differenze tra i vari ambienti, così da garantire un test puntuale anche in ambienti diversi. Per questo scopo possono essere usati dei tool di virtualizzazione che creano istantaneamente, a seconda della configurazione data, delle macchine specifiche di sviluppo.Ogni team può usare degli script ad esempio tramite [1] *Vagrant*, che ovviamente sono versionabili, per creare una nuova macchina con le stesse specifiche di un'altra, semplicemente digitando un comando. Se una di queste macchine dovesse cambiare, basterà committare la modifica che verrà poi trasmessa anche alle altre macchine, così da garantire sempre una versione aggiornata e allineata con tutto il team di sviluppo. Esitono anche altri tool come [2] *CloudFormation*, [3] *Chef*,[4] *Puppet* e [5] *Ansible* che possono essere usati per mantenere delle configurazioni specifiche a seconda dell'ambiente.

2.2.3.2 Esempio di Continuous Deployment: IMVU

IMVU, Inc. è una società che ha dato il nome al suo prodotto principale, una chat basata su avatar 3D. *IMVU si basa sull'integrazione continua, infatti i suoi sviluppatori committano spesso ed ogni commit innesca un'esecuzione di una suite di test. IMVU ha un migliaia di file di test, distribuiti su 30-40 macchine e l'esecuzione della suite di test richiede circa nove minuti. Una volta che un commit ha superato tutti i test, viene automaticamente inviato in produzione. Questa operazione richiede circa sei minuti. Il codice viene spostato su centinaia di macchine nel cluster, inizialmente però il codice viene reso attivo solo su un numero limitato di macchine (canaries). Un programma di campionamento esamina i risultati dei canaries e se son verificati degli errori, la versione precedente viene automaticamente ripristinata. In caso contrario, il resto del cluster viene reso attivo.IMVU distribuisce in media,50 volte al giorno il nuovo codice. L'essenza del processo è nella suite di test. Ogni volta che un commit attraversa la suite di test dando esito positivo ma poi si verificano degli errori per cui viene eseguito il*

[1] https://www.vagrantup.com/
[2] https://aws.amazon.com/it/cloudformation/
[3] https://www.chef.io/
[4] https://puppet.com/
[5] https://www.ansible.com/

rollback in produzione, viene generato un nuovo test che avrebbe rilevato la distribuzione errata e viene aggiunto alla suite di test. Da notare che una suite di test completa (con la sicurezza della distribuzione in produzione) che richiede solo nove minuti per l'esecuzione è rara per i sistemi su larga scala. In molte organizzazioni, la suite di test completa in produzione che garantisce la sicurezza della distribuzione può richiedere ore per essere eseguita, cosa che spesso da luogo ad esecuzioni notturne.[1]

2.2.3.3 Best Practise

Il rilascio di un nuovo sistema o di una sua nuova versione ai clienti è una delle fasi più delicate del ciclo di sviluppo del software. Se il sistema viene utilizzato da più di una persona, il rilascio di una nuova funzionalità/versione apre la possibilità di incompatibilità o guasti, con conseguente malcontento da parte dei clienti. Di conseguenza, le organizzazioni prestano molta attenzione alla stesura di un piano di rilascio. Tradizionalmente, la maggior parte dei passaggi viene eseguita manualmente ed include:

- definire e concordare i piani di rilascio e distribuzione con i clienti o con le parti interessate. Questo potrebbe essere fatto a livello di team o di organizzazione. I piani di rilascio e distribuzione considereranno quelle funzionalità da includere nella nuova versione, oltre a garantire che: il personale operativo (incluso l'helpdesk e il personale di supporto) sia a conoscenza degli orari di distribuzione, siano soddisfatti i requisiti delle risorse e qualsiasi formazione aggiuntiva che potrebbe essere richiesta sia in programma.

- Assicurarsi che ciascun pacchetto di rilascio sia costituito da un insieme di risorse e componenti di servizio correlati compatibili tra loro. Tutto cambia nel tempo, comprese librerie, piattaforme e servizi dipendenti. Le modifiche possono introdurre incompatibilità. Questo passaggio ha lo scopo di impedire che le incompatibilità diventino evidenti solo dopo la distribuzione.

- Garantire l'integrità di un pacchetto di rilascio e dei suoi componenti che verranno mantenuti durante le attività di transizione e registrati accuratamente nel sistema di gestione della configurazione. Questo passaggio è composto da due parti: il primo è assicurarsi che le versioni precedenti di un componente non vengano

inavvertitamente incluse nella versione, e il secondo è assicurarsi che vengano conservati i componenti di questa distribuzione. Conoscere gli elementi che verranno distribuiti è importante, così da avere la consapevolezza di quali erano in essere quando si è verificato l'errore.

- Garantire che tutti i pacchetti di rilascio e distribuzione possano essere tracciati, installati, testati, verificati o disinstallati e ripristinati, se necessario. Potrebbe essere necessario eseguire il rollback delle distribuzioni (nuova versione disinstallata, vecchia versione ridistribuita) in una varietà di casistiche, come ad esempio per via di errori nel codice, risorse inadeguate oppure per via di licenze o certificati scaduti. Le attività precedentemente elencate possono essere eseguite con diversi livelli di automazione, ma se tutte queste attività venissero svolte principalmente attraverso il coordinamento umano, allora sarebbe richiesta una grande quantità di lavoro, un grande tempo d'attesa e sono altamente soggette ad errori. Qualsiasi automazione riflette un coordinamento sul processo di rilascio a livello di team o organizzazione. Poiché gli strumenti vengono generalmente utilizzati più di una volta, una procedura comune e codificata del processo di rilascio ha valenza non solo per una versione, ma per tutte.Nel caso in cui si fosse tentati di non dare peso all'automatismo e alla necessità di controllare la distribuzione, si consiglia di prendere in considerazione alcuni episodi apparsi che ha comportato una grandissima perdita per le aziende coinvolte. Tra queste abbiamo:

  - Il 1 Agosto 2012, Knight Capital ha avuto un errore di upgrade che gli è costato 440 milioni di dollari.[26]
  - Il 20 agosto 2013, Goldman Sachs ha avuto un errore di aggiornamento che, potenzialmente, ha avuto un costo di milioni di dollari. [28]

  Questi sono solo due dei molti esempi che hanno provocato tempi di inattività o errori a causa di problemi durante la fase di rilascio. La distribuzione corretta di un aggiornamento è un'attività significativa e importante per un'organizzazione e, tuttavia, deve essere eseguita velocemente garantendo la minima possibilità di errore.

2.3 Glossario dei termini DevOps

Di seguito è riportato un elenco di termini essenziali che fanno parte dell'universo DevOps:

- **CAST Application Intelligence Platform (AIP)**[6]: Un engine di Software Intelligence che consente alle grandi Organizzazioni IT di misurare in modo accurato,ripetibile ed economico i Rischi Operativi presenti alli'interno dei propri asset applicativi nonché la Produttività all'interno dei processi di sviluppo e manutenzione del software.

- **ChatOps e RocketChat**: ChatOps è un modello che collega persone, strumenti, processi e automazione attraverso interazioni guidate dalla conversazione mediate da strumenti. Rocket Chat[7] è invece lo strumento di comunicazione che implementa ChatOps consentendo un più alto grado di automazione e integrazione degli strumenti attraverso l'uso di webhook e chatbot.[29]

- **Configuration Management (Gestione della configurazion,abbreviato in CM)**: Questa pratica gestisce sistematicamente le modifiche nel tempo per mantenere l'integrità del sistema. Nello sviluppo software, il processo CM identifica le modifiche del sistema in base agli attributi in momenti specifici del tempo, per gestire le modifiche nel corso del ciclo di vita del software. In DevOps, questo viene gestito con l'automazione per rendere il processo ripetibile e affidabile su larga scala e per aumentare i requisiti di velocità.

- **Cycle Time (Tempo di ciclo)**: è il tempo necessario per progettare, creare, distribuire e monitorare il software fornito agli utenti dopo l'osservazione, la sperimentazione e l'analisi di casi aziendali sui dati. I tempi di ciclo ridotti derivano da pratiche DevOps come la distribuzione continua, l'automazione e la progettazione dell'architettura in microservizi.

- **DataOps**: si tratta di un approccio all'analisi dei dati. DataOps è un metodo orientato ai processi che deriva dallo sviluppo Agile, dalla gestione Lean, e da

[6] `https://www.castsoftware.com/products/application-intelligence-platform`
[7] `https://rocket.chat/`

DevOps. Cerca di unire i processi statistici di monitoraggio e controllo con altri metodi per migliorare la velocità effettiva, la qualità, la sicurezza e l'affidabilità dell'analisi dei dati su larga scala.

- **Infrastructure as Code(Infrastruttura come codice)**: questa è la pratica descritta da SRE (vedi sopra). In questo processo, si gestisce l'infrastruttura di sviluppo software (sistemi) a livello di codice, utilizzando codice,tecniche (ad esempio l'integrazione continua) e metodi ottimizzati e ripetibili per velocizzarne la distribuzione. Gli amministratori di sistema interagiscono con l'infrastruttura di sistema, spesso in un ambiente cloud, utilizzando strumenti (codice software) e automazione anziché configurazioni manuali.

- **SAFe Framework: Scaled Agile Framework (SAFe®)**[8]: è un approccio scalabile e modulare per l'implementazione Agile in modo che soddisfi al meglio le esigenze dell'organizzazione. È disponibile come knowledge base online per l'implementazione dello sviluppo Lean-Agile. SAFe fornisce "indicazioni complete per il lavoro a livello di portfolio, soluzione di grandi dimensioni, programma e team".

- **Site Reliability Engineering (SRE)**: Fondato da Ben Treynor Sloss, VP Ingegnere presso Google.Site Reliability Engineering (SRE) è una disciplina di ingegneria informatica dedicata ad assistere le organizzazioni nell'ottenimento dei livelli di affidabilità appropriati per i sistemi, i servizi e i prodotti. Un team SRE utilizza il software come strumento principale per "gestire, mantenere e pensare" sistemi ed utilizza anche un insieme di principi e pratiche di lavoro. SRE sfrutta ingegneri con competenze ed esperienze nell'automazione software e, dunque, in quelle tecnologie che permettono all'uomo di delegare compiti ad un sistema informatico. In generale un team SRE è responsabile della disponibilità del servizio, della gestione della latenza, delle prestazioni, dell'efficienza del sistema, della gestione delle modifiche, del monitoraggio, della risposta alle emergenze e della pianificazione delle operazioni. Il nuovo termine "Site Reliability Engineering" è emerso come relativo al lavoro DevOps in fase di esecuzione in quanto, la figura DevOps, oltre ad automatizzare il processo di consegna, si concentra anche sull'utilizzo automatizzato

[8] `https://www.scaledagileframework.com/about/`

del monitoraggio per migliorare le proprietà del software come: le prestazioni, la scalabilità e la disponibilità. [30]

- **Systems Administration (SA)**: un amministratore di sistema è responsabile della gestione, della configurazione, del funzionamento e della manutenzione delle operazioni dei sistemi in un ambiente di elaborazione (ad esempio, in una rete di computer o server). Un amministratore di sistema risolve i problemi relativi: ai sistemi informatici, alle applicazioni, alla gestione degli utenti dei sistemi, all'implementazione dei criteri di sicurezza,alla formazione degli utenti e alla gestione dei progetti relativi ai sistemi.

- **Validated Learning (Apprendimento convalidato**: viene utilizzato in Scrum, e prevede l'utilizzo di dati ricavati da cicli di feedback che coinvolgono tutte le parti interessate, per promuoverne il miglioramento continuo.

- **Version Control(Controllo della versione)**: monitoraggio delle modifiche utilizzando strumenti come GitHub per comunicare la propria modifica, durante lo sviluppo quotidiano del software, e per integrarsi con altri strumenti software.

2.4 Qual è il futuro di DevOps

Ci sono un sacco di cambiamenti che si verificano nel mondo DevOps alcuni più importanti sono:

- Le organizzazioni stanno restringendo i tempi di rilascio di nuove funzionalità da anni a mesi/settimane.

- Presto si vedrà che le figure DevOps hanno più accesso e controllo sull'utente finale rispetto a qualsiasi altra persona nell'azienda.

- DevOps sta diventando una competenza preziosa per gli IT. Ad esempio, un sondaggio condotto nel 2019 da StackOverFlow[9], ha rilevato che sono pochissime le

[9] `https://insights.stackoverflow.com/survey/2019?utm_content=launch-post&utm_source=twitter&utm_medium=social&utm_campaign=dev-survey-2019`

persone con competenze DevOps in cerca di lavoro, ed è anche un ruolo molto ricercato e pagato.

- DevOps e continuous delivery sono due aspetti che saranno costantemente sempre. Pertanto le aziende dovranno cambiare mentalità ed adattarsi al cambiamento. Tuttavia, le nozioni basilari di DevOps richiedono da 5 a 10 anni.

Capitolo 3

DevOps applicato all'IoT e al mobile

3.1 L'evoluzione DevOps

Come DevOps si è evoluto negli anni, anche il contesto circostante si è voluto, infatti è nato il nuovo termine "Iot".

3.1.1 Iot: definizione

"The Internet of things (IoT) is the network of physical devices, vehicles, and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data."[31]

L'**Iot** (Internet of Things) è l'insieme degli oggetti e dei sensori che ricevono e trasferiscono i dati su reti wireless senza richiedere interventi manuali. Ciò si ottiene integrando semplici dispositivi di elaborazione con i sensori presenti negli oggetti. Gli oggetti si integrano con gli altri presenti nell'ambiente in cui sono situati, con gli utilizzatori e con la rete che eventualmente gli informa della presenza di altri oggetti.Grazie a ciò, si possono ottenere dei comportamenti più o meno smart tramite l'elaborazione dei dati e l'eventuale uso di algoritmi di intelligenza artificiale. Ad esempio, un "*termostato smart*", può ricevere dati sulla posizione dell'utente mentre questo è in viaggio, e utilizzarli per regolare la temperatura domestica prima del suo arrivo. Non è necessario che l'utente intervenga e il risultato è migliore rispetto ad una regolazione manuale e quindi dettata

dall'utente(che può anche dimenticarsi di farlo). Un tipico sistema IoT, come il termostato smart sopra descritto, funziona grazie all'invio, alla ricezione e all'analisi dei dati in un ciclo continuo di feedback. A seconda del tipo di sistema IoT, l'analisi può essere eseguita o tramite intervento manuale o da tecnologie di intelligenza artificiale e machine learning (AI/ML), in tempo reale o nel lungo periodo. Per prevedere il momento ottimale in cui far partire il termostato prima che l'utente finale arrivi a casa, il sistema IoT può connettersi all'API Google Maps per esaminare i modelli del traffico locale in tempo reale e utilizzare i dati dell'automobile acquisiti nel lungo periodo per conoscere le abitudini di viaggio dell'utente. I dati IoT acquisiti da ogni utilizzatore possono essere utilizzati dalle aziende dei servizi, per iniziative di ottimizzazione su più larga scala.

3.1.2 L'evoluzione dell'Iot nelle aziende

Nell'ultimo periodo le aziende di tutti i settori stanno guardando alla crescita dell'Internet of Things (IoT) come un insieme di infinità opportunità e come un mercato che non vogliono perdere. Un'analisi di McKinsey & Company stima il valore economico totale delle tecnologie IoT in un intervallo tra i 4 e i 11,1 trilioni di dollari l'anno entro il 2025.[32] Per superare i grandi cambiamenti della trasformazione IoT, gli sviluppatori di applicazioni IoT e le aziende devono comprendere che un time-to-market più rapido è fondamentale per il successo nel mercato IoT. A tale scopo, è necessario consentire alle piattaforme di sviluppo IoT di eseguire la maggior parte del lavoro più oneroso. Questo è attualmente un obiettivo importante per le grandi aziende IoT, e può effettivamente svolgere un ruolo molto importante nello sviluppo di applicazioni e servizi IoT scalabili, portando a tempi di vendita più rapidi dovuti alla riduzione dei costi e dei tempi di consegna. Per essere alla pari con il mercato, però, bisogna capire che l'attuale struttura aziendale è davvero il più grande ostacolo al successo nell'IoT. Ciò di cui ha bisogno un'azienda tradizionale nel complesso percorso di cambiamenti nell'IoT, è una cultura e un ambiente solidi che enfatizzino la collaborazione e l'efficienza,senza una netta distinzione tra i reparti e ben lontana dal vecchio approccio. Considerando l'alta posta in gioco, se c'è una cosa che i primi utilizzatori hanno imparato sulla natura del mercato IoT, è che il più grande ostacolo al successo sta nel modo in cui l'azienda moderna è strutturata in termini di sviluppo e operazioni. Quando si ha a che fare con dispositivi

connessi e grandi blocchi di dati sparsi in più, il modo in cui le persone, i processi e gli strumenti cooperano insieme giocherà un ruolo decisivo nel determinare se la propria soluzione è pronta. L'idea DevOps è recentemente emersa come una delle idee più praticabili per portare rapidi tempi di risposta, velocità, agilità e scalabilità nello sviluppo di soluzioni IoT. Fondamentalmente, i dispositivi connessi portano con sé la necessità di una distribuzione continua del software, per la quale le attuali aziende sono scarsamente attrezzata per affrontare le sfide che emergono durante un progetto in corso.

3.1.3 La connessione tra IoT e DevOps

DevOps,come già visto, è l'automazione completa di Agile, attraverso processi e strumenti che eliminano la latenza obsoleta dallo sviluppo dell'applicazione. Le trasformazioni basate su DevOps sono sistematiche sia per gli ISV (Independent Software Vendors :fornitori di software indipendenti) che per le imprese. Le persone del settore IT sanno che è molto importante sfruttare al meglio i big data, il cloud computing e l'IoT (Internet of Things). Alcune persone potrebbero trovare un po' difficile capire le connessioni. Ad esempio, non si troveranno risposte soddisfacenti dai gestori di sistemi IoT se gli verrà chiesta la migliore implementazione di DevOps. Non è molto difficile capire il perchè. Innanzitutto, l'IoT si basa su test di integrazione automatizzata, test di sicurezza e distribuzione. I sistemi DevOps richiedono il controllo sulla sicurezza delle informazioni emesse dai sensori. Questi sensori diventano dei punti d'accesso e possono essere facilmente compromessi, ad esempio qualcuno potrebbe piratare un termostato smart e accedere ad altri dispositivi connessi in rete, copiandone o danneggiandone i dati o il funzionamento. Gli strumenti di test di sicurezza di DevOps dovrebbero,attraverso l'utilizzo dell'IoT, assicurarsi che l'esposizione agli attacchi sia ridotta.

3.1.4 DevOps nell'IT rispetto a DevOps nell'IoT

Esistono però alcune differenze fondamentali nel modo in cui DevOps viene applicato per le soluzioni IoT, rispetto alla semplice progettazione di software. Nel settore IT, l'applicazione di DevOps è abbastanza semplice,come mostrato nella figura sottostante.

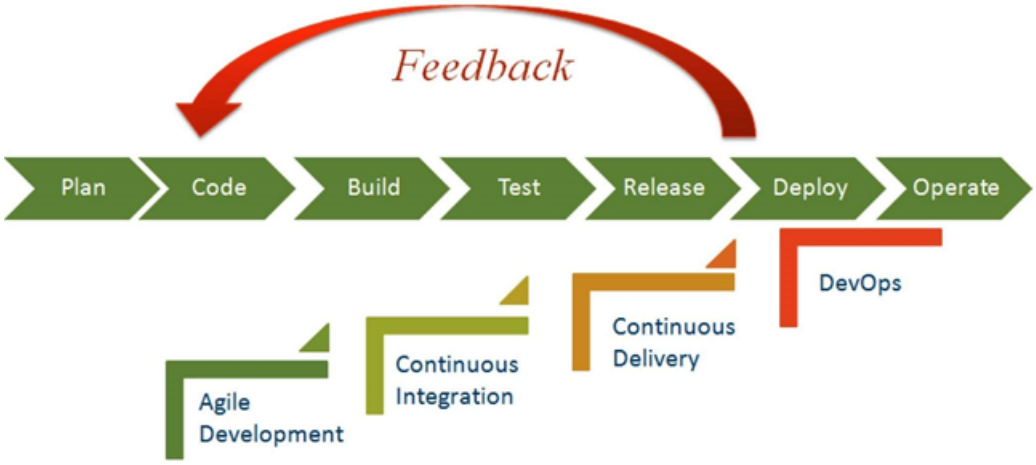

Figura 3.1: DevOps lifecycle in IT. [51]

Tuttavia, quando si ha a che fare con dispositivi connessi, sensori, protocolli unificati e gateway, la complessità diventa maggiore. Ciò si manifesta nei seguenti modi:

- **Pipeline di sviluppo complesse e frammentate**: nell'IoT, l'implementazione di DevOps è uguale a quella del software IT ma spezzettata. Le diverse pipeline di sviluppo, ad esempio: firmware, codice lato server, app mobile, app desktop; aggiungono un delivery a più endpoint. Oltre alla progettazione/aggiornamenti hardware si aggiunge un'ulteriore complessità. La composizione del team per ogni pipeline richiede competenze diverse e segue processi diversi. I team di sviluppo potrebbero essere in grado di fornire un codice di qualità per una determinata pipeline, ma è difficile integrare tutte le pipeline per garantire che i test dei prodotti end-to-end siano di qualità. Ciò richiede che i processi siano migliorati per supportare l'integrazione del codice con i normali test di verifica che assicurano che il codice sia sempre pronto per il deploy e sia disponibile per le correzioni/aggiornamenti di emergenza quando necessario.

- **Dispositivi legacy e soluzioni di supporto**. I prodotti, insieme all'infrastruttura cloud introducono i requisiti per la gestione, l'aggiornamento e la manutenzione dei dispositivi esistenti, aumentando nuovamente le variazioni e le complessità in DevOps. È necessario eseguire test rigorosi nei flussi di lavoro end-to-end. Ad esempio, un nuovo termostato lanciato sul mercato con funzionalità di autoap-

prendimento e con la funzionalità di raccolta/analisi dei dati tramite cloud sarebbe sicuramente la scelta preferita dai clienti finali; ma questo può rappresentare sfide indefinite per il progettista e per i team di test coinvolti.

- **Ambiente come codice (quando l'infrastruttura come codice non è sufficiente!).** Lo spazio di progettazione del prodotto comporta ulteriori sfide per la virtualizzazione di più dispositivi associati insieme all'infrastruttura server. Il concetto di "*infrastruttura come codice*" (IAC) deve essere esteso per offrire un "*Ambiente come codice*" completo. All'interno dei cloud privati e pubblici, le organizzazioni si stanno spostando dai modelli di spese in conto capitale (CAPEX) a spese operative (OPEX), pagando solo l'effettivo consumo. La virtualizzazione è necessaria per simulare sistemi legacy e nuovi sensori per test di Costumer Premise Equipment (CPE:è qualsiasi dispositivo situato presso un client collegato al circuito di telecomunicazione) che vanno integrati con l'analisi dei dati cloud.

- **Diverse pipeline di prodotto**: in IoT, ci sono maggiori variazioni nella consegna dei prodotto per più clienti (sia in termini di personalizzazione e miglioramento) così come le segmentazioni (basse, medie e alte). Questo implica diversi ambienti di produzione che possono essere difficili da riprodurre e mantenere nelle fasi di sviluppo.

- **Frequenza di rilascio**: I diversi componenti tra cui firmware, app Web, app per dispositivi mobili e app per PC, presentano cadenze di rilascio diverse, rendendo difficile un piano di rilascio. Lo sviluppo di più funzionalità in versioni diverse diventa difficile a meno che non venga seguito un processo ben definito, per l'integrazione continua e regolare del codice.

Inoltre, gli sviluppatori IoT possono sfruttare qualsiasi cosa una buona parte dell'implementazione di DevOps in aree che comprendono integrazione, test, implementazione e monitoraggio continui. Esistono già molti strumenti commerciali disponibili sul mercato per l'integrazione continua. Per l'IoT, ci aiuta a offrire una grande assistenza, ma richiederebbe alcune modifiche e miglioramenti. Ad esempio, l'integrazione continua nell'IoT richiederebbe l'integrazione di pipeline di sviluppo multiple e frammentate come dispositivi, Web e dispositivi mobili; oltre a preparare build principali e personalizzate. Ciò

include l'automazione della configurazione dell'ambiente, la revisione del codice, i test di collaudo(regression test) e il reporting. I test continui in IoT possono essere più efficaci con la virtualizzazione dei sensori e la simulazione di dispositivi fisici (vCloud, VSphere ecc.). Uno strumento CI (Jenkins, Bamboo ecc.) può essere sfruttato per integrare il framework di automazione. Quindi tutti questi varie caratteristiche, strumenti e processi devono fondersi insieme per fornire DevOps in IoT. Un'altra preoccupazione per le aziende è il modo in cui gli sviluppatori in IoT dovrebbero bilanciare vantaggi come aggiornamenti frequenti con le funzionalità di aggiornamento dei dispositivi remoti. Per risolvere questo problema, si deve notare che la frequenza dei rilascio in IoT differisce dalle tradizionali distribuzioni di software. La frequenza dei rilasci dell'IoT differisce a causa dei componenti. Ad esempio, un aggiornamento di una patch di un'applicazione web può essere un ciclo di rilascio mensile mentre un rilascio di un aggiornamento del dispositivo può variare da tre a sei mesi. Quindi i test continui devono evolversi per ideare un modo più ottimizzato di testare la soluzione in modo esaustivo. I test IoT sono costosi perché l'infrastruttura è enorme per dare una copertura esaustiva dei test.

3.1.4.1 Problemi di sicurezza e memoria dei dispositivi integrati

Come già visto precedentemente, bisogna analizzare come funziona DevOps per IoT all'interno di solide politiche di sicurezza come l'autenticazione, l'aggiornamento sicuro "*over-the-air*", ecc. . Poiché vi sono più endpoint di distribuzione nell'IoT, la sicurezza deve essere considerata da più aspetti. Esistono fondamentalmente tre tipi di rischi: **data at rest** (dati inattivi che vengono archiviati fisicamente in forma digitale), **data motion** (dati in movimento,cioè che dati che si spostano attraverso qualsiasi tipo di rete) e **dati nel cloud**. Esiste un rischio computazionale all'interno dei dispositivi o nel cloud. Inoltre ci sono rischi di manomissione dell'hardware nei dispositivi fisici. Gli attacchi DDoS del 21 ottobre 2016 (ora noti semplicemente come attacchi 10/21) utilizzando una rete di oggetti di uso quotidiano (fotocamere, stampanti e così via) compromessi avevano paralizzato il server DNS **Dyn**[1], danneggiando efficacemente il traffico in siti di alto spessore come Twitter, Amazon e Netflix.[33] Chiaramente, le aziende che lavorano su con IoT dovranno trovare delle soluzioni per migliorare l'affidabilità dei loro prodotti

[1] `https://dyn.com/`

finali. I problemi di sicurezza nell'ambiente di sviluppo IoT possono essere notevolmente mitigati inserendo una migliore visibilità e un maggiore controllo sui dispositivi, dando a ciascun dispositivo una propria identità legata all'ambiente aziendale o all'utente finale.

3.1.5 Importanza di DevOps nell'IoT

Lo sviluppo dell'IoT sta portando un cambiamento rivoluzionario nel settore IT e le persone hanno iniziato a trarre vantaggio da questo. Il ruolo DevOps nell'IoT è di grande importanza. Qui verranno evidenziati i sei motivi che enfatizzano i ruoli DevOps in IoT.

- **L'effetto diffusione**: Se l'ultima versione del software viene aggiornata sul server in modo continuo o regolare, i sistemi connessi a quel particolare server o, collegati a quel software, necessitano di un aggiornamento frequente.
- **L'effetto Evoluzione**: l'approccio DevOps verrà applicato a qualsiasi contesto, come la fase successiva dell'evoluzione dallo sviluppo Agile alla distribuzione continua. Le aziende che lo utilizzano tramite team integrati e interfunzionali dovrebbero godere di maggiori vantaggi rispetto ad altri.
- **Qualsiasi cosa è Software-Defined**: la funzionalità del sistema è definita dal programma o dal software che è installato sull'hardware. Pertanto, quando una funzionalità deve essere modificata, è necessario aggiornare il software invece di apportare modifiche manuali (elettroniche e/o meccaniche).
- **L'infrastruttura in atto**: nell'era moderna dell'innovazione tecnologica, le persone collegano i sistemi al World Wide Web e al cloud. Ecco perché è possibile installare e aggiornare il software regolarmente in più dispositivi in remoto.
- **Efficienza dei costi e maggiore produttività**: un processo che aumenta la velocità del ciclo di sviluppo senza compromettere la qualità attraverso un modo più intelligente, veloce, migliore e con la possibilità di ridurre i costi di produzione.
- **Nuovi flussi di entrate e modelli di business**: lo scopo principale dell'IoT è quello di creare maggiori possibilità per diversi modelli di business. DevOps ha

procurato un modo per fornire costantemente un nuovo aggiornamento software per commerciare il servizio offerto piuttosto che vendere il prodotto una tantum e già completo di funzionalità.

DevOps ha dimostrato il suo valore a livello aziendale e ora sta crescendo verso gli **Edge Device** ("*dispositivo perimetrale*": sono dispositivo che forniscono un punto di accesso alle reti core aziendali o un provider di servizi core networs. Ad esempio router, switch di routing, dispositivi di accesso integrati, multiplexer e una varietà di dispositivi di accesso alla rete metropolitana e alla rete geografica) e i gateway per affrontare tutte le sfide di sviluppo e per gestire i sistemi IoT.

3.1.6 Guardando al futuro

Il pensiero comune delle imprese che lavorano ai progetti IoT è che in un dato momento, semplicemente non ci saranno abbastanza risorse di talento con le giuste capacità e competenze per sincronizzarsi con le tempistiche dei progetti. Garantire la disponibilità tempestiva del personale tecnico e delle competenze richieste è una caratteristica importante e desiderabile. Inoltre, si deve tener conto anche dell'eventuale rifiuto alle nuove tecnologie, pratiche e processi da parte del personale aziendale. A seconda del requisito, le competenze esterne possono essere in linea con i fornitori di servizi che colmano il divario apportando ulteriori competenze.

3.2 DevOps e il mobile: introduzione

Con il passare del tempo, il mondo dello sviluppo mobile si è altamente avanzato, ciò significa che uno sviluppatore deve rimanere costantemente aggiornato così da poter essere anche altamente competitivo. Questo è il motivo dietro l'aumento del numero di fornitori di servizi mobile che fatto uso dei processi DevOps mobile. Oggigiorno,infatti si ha bisogno di abbracciare le moderne tecnologie, gli strumenti, e le tendenze di sviluppo del software Agile per le applicazioni mobili che si rivolgono a piattaforme diverse. Questo quindi aiuta a capire il processo DevOps applicato al contesto mobile.

3.3 Mobile Computing: definizione

Ma cosa si intede per mobile? Il Mobile in generale è un contesto molto ampio, si partirà con un concetto fondamentale, il Mobile Computing. Il **Mobile Computing** è una tecnologia che permette la trasmissioni di dati,voce e video via computer o qualsiasi dispositivo wirless senza che sia per forza connesso ad un collegamento fisico fisso e anzi, sia possibile utilizzarlo "*in movimento*". Vi sono tre aspetti principali che verranno analizzati:

1. Mobile communication.
2. Mobile hardware.
3. Mobile software.

3.3.1 Mobile communication

In questo caso per *Mobile communication* si intende l'infrastruttura messa in piedi per garantire che la comunicazione sia continua e affidabile. Questo include i device con al loro interno protocolli e servizi, ma include anche i portali necessari per facilitare e supportare lo stato dei servizi. Anche il formato dei dati viene definito in questa fase. Questo garantisce che non vi siano collisioni con altri sistemi esistenti che offrono lo stesso servizio. Poiché i media non sono guidati/illimitati, l'infrastruttura di overlaying è sostanzialmente orientata alle onde radio, cioè i segnali vengono trasportati nell'aria ai dispositivi previsti che sono in grado di ricevere e inviare tipi di segnali simili.

3.3.2 Mobile hardware

L'hardware mobile include dispositivi mobili o componenti di dispositivi che ricevono o accedono al servizio di mobilità. Si passerà dai laptop portatili, smartphone, tablet PC, fino ai Personal Digital Assistants. Questi tipi di dispositivi avranno un supporto recettoriale in grado di rilevare e ricevere segnali. Questi dispositivi sono configurati per funzionare in full-duplex, per cui sono in grado di inviare e ricevere segnali contemporaneamente. Infatti non hanno bisogno di aspettare che un device abbia finito di

comunicare con un altro device per avviare la comunicazione. I dispositivi sopra menzionati utilizzano una rete esistente e stabilita per operare. Nella maggior parte dei casi, si tratta di una rete wireless.

3.3.3 Mobile software

Il software mobile è l'effettivo programma che gira sull'hardware mobile. Si occupa delle caratteristiche e dei requisiti delle applicazioni mobili. Questo è il motore del dispositivo mobile. In altri termini, è il sistema operativo dell'appliance, il componente essenziale che gestisce il dispositivo mobile. Poiché la portabilità è il fattore principale, questo tipo di elaborazione garantisce che gli utenti non siano collegati o bloccati a una singola posizione fisica, ma siano in grado di operare da qualsiasi luogo, incorporando tutti gli aspetti delle comunicazioni wireless.

3.4 Ubiquitous and Pervasive Computing

La mobilità introduce una serie di sfide per i sistemi distribuiti, tra cui la necessità di gestire la connettività variabile,la disconnessione, e la necessità di mantenere il funzionamento nonostante la mobilità dei dispositivi. L'Internet of Things (IoT), il mobile computing e la rivoluzione DevOps: sono tutti discendenti del movimento pervasive o ubiquitous computing. Il calcolo diventa uno strumento creativo per aiutare le persone a risolvere i problemi. Ma cosa si intende per Pervasive Computing? Prima di parlare dei sistemi pervasivi, verrà introdotta la Ubiquitous Computing e cosa ha portato a tutte queste iterazioni tra uomo e macchina.

3.4.1 Nascita della human-computer interaction

Negli ultimi anni si è assistito ad una sempre crescente diffusione di dispositivi personali mobili (smartphones,palmari, tablets). Tale diffusione e accettazione è dovuta principalmente a:

- maggiore potenza di calcolo.
- Aggiunta di sensori (accelerometro, giroscopio, gps, ecc.).

- Iterazione intuitiva (touch screen, ecc.).
- Molte connessioni wireless in contemporanea (Wifi, bluetooth, RFID,ecc).

I dispositivi mobili sono diventati parte integrante della vita quotidiana, permettendo operazioni di qualsiasi tipo e in qualsiasi momento, tra queste abbiamo: l'utilizzo di molti servizi on-demand e l'utilizzo come iterazione uomo-ambiente (realtà aumentata, sensori). Da questa idea deriva il termine utilizzato per la prima volta nel 1991, *Ubiquitous Computing* da Mark Weiser.

3.4.2 Ubiquitous Computing

L'idea di Mark Weiser era quella che:

"Le tecnologie più potenti sono quelle che sono diventate invisibili. Quelle che legate insieme, formano il tessuto della nostra vita quotidiana al punto di diventare una parte da cui non riusciamo più a farne a meno."[34]

In generale le caratteristiche della Ubiquitous Computing sono opposte al paradigma del "*desktop (computer)*" in cui l'utente aziona consciamente una singola apparecchiatura per uno scopo specifico, chi "utilizza" la ubiquitous computing aziona diversi sistemi e apparecchiature di calcolo simultaneamente, nel corso di normali attività, e può anche non essere cosciente del fatto che questi macchinari stiano compiendo delle operazioni. Infatti i dispositivi hanno la capacità di: ottenere informazioni dall'ambiente in cui sono e di adattare il loro comportamento selezionando diverse modalità di elaborazione. Il termine "*onnipresente*" (Ubiquitous) ha lo scopo di suggerire che i piccoli dispositivi informatici alla fine diventeranno così presenti negli oggetti di tutti i giorni che quasi non verranno più notati; cioè, il loro comportamento computazionale sarà legato in modo trasparente e intimamente alla loro funzione fisica.

3.4.3 Pervasive Computing

I sistemi pervasivi possono essere considerati come la realizzazione attuale del paradigma Ubiquitous Computing, che diventa *Pervasive Computing*. Il calcolo non è incentrato

su un singolo dispositivo mobile, ma è ovunque; i dispositivi sono interconnessi e cooperanti tra di loro al fine di fornire servizi ai loro utenti. Così come il calcolo pervasivo, i dispositivi IoT connessi comunicano e forniscono notifiche sull'utilizzo. Nell'informatica pervasiva la potenza di calcolo è ampiamente dispersa negli oggetti quotidiani. L'IoT è in procinto di fornire questa visione e trasformare oggetti comuni in dispositivi connessi, ma per ora richiede una grande quantità di configurazione e interazione uomo-computer, cosa che l'onnipresente elaborazione di Weiser non fa. L'IoT può utilizzare reti di sensori wireless. Queste reti raccolgono dati dai singoli sensori prima di inoltrarli al server IoT. Un esempio di utilizzo è quando si raccolgono dati sulla quantità di acqua che fuoriesce dalla rete idrica di una città, qui può essere utile raccogliere i prima dati tramite la rete di sensori wireless e poi elaborarli. In altri casi, ad esempio, nei dispositivi indossabili, come l'Apple Watch, la raccolta e l'elaborazione dei dati è inviata direttamente a un server centralizzato. La presenza in qualsiasi luogo di diversi computer diventa utile solo quando essi possono comunicare tra loro. Ad esempio, può essere conveniente per gli utenti controllare la propria lavatrice dal proprio telefono o da un dispositivo presente in casa. Allo stesso modo, la lavatrice potrebbe avvisare l'utente tramite una notifica sul cellulare al termine del lavaggio. La pervasive e il mobile computing si sovrappongono, poiché l'utente mobile può in linea di principio beneficiare di "*computer ovunque*" ma, sono distinti, in generale la pervasive computing potrebbe essere utile agli utenti mentre rimangono in un unico ambiente come la propria abitazione o un ospedale. Allo stesso modo, il mobile computing presenta vantaggi anche se coinvolge solo computer e dispositivi convenzionali come laptop e stampanti. La Figura [57] sottostante mostra un utente che sta visitando un server web(host). La figura mostra l'intranet dell'utente e l'intranet dell'host nel sito che l'utente sta visitando. Entrambe le intranet sono collegate al resto di Internet.

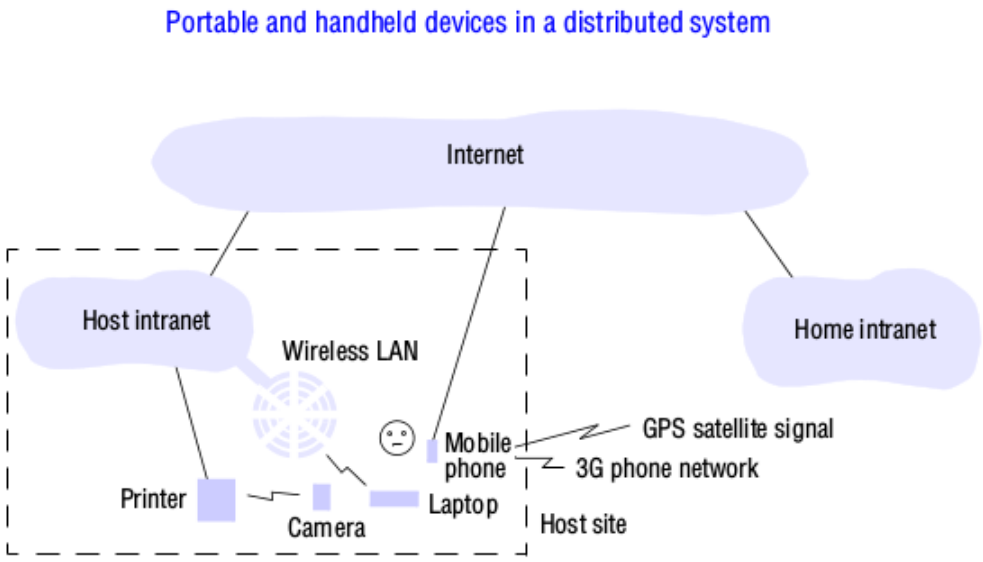

Figura 3.2: Mobile And Ubitous Computing. [57]

L'utente ha accesso a tre forme di connessioni wireless. Il proprio laptop che si può connettere alla wireless LAN dell'host(interna); questa rete fornisce una copertura di alcune centinaia di metri (per esempio un piano di un edificio). Si collega al resto della "Host intranet" tramite un gateway o un punto di accesso. L'utente ha anche un telefono cellulare (mobile phone), che è collegato a Internet. Il telefono consente l'accesso al Web, ad altri servizi Internet, e può anche fornire informazioni sulla posizione tramite la funzionalità GPS integrata. Infine, l'utente possiede una fotocamera digitale, che può condividere all'interno della propria rete wireless (con una portata fino a circa 10 m) con un altro dispositivo come ad esempio una stampante. Con un'adeguata infrastruttura di sistema, l'utente può eseguire alcune semplici attività nel sito host utilizzando i dispositivi a sua disposizione. Durante la navigazione, se ad esempio l'utente sta facendo una ricerca in un determinato negozio, può recuperare alcune informazioni direttamente dal server Web utilizzando lo smartphone e può anche utilizzare il GPS integrato e il software di ricerca integrato per calcolare il percorso per raggiungere l'eventuale ubicazione. Durante ciò, l'utente può inviare alla propria stampante, opportunamente abilitata, una fotografia prendendola direttamente dalla fotocamera digitale. Ciò richiede solo il colle-

gamento wireless tra la fotocamera e la stampante. In linea di principio, si può inviare un documento dal proprio laptop alla stampante, utilizzando i collegamenti LAN wireless o tramite rete Ethernet cablata.

3.5 DevOps Mobile perchè viene scelto

Ritornando al mobile, vi è un principio fondamentale che si è creato ultimamente: il mobile è diventato molto importante. La crescente concorrenza con le applicazioni per dispositivi mobili e l'acquisizione da parte degli utenti ha spinto le aziende ad agire rapidamente con il suo sviluppo e le sue operazioni. Ottenere l'applicazione "giusta" in primo luogo e lavorare correttamente con una miriade di dispositivi diversi è un'attività che deve essere considerata prima che l'applicazione venga pubblicata. In questa cultura DevOps, il processo e gli strumenti hanno un ruolo significativo. Verranno ripresi i principici DevOps precedentemente elencati e verrà specificato cosa significa applicarli ai dispositivi mobili. I principi DevOps vengono adottati ampiamente in una varietà di tipologie di società di software che costruiscono ancora più diversi tipi di prodotti. Ma che cosa significa effettivamente DevOps e quali sono le implicazioni di questo approccio per lo sviluppo e il test di applicazioni per dispositivi mobili? *"Pratica che enfatizza la collaborazione e la comunicazione sia degli sviluppatori di software che di altri professionisti IT, automatizzando al contempo il processo di distribuzione del software e i cambiamenti dell'infrastruttura."*[35] Cosa spinge le aziende verso questo tipo di approccio DevOps? Prima di tutto, ci sono sempre requisiti aziendali; è la domanda che guida l'adozione di qualsiasi tipo di processo o cambiamento culturale all'interno delle aziende. Con l'approccio DevOps il vantaggio apparente per le aziende è l'adozione completa della metodologia Agile per eseguire le operazioni più rapidamente e con una mentalità mirata in materia di supporto, operazioni e servizi. Quando queste due mentalità(Agile e DevOps) sono insieme le aziende vogliono ottenere entrambi i risultati nel modo più efficiente possibile. Nonostante la stima sia stata impostata su un valore elevato per il successo delle applicazione per dispositivi mobili, ci sono molte e diverse caratteristiche che devono essere considerate quando si adotta un approccio DevOps per lo sviluppo e il test di applicazione per dispositivi mobili. Questi requisiti sono guidati

dagli sviluppatori, dai proprietari di aziende e, soprattutto, dagli utenti finali.

3.5.1 DevOps Mobile cos'è

Il pensiero DevOps è diventato una parte consolidata di aziende e team di sviluppo che si affidano a metodologie Agili. La combinazione di metodologie Agili e DevOps è essenziale per fornire una integrazione,una distribuzione continua e per velocizzare: l'integrazione, la distribuzione e i test, oltre che a migliorare la qualità dell'applicazione. Ottenere l'applicazione mobile istantaneamente dal sistema di compilazione su un dispositivo reale in cui l'applicazione può essere accuratamente testata migliora la velocità complessiva del processo e rende la correzione del bug più facile e veloce. DevOps in mobile significa incorporare il processo di integrazione e test continuo congiuntamente con il delivery,il deployment di tutto il sistema in operazioni continue. Gli sviluppatori, e lo sviluppo in generale, si concentra sulla costruzione del prodotto effettivo, facendo affidamento ai propri strumenti per il rilevamento di bug e/o malfunzionamenti. La garanzia della qualità (QA) viene fatta da uno stakeholder che è interessato (e responsabile) nel testare tali regressioni, nelle prestazioni del prodotto e nell'automatizzare il maggior numero possibile di flussi di test. Il team operazionale mantiene il build, l'integrazione, il deployment e il delivery continuo e si occupa di tali rilasci. Quando si combinano tutte queste tre diverse funzioni in un unico approccio olistico, si può usare Mobile DevOps come definizione, come mostrato nella figura [58].

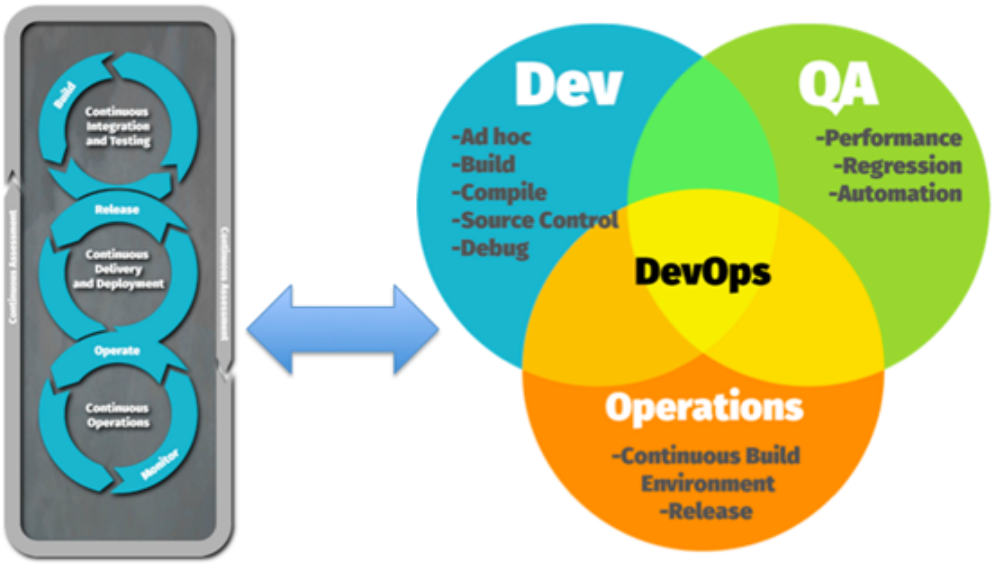

Figura 3.3: SDLC lifecycle. [58]

3.5.2 DevOps Mobile nello specifico

Come già visto precedentemente, l'approccio DevOps migliora lo sviluppo del software concentrandosi sulla collaborazione tra: le varie parti interessate,i membri del progetto, i responsabili del prodotto, gli sviluppatori e il personale operativo; allineando i progetti con l'obiettivo aziendale. In passato, le principali società di sviluppo mobile, incluse Apple con iOS e Google con Android, usavano applicazioni native per sviluppare prodotti ibridi. Tuttavia,con l'avanzare della tecnologia, si è reso necessario soddisfare gli obbiettivi di business dei cliente sfruttando i processi di sviluppo, le persone e le ultime tecnologie, facendo guadagnare a DevOps popolarità tra i team di sviluppo di applicazioni per dispositivi mobili, tuttavia però, come sottolineato da Gartner: "*solo il 42% di coloro che hanno implementato DevOps hanno indicato che DevOps viene utilizzato per supportare lo sviluppo di applicazioni per dispositivi mobili*".[36] Sebbene i principi di base di DevOps siano gli stessi per le applicazioni aziendali e per dispositivi mobili, la mobilità introduce nuove sfide e requisiti che devono essere presi in considerazione. Ecco alcuni esempi:

1. La continua frammentazione e la rapida proliferazione di dispositivi mobile, stru-

menti di gestione e sistemi operativi influiscono negativamente sulla capacità di DevOps di tenere il passo con le nuove versioni delle applicazioni per dispositivi mobili.

2. Le applicazioni per dispositivi mobile generano modifiche ai sistemi aziendali back-end che richiedono una maggiore collaborazione tra i team di sviluppo.

3. I progetti di sviluppo mobile hanno in genere linee temporali estremamente ristrette. A causa della forte motivazione aziendale a fornire rapidamente applicazioni mobili sul mercato, i progetti di sviluppo mobile hanno linee temporali estremamente brevi. La pressione per distribuire rapidamente applicazioni per dispositivi mobile comporta l'adozione di metodi di sviluppo Agili per progetti per dispositivi mobile.

4. App store. Niente è più dannoso per un'azienda software di una bassa reputazione (ad esempio una stella), in particolare quando tale valutazione viene trasmessa tramite un App store. L'insoddisfazione degli utenti per le applicazioni mobile può diventare rapidamente e visibilmente di dominio pubblico, indipendentemente dal fatto che l'applicazione sia a pagamento o gratuita. Mentre i reclami relativi a problemi nei siti Web vengono comunicati a un supporto tecnico, e quindi di dominio privato, le lamentele sulle applicazioni per dispositivi mobili vengono trasmesse tramite l'App store per consentire a tutti di vederle. Le applicazioni per dispositivi mobile devono essere sottoposte a test funzionali, di usabilità e di prestazioni per garantirne la qualità.

In che modo DevOps può soddisfare i requisiti per le applicazioni per dispositivi mobile (Mobile DevOps)?

1. **Accelerare i tempi di sviluppo**. La velocità è la chiave nell'ambiente aziendale competitivo di oggi. Che si tratti di un'app aziendale o di un'app consumer, la metodologia Agile di DevOps aiuta a ridurre il time to market. Gli aspetti chiave dello sviluppo sono stati automatizzati in questo ecosistema, garantendo così meno errori, una distribuzione più rapida e una facile integrazione dei vari sistemi.

2. **Distruggere i sistemi silos**(Un silos di informazioni, o un gruppo di tali silos è un sistema di gestione in cui un sistema di informazione o sottosistema non è in grado di operare con altri sistemi che sono o dovrebbero essere correlati). Mobile DevOps è una pratica per portare le diverse discipline coinvolte nello sviluppo, test, rilascio e gestione del software all'interno di un unico team che lavora a stretto contatto. Oltre a ciò, consiste anche nell'eliminare i silos e semplificare il processo tra le fasi di compilazione e produzione (inclusi test e distribuzione). Riunendo sviluppatori (Dev) e operazioni (Ops), il team è in grado di fornire costantemente il proprio prodotto in base al feedback continuo e all'iterazione.

3. **Integrazione continua**. L'adozione di DevOps per le applicazioni inizia dalla creazione di codice sorgente che include l'uso dell'integrazione continua. Ogni rilascio deve passare prima tramite un test reale del dispositivo. Questa configurazione di test deve includere un numero elevato di dispositivi mobili reali utilizzati dal pubblico di destinazione. DevOps in questo contesto garantisce una versione e un ambiente stabile. Pur avendo un tempo limitato a portata di mano, non si possono commettere errori durante la creazione di applicazioni mobile. Pertanto, è necessario risolvere i problemi anche prima che la modifica o l'applicazione raggiunga gli utenti finali.

4. **Automatizzare il test delle applicazioni**. Il test delle applicazioni per dispositivi mobile è una delle fasi più importanti del processo di sviluppo. È anche uno dei più grandi colli di bottiglia dato che gli sviluppatori tradizionali tenderebbero a verificare manualmente gli script di test. Utilizzando DevOps, l'intero processo di test è automatizzato, risparmiando così tempo prezioso. Infatti, gli sviluppatori possono adottare la strategia testing-as-a-service che consente loro di testare l'intera applicazione o le funzionalità selezionate come e quando necessario.

I processi utilizzati in passato mancavano di un coordinamento efficace e potente tra i diversi settori di sviluppo software. Questo significava un tempo aggiuntivo di sviluppo, scappatoie e insoddisfazione del cliente, introducendo il bisogno di processi DevOps mobile.

3.5.3 Introduzione ai processi Mobile Devops

Un processo DevOps mobile, per essere definito tale, deve:

- portare differenti stakeholder nello sviluppo di applicazioni mobile. Questo include operazioni e team di sviluppo mobile.
- Distribuire facilmente le responsabilità tra il team di sviluppo.
- Migliorare il processo di integrazione degli obbiettivi aziendali con esperti: IT, operativi e di sviluppo.
- Migliorare lo sviluppo di applicazioni mobile e il loro rilascio.

Mobile DevOps aiuta anche gli sviluppatori a mettere insieme tutti i loro processi per favorire un flusso di lavoro continuo e per risolvere i problemi. Inoltre, aiuta a prendere decisioni cruciali e funzionanti per il successo dello sviluppo mobile.

3.5.4 I sei processi del Mobile Devops

DevOps prevede l'implementazione di 6 processi basilari, che sono:

1. *Continuous Planning(Pianificazione continua)*: la pianificazione continua prevedere il riunire insieme il team operaionale,i tester, gli sviluppatori e il business analysis in un'unica piattaforma. Questo è fatto su misura per assicurare un chiaro piano di rilascio.
2. *Continuous Integration(Integrazione continua)*: Un processo incentrato sull'impiego di pratiche prive di errori e sulla definizione di standard per il controllo delle versioni. Questo avviene rivolgendosi alla comunità degli sviluppatori mobile.
3. *Continuous Testing (Test continuo)*: Una parte integrante del mondo dello sviluppo software. È specificamente progettato per assicurare la qualità di un prodotto rilasciato ad un cliente di prim'ordine.

4. *Continuous Monitoring(Monitoraggio continuo)*: Un processo che assicura che l'applicazione funzioni come desiderata. Garantisce anche un ambiente di produzione stabile nonostante alcuni cambiamenti imprevisti che potrebbero sorgere nel processo.

5. *Continuous Delivery (Rilascio continuo)*: Una pratica secondo cui l'ambiente di produzione aggiorna il codice con quello corretto. Fornisce il codice in tempo reale.

6. *Continuous Development (Sviluppo continuo)*: Lo sviluppo continuo è un processo di consegna continua in cui ogni modifica è automatizzata e testata. Una volta testato con successo, viene automaticamente distribuito o consegnato nell'ambiente di produzione.

Grazie a questi processi, l'adozione di un processo DevOps mobile è una svolta nel mondo dello sviluppo mobile. Garantisce un funzionamento regolare consentendo a tutti i membri del team di eseguire le attività assegnate nel minor tempo possibile. L'intero processo di DevOps non è però sempre facile da applicare. È necessario prendere decisioni aziendali ed adottare tecniche per la corretta implementazione. Così facendo, la propria azienda o organizzazione ha maggiori possibilità di ottenere i benefici derivati dall'approccio DevOps. I benefici risultanti sono:

- viene aumentata l'efficienza del rilascio: viene rilasciato software di qualità e ciò avviene tramite un rilascio "*semplice*".

- Una maggiore soddisfazione da parte del cliente finale, tramite un eccellente supporto ed esperienza.

- Trasparenza nei processi di produzione.

- Riduzione dei costi dello sviluppo di un'applicazione.

- Maggiore ritorno sugli investimenti.

- Maggiore fidelizzazione e coinvolgimento dei dipendenti.

- Offre più tempo per l'innovazione di qualità, lo sviluppo mobile e una maggiore efficacia.
- Adozione di processi DevOps Mobile.

L'adozione dei processi DevOps Mobile comporta però delle regole cruciali che sono raggruppate in tre procedure principali che non vanno mai ignorate, e sono:

1. **Continuous Integration e Continuous Delivery**.All'interno di questi passaggi è necessario:
   (a) Assicurarsi che tutti gli scripts, documenti, codice e file di testo siano tracciabili. Ogni codice deve essere sincronizzato.
   (b) Mantenere sempre una build unica per tutte le varianti in base alle piattaforme di destinazione.
   (c) Gestire i propri script assegnando le versioni giuste e assicurarsi che le build vengano riprodotte.
2. **Test e Monitoraggio**
   (a) Testare la propria applicazione utilizzando gli strumenti automatici giusti ed efficaci.
   (b) Utilizzare l'intero ecosistema per le proprie istanze virtuali. Ciò aiuta a testare rapidamente e ridurre i costi di sviluppo di un'applicazione; oltre a ridurre le spese per le risorse hardware.
   (c) Monitorare continuamente la propria applicazione per evitare insoddisfazioni del cliente. Ciò può essere ottenuto incorporando un SDK di terze parti nella propria applicazione.
3. **Qualità e rilascio**
   (a) Come sviluppatore, è indispensabile monitorare attentamente il feedback delle applicazioni. Aiuta a migliorare la propria applicazione ed ad aumentare le possibilità di successo.

3.5.4.1 Possibili sfide nell'adozione di un processo DevOps mobile

Mentre DevOps offre straordinari vantaggi, implementare il processo può essere impegnativo. Pertanto, è consigliabile identificare tutti i possibili problemi e ostacoli che si verificheranno. Alcune delle sfide più comuni includono:

1. Multi-piattaforme di destinazione tra cui le differenti versioni: del sistema operativo, dei dispositivi e le specifiche hardware.

2. Google Play Store per Android e App Store per iOS. Le applicazioni mobile non possono essere direttamente utilizzate/installate in un dispositivo mobile. Devono essere inserite e revisionate dai processi di un App store.

3. Allo stesso modo, il modello di distribuzione tramite push rappresenta una sfida perché spetta all'utente finale scegliere se aggiornare un'applicazione.

Test rigorosi nello sviluppo di applicazioni mobili sono allo stesso modo una sfida a causa degli utenti finali(tester client). Il cliente insoddisfatto danneggerà il marchio in diversi App Store. Ciò significa che bisogna testare attentamente l'usabilità di un'applicazione, le sue prestazioni e utilizzare le giuste tecniche di test.

Capitolo 4

DevOps applicato in Android

Tutte le pratiche precedentemente descritte possono essere applicate a qualsiasi ambiente mobile, ora però si andranno ad analizzarle ed ad applicarle nello specifico di Android.

4.1 Android e la sua architettura

Android é un sistema operativo sviluppato da Google e lanciato per la prima volta nel 2008. Android segue un modello di sviluppo Open Source, offre quindi all'utente una vasta libertà di utilizzo ed anche una grande flessibilità sopratutto per quanto riguarda la progettazione e il rilascio del software ad esso dedicato. È uno dei principali sistemi operativi al mondo, non solo per quanto riguarda l'ambito prettamente mobile, è presente infatti anche in tablet, computer, dispositivi indossabili, smart tv, ecc. . Android è costituito da un **kernel Linux** prima in versione 2.6, poi 3.x (da Android 4.0 in poi), e direttamente al suo interno sono inseriti i driver per il controllo dell'hardware del dispositivo (Wi-fi, Bluetooth, controllo dell'audio ecc.); sopra il kernel abbiamo gli HAL (hardware abstraction layer/strato di astrazione dell'hardware): forniscono interfacce standard che espongono le capacità hardware del dispositivo al framework API Java di livello superiore senza influirlo o modificarlo. L'HAL sono costituiti da più moduli, ognuno dei quali implementa un'interfaccia per un tipo specifico di componente hardware, come i moduli per la videocamera o il bluetooth. Quando un'API framework

effettua una chiamata per accedere all'hardware del dispositivo, il sistema Android carica il modulo della libreria per quel componente hardware.[37] Sopra gli HAL sono presenti le librerie fondamentali che sono state prese dal mondo Open Source, tra queste abbiamo **OpenGl** (per la grafica), **SQLite** (per la gestione dei dati), e **WebKit** (per la visualizzazione delle pagine web). L'architettura prevede poi una macchina virtuale e una libreria fondamentale che, insieme, costituiscono la piattaforma di sviluppo per le applicazioni Android. Questa macchina virtuale all'inizio era *Dalvik*, sostanzialmente una Java Virtual Machine, poi con il passare degli anni, precisamente dalla versione 5.0 di Android è stata sostituita con **ART** (Android RunTime). La differenza principale è nel tipo di compilazione utilizzata. Dalvik si basava su tecnologia **JIT** (just-in-time) mentre ART invece è basato su tecnologia **AOT** (ahead-of-time). La differenza è che con ART ci vorrà più tempo per installare una nuova applicazione che con Dalvik, e le applicazioni occuperanno più spazio nella memoria interna ma, allo stesso tempo, dal momento che l'applicazione sarà completamente compilata appena installata, i tempi di esecuzione saranno molto più rapidi riducendo anche l'utilizzo del processore (minore quantità complessiva di elaborazione), favorendo di fatto una maggiore durata della batteria. Prima invece con Dalvik, ogni app veniva compilata solo in parte e ogni volta che l'app veniva utilizzata, il software Dalvik eseguiva il codice e lo compilava. Nel penultimo strato dell'architettura è possibile trovare i gestori e le applicazioni di base del sistema. Ci sono gestori per le risorse (Resource Manager), per le applicazioni installate, per le telefonate, per le notifiche (Notification Manager) per il file system e altro ancora: tutti componenti di cui difficilmente si può fare a meno. Infine, sullo strato più alto dell'architettura, sono presenti gli applicativi destinati all'utente finale, molti, naturalmente, sono già inclusi nell'installazione di base; in questo livello sono presenti anche le applicazioni. L'architettura completa è visibile nella figura [59]

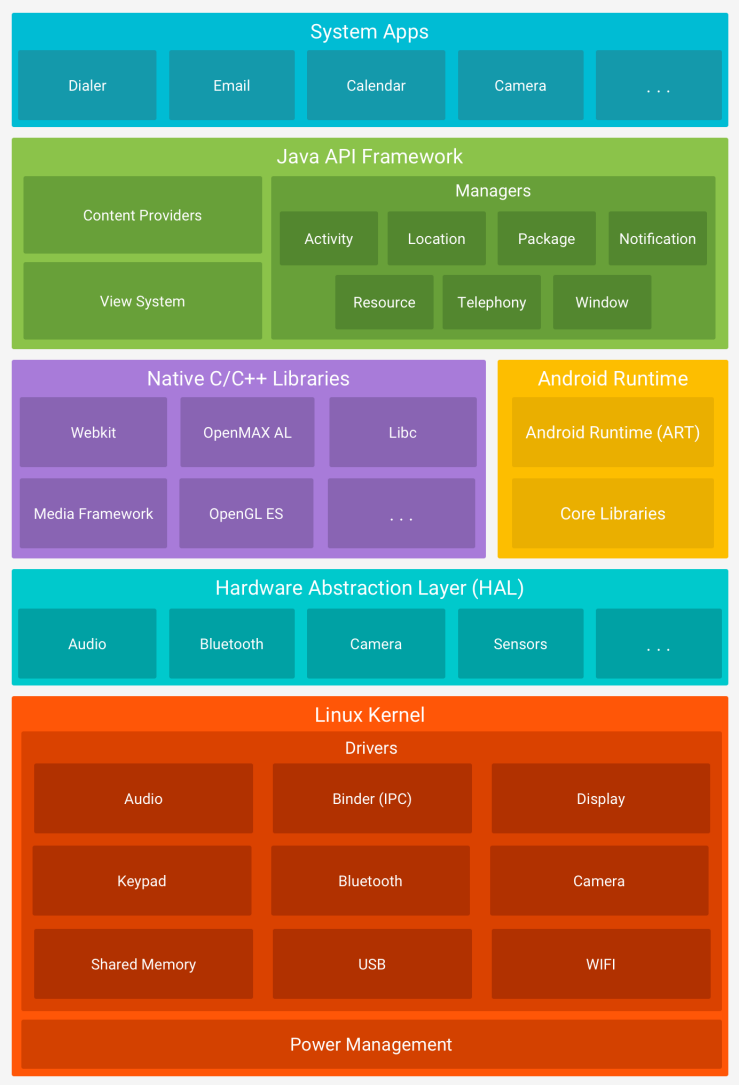

Figura 4.1: Architettura Android. [59]

4.1.1 Android Package

Un'applicazione per poter esistere deve essere rilasciata nello store o direttamente nel dispositivo in cui si vuole installarla, per farlo la nostra applicazione sarà in un formato detto **APK** (Android PacKage). In generale il file APK è un file dove al suo interno troviamo le seguenti cartelle:

- META-INF
- res

e i file:

- AndroidManifest.xml
- classes.dex
- resources.arsc

Il file avrà come estensione .apk (application/vnd.android.package-archive). Con il passare degli anni, questo tipo di file ha generato alcune problematiche, più che altro perchè era inizialmente concepito per contenere al suo interno tutte quelle librerie specifiche che erano incluse nel progetto ma che a loro volta potevano avere comportamenti diversi a seconda del device in cui venivano installate in quanto generiche e non specifiche, oltre a ciò, il fatto di includere queste librerie all'interno dell'apk aumentava anche la dimensione dell'Apk stesso. Oltre a ciò la dimensione di un'applicazione diventa uno dei tanti fattori che ne compromette l'usabilità in quanto molti utenti non hanno accesso ad una rete wi-fi ma dispongono solo di reti 3g e quindi dover scaricare un'applicazione di 100MB era proibitivo e molti preferivano non scaricarla; *in generale un'applicazione di 10 MB ha il 30% di probabilità in più di essere scaricata rispetto ad un'applicazione di 100MB.*[38] Dalla versione 5 di Android, precisamente dal 2018, è stato introdotto il concetto di Android App Bundle(AAB), un nuovo formato di caricamento di un'applicazione che include tutto il codice e le risorse compilate dell'app e, che contrariamente al precedente tipo, utilizza un modello di pubblicazione noto come Dynamic Delivery di Google Play, che crea e pubblica APK ottimizzati a seconda del dispositivo. Infatti rispetto agli APK, gli app bundle:

- hanno dimensioni più piccole.
- Possono utilizzare librerie native che prima non venivano comprese (dalla versione 6 in poi) archiviate nell'APL anzichè sul dispositivo dell'utente, che come già detto riducono le dimensioni sia su disco che, di conseguenza, download e tempo d'installazione dell'app.

- Offrono agli utenti le funzionalità e le configurazioni di cui hanno bisogno on demand, anziché durante l'installazione.
- Semplificano la gestione delle versioni e delle build, poiché non è necessario creare e pubblicare più APK.

Il nuovo modello di pubblicazione di app di Google Play, Dynamic Delivery, utilizza quindi il pacchetto di app per generare e pubblicare APK ottimizzati per il dispositivo dell'utente, in modo da scaricare solo il codice e le risorse necessarie per eseguire l'app. Non è più necessario creare, firmare e gestire più APK per supportare diversi dispositivi e gli utenti ottengono download più piccoli e ottimizzati. Inoltre, utilizzando la libreria Play Core, alcuni moduli e/o funzionalità dinamiche dell'app possono non essere incluse nel pacchetto iniziale dell'app ma possono essere scaricate successivamente dall'utente come APK con funzionalità dinamiche, ad esempio se l'app supporta diverse lingue ma l'utente di default sceglie l'Italiano, le altre lingue potranno essere scaricate successivamente. Come già detto, grazie al Dynamic Delivery e all'utilizzo dell' Android App Bundle, viene aumentata anche la limitazione della dimensione del download compresso a 150 MB senza che siano utilizzati i file di espansione APK. Se l'utilizzo del Dynamic Delivery, non era inizialmente previsto nella propria applicazione, lo si può integrare facilmente, anche se l'aggiunta di moduli di funzionalità dinamiche richiede uno sforzo iniziale maggiore e probabilmente anche il refactoring della propria app.

4.1.2 Il nuovo formato .aab

Un pacchetto di app Android è un file (con estensione .aab) che viene caricato su Google Play per supportare il Dynamic Delivery. I bundle di app sono file binari firmati che organizzano il codice e le risorse della propria app in moduli, come illustrato nella Figura [60].

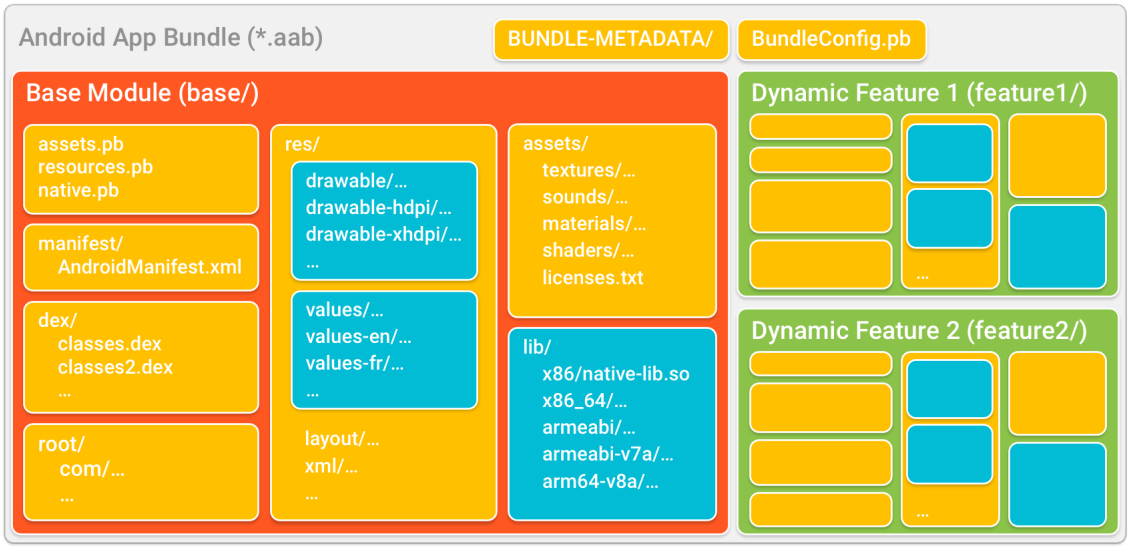

Figura 4.2: Android App Bundle. [60]

Il codice e le risorse per ciascun modulo sono organizzate in modo simile a quelle di in un APK, in quanto ciascuno di questi moduli potrebbe essere generato come un APK separato. Google Play utilizza quindi il pacchetto di app per generare i vari APK da distribuire agli utenti, come l'APK di base, gli APK con funzionalità dinamiche, gli APK di configurazione e (per i dispositivi che non supportano APK divisi) gli APK multipli. Le directory che sono colorate in azzurro, come le directory drawable/ values/ lib rappresentano il codice e le risorse che Google Play utilizza per creare APK configurati per ogni modulo.[39]

4.1.2.1 Build e deploy Android App Bundles

Come già detto precedentemente un App Bundles risulta diverso da un semplice APK, ed in generale è un tipo di formato che include tutto il codice compilato della propria app e le risorse in un singolo artefatto di build. Dopo che viene eseguito l'upload dell'App Bundle firmato, Google Play ha tutti gli strumenti necessari per eseguire il build, firmare l'app e distribuire il tutto agli utenti tramite Dynamic Delivery. Per creare un App Bundle, basta disporre o di Android Studio oppure tramite un altro IDE più un comando da console. Dopodiché basterà eseguire l'upload della propria App

Bundle nella "Play Console" per testare o pubblicare la propria app tramite il Dynamic Delivery. Il servizio di Google Play, chiamato Dynamic Delivery usa le App Bundles per generare APKs ottimizzati per ogni utente (device), così che l'utente possa scaricare solo il codice e le risorse necessarie per la propria app, evitando ad esempio di scaricare lingue o risoluzioni differenti che non sono specifiche per il suo device. In questo modo l'app viene "*divisa*" in porzioni di app, ognuna delle quali al suo interno ha delle funzionalità in più o specifiche ed in questo modo l'APK non è più monolitico ma è composto da tanti piccoli pacchetti che vengono installati on demand sul device. L'APK base contiene il codice e le risorse a cui tutte le altre "*porzioni*" APK possono accedere e prelevare le funzionalità base.

4.1.3 Il linguaggio utilizzato: Java e/o Kotlin

Abbiamo visto gli strumenti per sviluppare e i formato con cui viene rilasciata un'applicazione Android, e ora parleremo del linguaggio utilizzato per sviluppare Android. Se escludiamo i linguaggi e i corrispettivi tools (Xamarin, Cordova, Ionic, ecc.) che permettono uno sviluppo multipiattaforma, inizialmente Android veniva sviluppato in Java, ma con gli anni è stato sostituito da **Kotlin**. Kotlin è un linguaggio di programmazione open-source, progettato dai programmatori di JetBrains nel 2011, si basa sempre sulla JVM, ed è ispirato a linguaggi quali Scala e Java, Pascal,Go e F#. È un linguaggio a tipizzazione statica e forte, orientato alla programmazione ad oggetti ma permette anche l'uso del paradigma funzionale. È stato pensato per "*snellire*" il codice Java nella creazione di applicazioni mobile ed è il linguaggio attualmente usato da diverse società per sviluppare le proprie app Android, tra queste abbiamo: *Uber, Amazon, Netflix, Pinterest, Atlassian, TikTok*. Secondo uno studio, nel 2018 Kotlin risulta essere tra i migliori linguaggi di programmazione per lo sviluppo di applicazioni Android, detenendo il 28% della quota di mercato[1]. Come già detto, rispetto a Java, Kotlin permette di scrivere meno codice e offre anche altri vantaggi, quali:

- una minore probabilità di bug.
- Tempi di sviluppo più brevi.

[1] `https://www.appbrain.com/stats/libraries/details/kotlin/kotlin`

- Riduzione dei costi.

Oltre a ciò, vi sono una serie di librerie Java per creare moduli Kotlin e allo stesso tempo si può generare codice Java partendo da quello scritto in Kotlin grazie a dei convertitori ideati dalla società JetBrains, ciò rende quindi i due linguaggi totalmente compatibili ed offre anche la possibilità di poter scrivere porzione di codice indifferentemente nei due linguaggi. Kotlin però offre anche dei vantaggi in più rispetto a Java:

- è pensato per avere oggetti che di base non possono essere nulli, risolvendo di fatto il "*Bilion-dollar mistake*" (Nullpointer)[40] introdotto da Java evitando quindi interruzioni anomale dell'applicazione.
- Le applicazioni funzionano più velocemente in quanto è basato su una libreria *runtime* che permette alla applicazioni di funzionare più velocemente.

Rispetto però a Java ha anche i seguenti svantaggi:

- Velocità di compilazione non stabile, molte volte risulta più lento di altre.
- Come tutti i nuovi linguaggi, non dispone di tutti gli strumenti e documentazioni riguardanti bug o funzionalità specifiche.

Dal 7 Maggio 2019, Kotlin è diventato il linguaggio consigliato da Google per lo sviluppo di applicazioni Android e la prima opzione di linguaggio se si vuole creare un nuovo progetto in Android Studio.

4.1.4 Come utilizzare un approccio DevOps per lo sviluppo di applicazioni Android

Con le nuove App Bundles sono state colmate molte problematiche che erano emerse in fatto di spazio, tempo di download, e funzionalità specifiche ed ottimizzate per a seconda del dispositivo finale ma nel ciclo di vita di un'applicazione Android e in generale ciò che porta a tramutare il proprio lavoro in APK, vi sono una serie di passi ben specifici. Ovviamente prima di tutto bisogna scegliere l'IDE di sviluppo, fino a diversi anni fa, Android non possedeva un IDE specifico e stabile, ma si appoggiava su Eclipse,

ultimamente però **Android Studio** è diventato stabile ed l'IDE specifico per sviluppare Applicazioni Android. L'IDE però è solo uno strumento con cui viene effettuato lo sviluppo del codice, a questo, se si vuole adottare una mentalità DevOps, bisognerà aggiungere:

- uno strumento che ci permette di **tener traccia delle proprie modifiche**, nello specifico un Software di controllo versione, ovviamente la scelta migliore è utilizzare Git che è direttamente integrato dentro l'IDE, basterà solo configurare il server git preferito (Github, GitLub, bitbucket, ecc.).
- un framework o uno strumento che ci permetta di eseguire dei **Test automatici**, all'interno dell'IDE è già configurato Junit, ma possiamo utilizzare altri framework per i nostri Unit test quali: Mockito[2], Espresso[3], UI Automator per testare anche l'user interaction.
- uno strumento per il check della qualità del proprio codice, tra questi abbiamo Checkstyle[4], SonarQube[5], SpotBugs [6] e Analizo [7]
- la continuous integration/continuous deployment.

4.2 La Continuous Integration in Android

Partiremo dalla Continuous Integration, in generale esistono molte opzioni per configurare il server per la CI. Si può usare un'impostazione personalizzata utilizzando le numerose funzionalità di Jenkins oppure si può usare uno dei numerosi servizi che una CI fornisce di tipo "*click-and-go*".

4.2.1 Tool d'esempio

Nello specifico di Android abbiamo:

[2] `https://github.com/mockito/mockito`
[3] `https://developer.android.com/studio/test/espresso-test-recorder`
[4] `https://checkstyle.sourceforge.io/`
[5] `https://www.sonarqube.org/`
[6] `https://spotbugs.github.io/`
[7] `http://www.analizo.org/`

- Tramite **CircleCi**[8]: permette Continuous integration per le applicazioni Android in una macchina Linux virtuale pre-configurata. Per comodità, CircleCI fornisce una serie di immagini Docker per la creazione di app Android. Queste immagini predefinite sono disponibili nell'organizzazione CircleCI su Docker Hub, invece il codice sorgente e i file Docker sono disponibili in un repository GitHub.

- Tramite **Travis CI** [9]: fornisce un servizio per compilare, testare ed eseguire il deploy del proprio codice contenuto da GitHub. È sotto licenza MIT e va configurato tramite YAML. Il software è tecnicamente gratuito e disponibile frammentariamente su GitHub con licenze permissive. La società nota, tuttavia, che il gran numero di attività che un utente deve monitorare ed eseguire può rendere difficile per alcuni utenti integrare correttamente la versione Enterprise con la propria infrastruttura.

- Tramite **TeamCity by JetBrains**[10]. Team City è sia un build management che un server di Continuous Integration di JetBrains. Al suo interno viene usato anche **Gradle Play Publisher**: è il plug-in di Gradle non ufficiale di Android. Può fare qualsiasi cosa, dalla creazione, il caricamento e la promozione della propria App Bundle o dell'APK alla pubblicazione di elenchi di Applicazioni ed altri metadati.

- Tramite **Firebase**[11] e Jenkins: Firebase è una piattaforma di sviluppo di applicazioni Web e mobili sviluppata da Firebase, Inc. nel 2011, poi acquisita da Google nel 2014. A partire da ottobre 2018, la piattaforma Firebase ha 18 prodotti, utilizzati da 1,5 milioni di Applicazioni. Firebase è principalmente usato per testare la propria applicazione. Jenkins invece è un server di Continuous Integration, ma necessita di un server di installazione per poter funzionare.

- Tramite **Bitrise**[12]:un server di CI/CD specifico per dispositivi mobile.

- Tramite **Buddybuild**[13]: unisce CI/CD e una soluzione di feedback iterativo in un'unica piattaforma.

[8] `https://circleci.com/build-environments/linux/android/`
[9] `https://docs.travis-ci.com/user/languages/android/`
[10] `https://www.jetbrains.com/teamcity/`
[11] `https://firebase.google.com/docs/android/setup`
[12] `https://www.bitrise.io/features/android-features`
[13] `https://www.buddybuild.com/blog/buddybuild-for-android`

Tutti gli strumenti elencati possono essere applicati nel mondo Android, tra quelle elencate è emerso che:

- CircleCi: non è molto pensato ed ottimizzato per dispositivi mobile, ciò rallenta parecchio il processo di sviluppo ed automazione di un'applicazione in quanto risulta difficile configurare il tutto.
- Travis era un'ottima scelta, ma non supporta progetti presenti in BitBucket o GitLab. Oltre a ciò manca un sistema di automazione delle impostazioni di compilazione, la distribuzione automatica e la firma automatica del codice, che potrebbe essere evitato tramite uno strumento CI apposito per mobile.
- TeamCity by JetBrains: ha molte funzionalità a pagamento; con il piano gratuito si possono eseguire massimo 100 build e molti plugins presenti non funzionano bene. Oltre a ciò necessita di una macchina server per essere installato e configurato.
- Firebase: va utilizzato insieme a Jenkins, quindi necessita di una doppia configurazione/installazione, una per Firebase ed una per Jenkins, e, quest'ultimo, come teamcity necessita di una macchina server per essere installato e configurato.
- Buddybuild: come TeamCity ha molte funzionalità a pagamento che ne limitano l'utilizzo.

La scelta quindi è stata quella di Bitrise in quanto oltre a permettere anche il Continuous Deployment, è uno strumento pensato per eseguire CI/CD principalmente su dispositivi Android, oltre a permettere l'integrazione anche personalizzata con tantissimi software (come Bitbucket, GitHub, ecc.). Quasi tutti gli step del workflow sono open source in modo da poterli modificare e condividerli a piacimento oppure crearne di nuovi.

4.2.2 Bitrise

Bitrise è: una soluzione specifica per le app Android, infatti è possibile sia eseguire la CI che configurando e firmare il progetto Android, basterà poi eseguire l'upload del proprio keystore file, aggiungere lo step della firma Android e poi l'app potrà essere distribuita. Si possono vedere la UI(User Interface) e gli Unit/UI test tramite una pagina

apposita (con tanto di video, screenshot e log), che sono integrati con Firebase Test Lab's, grazie a ciò è facile analizzare e trovare i bug presenti nella propria applicazione. Dopo aver configurato il Release, quest'ultimo diventa automatizzato (Continuous Deployment). Per i test si può utilizzare un link pubblico dove una persona può accedere e testare la propria app sul proprio device, oppure per test esterni inviando la propria modifica nell'app di Google Play. Come già detto si può direttamente aggiornare la propria app firmata nello store di Google Play, ciò è possibile grazie al fatto che le proprie credenziali sono state configurate e salvate in modo criptato dentro Bitriste che poi automaticamente eseguirà il deploy delle modifiche dopo che sarà stato eseguito il build dell'app. Oltre a ciò il deploy può essere già eseguito in modo separato tramite molti tools che possono essere direttamente integrati come Appaloose, DeployGate, Appetize.io, Amazon Device Farm.

4.3 Il Continuous Testing e Development in Android

Gli **Unit Test** sono importanti in qualsiasi ambito in quanto danno un'idea esatta di quale parte del proprio codice sia funzionante e quale no. L'esecuzione di Unit test nella Continuous Integration è molto semplice, ma esiste una metrica importante che si può direttamente osservare: la copertura dei test. Fortunatamente Gradle viene fornito con un plug-in di test chiamato JaCoCo già integrato al quale viene poi aggiunto il plug-in JaCoCo di Android. È anche facile da abilitare, basta semplicemente aggiungerlo al file gradle e scegliere il tipo di report che si desidera avere.

Gli **Instrumentation tests** sono invece test eseguiti su dispositivi Android in modo che possano accedere all'API del framework Android. Normalmente sono più lenti degli Unit Test e richiedono di essere eseguiti su dei dispositivi specifici, di solito un emulatore o una farm di dispositivi, sia nel cloud che nel proprio dispositivo. Esistono molti servizi che forniscono farm di dispositivi cloud ma se si hanno abbastanza dispositivi nel proprio ufficio, si può cercare di crearne uno proprio tramite l'aiuto di [14]Open STF | Farm Test per smartphone. Se l'esito dei test è positivo, si può direttamente portare la modifica in

[14]https://openstf.io/

produzione tramite un rilascio,come visto al punto precedente, andando ad adottare la pratica del continuous development.

4.3.1 Continuous Monitoring in Android

In questo caso non vi è una vera e propria pratica di monitoraggio, sicuramente il monitoraggio può essere fatto tramite i feedback degli utenti che vanno a coprire i casi non presi in considerazione, sia per quanto riguarda la user experience sia per quanto riguarda i modelli o le versioni di Android customizzate che potrebbero presentare problematiche non riscontrare in fase di test.

4.4 Continuos Deployment in Android

Dover eseguire degli aggiornamenti manuali è un attività che deve essere automatizzata per evitare errori umani e perdite di tempo. Se si dovesse caricare un APK errato sull'App Store, saranno necessari giorni prima che la correzione raggiunga gli utenti, il che può significare: perdere clienti, denaro, rilasciare funzionalità che non sarebbero dovute essere rilasciate e altro ancora. Inviare allo store o al client una build errata farà perdere tempo nel cercare di interpretare i feedback e capire qual è il problema. L'automazione di questo processo consente di ridurre la percentuale di errori,la quantità di lavoro da svolgere aumentando quindi il tempo disponibile per il team di sviluppo che può costruire e distribuire l'applicazione sull'App Store. Ci sono infatti diversi strumenti di deploy, tra cui Bitrise che è stato visto precedentemente e *Fastlane*.[15] Fastlane è uno strumento basato su ruby per il Continuos Deployment che permette di scrivere script che consentono non solo di distribuire, ma anche di aggiornare il controllo delle versioni, acquisire schermate, aggiornare i registri delle modifiche e altro ancora[16].

[15]https://fastlane.tools/

[16]https://docs.fastlane.tools/getting-started/android/setup/

4.5 Un caso di studio basato su Android: CERERE

Ora applicheremo tutte le metodologie viste e gli appositi strumenti ad un'applicazione Android, denominata **CERERE**. CERERE nasce dall'esigenza di una struttura ospedaliera, nello specifico l'Ospedale Bufalini di Cesena che da alcuni anni e con diversi progetti sta puntando a migliorare la propria assistenza al paziente. Ne è un esempio il progetto **CERERE**, acronimo di *maChinE leaRning in travaglio di parto con e senza analgEsia spino-periduRalE*, che si sviluppa in un contesto di collaborazione tra l'ospedale, in particolare tra le Unità Operative di Analgesia e Rianimazione, Ostetricia e Ginecologia e l'Università di Bologna attraverso il gruppo di ricerca del Dipartimento di Informatica -Ingegneria E Scienze Informatiche (DISI) della sede di Cesena.

4.5.1 Architettura di CERERE

E' chiaro che il sistema debba riprendere il modello client-server in cui vi è un solo server di riferimento che permette la centralizzazione dei dati e la propagazione delle modifiche fra tutti i client connessi. Il modello client-server si basa su una sostanziale differenza di poteri fra chi utilizza e richiede i dati (il client) e chi invece fornisce i dati e gestisce le autorizzazioni di accesso (il server). Le due figure devono avere dei protocolli di comunicazione in comune e interagire attraverso una connessione internet che può essere privata o pubblica. Per rispondere a queste esigenze verrà utilizzata l'architettura REpresentational State Transfer (REST). I singoli componenti del sistema hanno subito dei test durante tutta la fase di implementazione per testare in particolar modo la corretta interazione client-server e che il workow dei dati fosse gestito correttamente. I test sono stati effettuati attraverso i framework dedicati a questo scopo come JUnit e AndroidJUnit.

4.5.2 Come applicare i principi DevOps

Il progetto è già nato con un'ottica DevOps in quanto: è presente un repository Git condiviso, nello specifico Bitbucket che, tramite anche Sourcetree, permette una buona gestione delle modifiche e dei branch presenti. Come già detto, sono già stati eseguite delle integrazione fra le singole componenti, ed ha anche un'architettura basata su più

servizi. L'IDE utilizzato è comune a tutti gli sviluppatori, ed è Android Studio. Come tutti i progetti Android, troviamo anche un'automation build che in questo caso è Gradle. Al suo interno sono anche presenti alcune classi di test.

4.5.3 Come applicare la CI/CD tramite Bitrise

Ora che sappiamo com'è strutturato, andremo ad aggiungere alcuni classi di test, in quanto sono presenti solo degli Unit test.

4.5.3.1 Creazione di UI Test

Grazie ad **Express** che è già presente all'interno di Android Studio, la creazione di un UI(User Interface) Test risulterà poco difficoltosa. Basterà infatti cliccare sul bottone "*Record Espresso Test*" e la nostra app partirà in automatico nell'emulatore da noi impostato. La differenza rispetto ad un normale build o utilizzo è che tutte le operazioni che vengono svolte da quando l'app si è avviatab vengono poi salvate e tramutate in automatico in codice da Espresso; non solo, è possibile anche scegliere esattamente cosa ci si aspetta di trovare in una determinata schermata, che, in termini di codice, si traduce poi in un AssertEquals o un AssertNotEquals. La nostra class di test, sarà una classe che conterrà al suo interno tutti gli step necessari al normale utilizzo dell'app infatti verrà:

- eseguito il login.
- Premuta la prima stanza e verrà inserito un paziente.
- Riempita la cartella clinica del paziente, che verrà poi salvata in locale.
- Eseguito il logout dell'applicazione.

Se uno degli step presenti nel nostro test dovesse fallire, il test stesso fallirebbe. Gli step sono stati raggruppati per una maggiore facilità di utilizzo in quanto comunque, pur appartenendo alla stessa classe di test, sono facilmente individuabili e risolvibili. La classe avrà quindi la seguente struttura:

```
@LargeTest
@RunWith(AndroidJUnit4.class)
public class InsertFirst {

@Rule
public ActivityTestRule<LoginActivity> mActivityTestRule =
new ActivityTestRule<>(LoginActivity.class);

@Test
public void insertFirst() {
ViewInteraction appCompatEditText = onView(
allOf(withId(R.id.login_et_name),
childAtPosition(childAtPosition(withClassName
(is("android.widget.LinearLayout")),
0),1),isDisplayed()));
appCompatEditText.perform(replaceText("admin"), closeSoftKeyboard());
```

```
ViewInteraction appCompatTextView = onView(
allOf(withId(R.id.title), withText("Disconnetti"),
childAtPosition(childAtPosition(withId(R.id.content),
0),0),isDisplayed()));
appCompatTextView.perform(click());
}
private static Matcher<View> childAtPosition(
final Matcher<View> parentMatcher, final int position) {

return new TypeSafeMatcher<View>() {
@Override
public void describeTo(Description description) {
description.appendText("Child at position "+position+" in parent ");
parentMatcher.describeTo(description);
}

@Override
public boolean matchesSafely(View view) {
ViewParent parent = view.getParent();
return parent instanceof ViewGroup && parentMatcher.matches(parent)
&& view.equals(((ViewGroup) parent).getChildAt(position));}
  };
}
```

Come si può notare, ci sono alcuni metodi che sono presi direttamente da Espresso, tra questi troviamo, ad esempio, **closeSoftKeyboard**, che permette di usare (e simulare) la tastiera e inserire la parola desiderata, ovviamente oltre alla tastiera verranno simulate tutte le azioni che un normale utente può intraprendere, come premere un determinato bottone, eseguire lo scorrimento dello schermo, e tanto altro. Oltre ad aver aggiunto alcune classi di test, altre sono state modificate e/o in quanto vecchie e/o obsolete.

4.5.3.2 Installazione e configurazione iniziale di Bitrise

Dopo aver creato le nostre classi di test, si passerà poi a Bitrise. Bitrise non va installato sul proprio computer ma verrà eseguito tutto online in una pagina dedicata. I passi per importare un progetto in Bitrise sono:

1. **Choose Account**: serve per scegliere se il proprio progetto sarà pubblico (accessibile a tutti) o privato.

2. **Connect your repository**: serve per scegliere da che piattaforma si vuole importare i propri progetti; nel nostro caso sceglieremo Bitbucket visibile in figura [61]

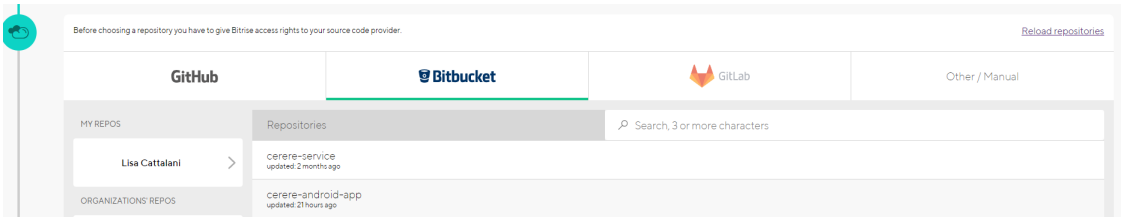

Figura 4.3: Step1. [61]

3. **Setup repository access**: dopo aver scelto il repository e il progetto, si associerà una chiave SSH di condivisione, se si dispone già di una chiave di condivisione, basterà aggiungerla, altrimenti verrà creata una chiave SSH, che dovrà poi essere configurata anche nel repository scelto, questa chiave serve poi per lo step del Webhook che spiegheremo dopo. La configurazione è visibile nelle figura [62] e la chiave viene poi aggiunta su Bitbucket come in figura [63]

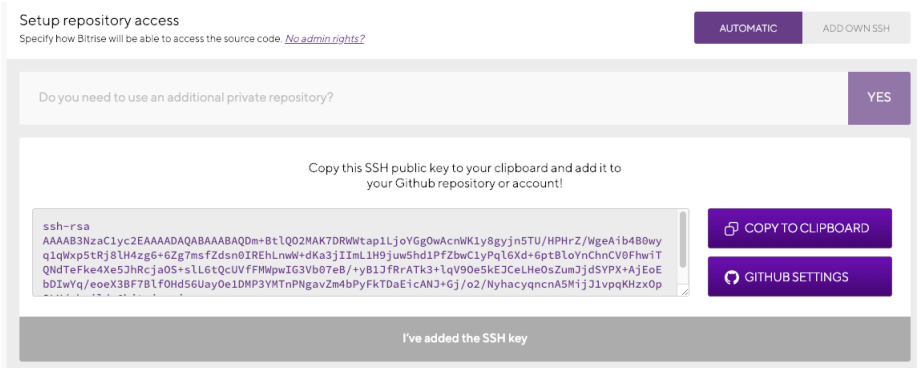

Figura 4.4: Step2. [62]

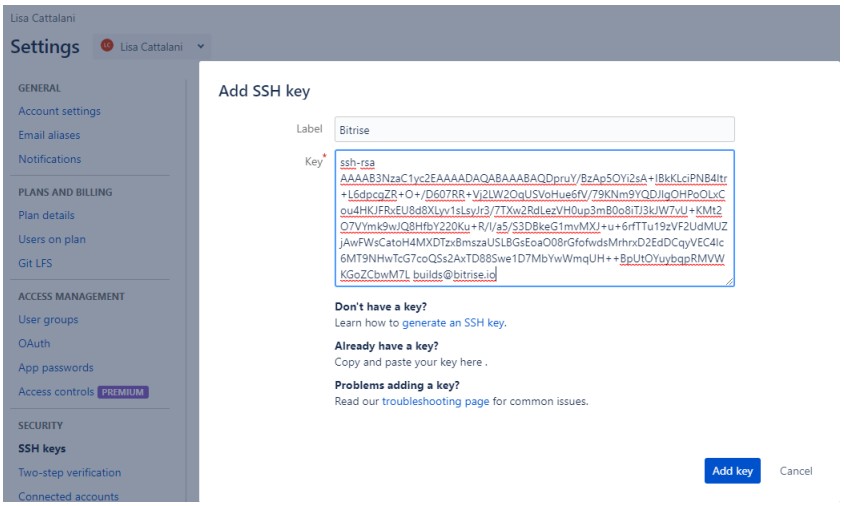

Figura 4.5: Step2 Bitbucket. [63]

4. **Choose branch**: serve per scegliere su quale branch si preferisce lavorare.

5. **Validating repository**: Non è un vero e proprio step, ma ciò permette a Bitrise di validare il progetto. Gli errori che potrebbero presentarsi in questa fase sono molteplici, uno di questi potrebbe essere dovuto al fatto di non aver correttamente configurato la chiave SSH, e l'altro si potrebbe verificare se si prova ad importare un progetto non consentito infatti darebbe esito negativo in quanto non troverebbe i file che si aspetterebbe di trovare, come Nella figura [64], per la precisazione Bitrise accetta progetti di tipo: ios, Android, per poi passare a progetti crossPlatform quali Xamarin, Cordoba, Ionic, ecc.

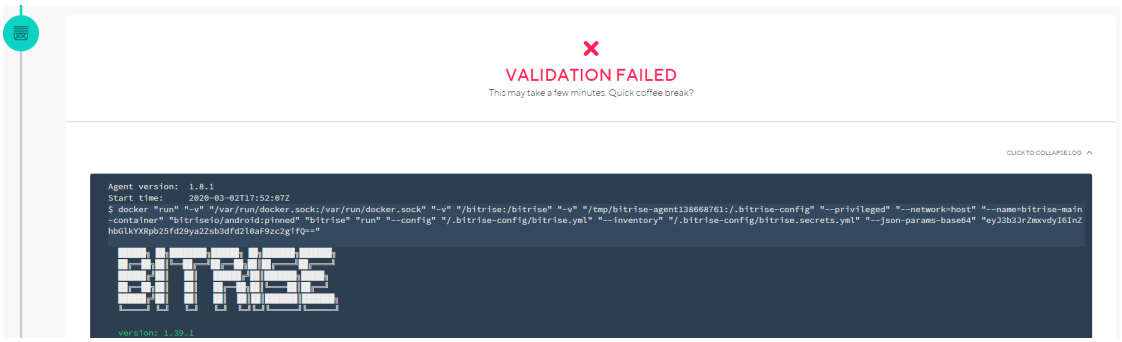

Figura 4.6: Step non valido. [64]

6. **Project build configuration**: a seconda della tipologia di progetto rilevata allo step precedente, qua verrà mostrato il tipo rilavato, e si può specificare il modulo che si vuole prendere in considerazione, nel nostro caso sarà "app". Oltre a ciò, si specificherà anche lo stack Android che si vuole utilizzare, nel nostro caso lasceremo quello di Default di Android & Docker, in ubuntu 16.04 (sarà la macchina virtuale dove verrà eseguito il build).

7. **App icon**: ci verrà proposto se vogliamo utilizzare anche un'icona per la nostra app, serve più che altro per riconoscere più facilmente le nostre applicazioni o, in caso in cui venisse poi fatto il build nello store dell'applicazione, verrà usata come icona.

8. **Webhook setup**: grazie a questo step sarà possibile anche associare il Webhook (praticamente quando si verifica un evento su Bitbucket, ad esempio un commit, verrà inviata una post htpp verso il webhook configurato) in modo tale che ad ogni commit sia fatto partire un build automatico.

Una volta configurato, partirà il primo build, quest'ultimo avrà un Workflow composto da pochi step, ma lo vedremo meglio nel prossimo capitolo.

4.5.3.3 Configurazione del Workflow

Come già accennato, la build della nostra applicazione in realtà avviene attraverso una serie di passi denominati Workflow. In quello inizialmente configurato da Bitrise, troviamo i seguenti passi:

1. Activate SSH key (RSA private key)
2. Git Clone Repository
3. Bitrise.io Cache:Pull
4. Do anything with Script step
5. Install missing Android SDK components
6. Android Lint
7. Android Unit Test
8. Deploy to Bitrise.io - Apps, Logs, Artifacts
9. Bitrise.io Cache:Push

ognuno di questi step viene rappresentato con un'icona specifica, oltre a ciò ogni step contiene:

- **una versione**: serve per specificare quale versione di quello specifico step si vuole utilizzare, se si è abbastanza sicuri che quello specifico step non influisca sui restanti, si può usare anche l'opzione "Always latest" che in automatico prende sempre l'ultima versione disponibile.
- **un flag (Run if previous Step failed)**: questo flag serve per dire se si vuole comunque eseguire lo step, nonostante lo step precedente, o altri, non siano andati a buon fine. Se ad esempio si vuole usare uno step che è in stretto collegamento con un altro step, può aver senso utilizzare questa opzione.
- **Input variables**: sono le variabili che lo step in questione prende in input. Il valore che può prendere in input di solito è sempre inserito di default, ma può anche variare a seconda dell'esigenza oppure richiedere un particolare input che è il risultato di uno step precedente.

- **Altre variabili**: molti step, possono presentare configurazioni particolari a seconda dell'utilizzo, ad esempio possono essere specificati utilizzi diversi a seconda del tipo di test che si vuole eseguire o del package che si vuole prendere in considerazione.

- **Output variables**: quasi tutti gli step producono delle variabili in output che possono essere utilizzate come input dai successivi step.

Un esempio può essere visionato nella figura [65]. Gli step che sono presenti di default,

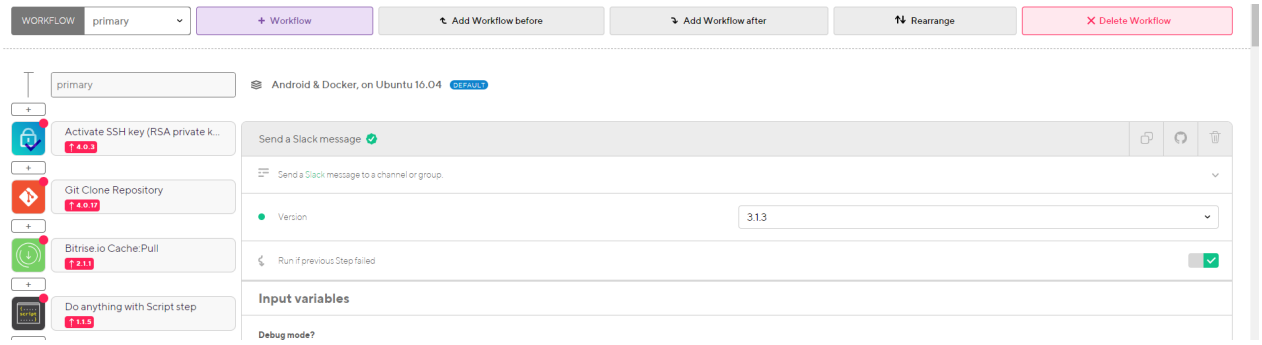

Figura 4.7: Configurazione degli step. [65]

non coprono però l'interezza dei test che ci servono, infatti andremo ad aggiungere 2 step: Android Build for UI Testing e [BETA] Virtual Device Testing for Android, quindi il nostro nuovo workflow sarà così strutturato:

1. **Activate SSH key (RSA private key)**: se il nostro repository risulta privato, questo step serve per associare la chiave ssh, l'attivazione di questa chiave, consente l'accesso al nostro repository. Questo step viene aggiunto in automatico da bitrise se nella procedura iniziale si è scelto un repository privato.

2. **Git Clone Repository**: permette un clone in locale del repository, questo viene sempre fatto in quanto, essendo presente il WebHook, ad ogni nuovo commit il progetto cambia e quindi è bene rifare un clone del progetto. Anche questo step è aggiunto in automatico da Bitrise, se nella procedura guidata è stato configurato un determinato repository.

3. **Bitrise.io Cache:Pull**: serve per scaricare la cache di build da Bitrise.io. Lo scopo è quello di velocizzare le build in quanto non dover scaricare ogni volta ogni singola dipendenza rappresenta un ingente risparmio di tempo. Anche questo step viene automaticamente aggiunto di default da Bitrise.

4. **Do anything with Script step**: Anche questo passaggio viene aggiunto automaticamente al flusso di lavoro. Serve per permettere il fallimento del build se uno degli step presenti dovesse fallire e per avere la verbosità settata a debug dei log.

5. **Install missing Android SDK components**: anche questo passaggio viene aggiunto automaticamente al flusso di lavoro per progetti Android. Permette in automatico l'installazione di eventuali componenti sdk che non sono stati installati ai punti precedenti o che necessitano di un'installazione perchè espressamente richiesta nel progetto, oltre a ciò controllerà che siamo aggiornati all'ultima versione SDK presente.

6. **Android Lint**: permette un'analisi più approfondita del codice, a seconda poi di come si è configurata la verbosità all'interno del file Gradle, anche i semplici warning potrebbero diventare degli errori e quindi di fatto compromettere il build dell'applicazione.

7. **Android Unit Test**: esegue gli unit test.

8. **Android Build for UI Testing**: serve per poter eseguire il punto 9, in quanto crea la base dell'apk che poi verrà testato nel punto successivo.

9. **[BETA] Virtual Device Testing for Android**: esegue gli UI Test della nostra applicazione, tramite Firebase test lab, per farlo bisognerà configurare il tipo di classe di test che andremo ad utilizzare, nel nostro caso:
android.support.test.runner.AndroidJUnitRunner, il tipo di dispositivo dove sarà poi emulata la nostra app, di default sarà un Nexus con android 24 al suo interno.

10. **Deploy to Bitrise.io - Apps, Logs, Artifacts**: serve per generare gli APK e gli artefatti che saranno poi disponibili nella sezione "Apps and Artifacts".

11. **Bitrise.io Cache:Push**: esegue delle operazioni necessarie a Bitrise per tener traccia della build e salvarla in cache, se la build risultasse uguale ad una precedente, la cache non verrà aggiornata.

4.5.3.4 La prima build e gli errori riscontrati

Dopo che abbiamo configurato sia Bitrise che il Workflow da applicare al nostro progetto, eseguiremo la nostra prima build, che purtroppo darà esito negativo. Nei log di Bitrise troviamo infatti scritto:

```
> Task :app:lintDebug FAILED
FAILURE: Build failed with an exception.
* What went wrong:
Execution failed for task ':app:lintDebug'.
> Lint found errors in the project; aborting build.

Fix the issues identified by lint, or add the following
to your build script to proceed with errors:
...
android {
lintOptions {
abortOnError false
}
}

* Try:
Run with --stacktrace option to get the stack trace.
Run with --info or --debug option to get more log output.
Run with --scan to get full insights.
```

Quest'errore era dovuto principalmente al fatto che nel nostro file non fosse presenta la dicitura che Bitrise stesso suggeriva (lintOptions). Infatti Lint (un tool android che ricerca nel progetto potenziali bugs) era settato per dare errore anche in caso di warning.

Come già detto prima, essendo questo step fallito, non sono stati eseguiti i restanti step, quindi dopo aver sistemato l'errore e commitato, il secondo build è partito in automatico(grazie al Webhook che abbiamo configurato). Nel secondo build lo step Android Lint, è risultato corretto, ma questa volta sono comparsi nuovi errori nello step Android Unit Test. Infatti in questo step vengono eseguiti tutti gli Unit test presenti nella cartella *src/test/java*, in quanto alcune classi mancavano di notazioni, oppure presentavo metodologie non più in linea con il progetto e quindi sono state rimosse, o sono stati rimossi alcuni metodi presenti al loro interno. Oltre a ciò, mancava l'annotazione utile nelle classi di test che è stata inserita nel file gradle.setting nel seguente modo:

```
javaCompileOptions {
annotationProcessorOptions {
includeCompileClasspath true
 }
}
```

Dopo aver risolto i vari problemi ed essere riusciti finalmente nel build della nostra applicazione, nella sezione **APPS & ARTIFACTS**, a destra della sezione Logs(la sezione dove vengono mostrati gli step visti in precedenza), possiamo trovare:

- le nostre classi di test all'interno di un file zip,
- il risultato dei test in un file zip,
- 2 file html contenente il risultato della scansione eseguita tramite lint e
- gli apk della nostra applicazione.

4.5.4.5 L'esito delle classi di test

Ciò che è interessante di Bitrise è che è possibile vedere nel dettaglio quali test hanno dato esito negativo, quali no, e gli errori riportati, per farlo basterà cliccare su una delle funzionalità presente nel riquadro "*Add-ons Access your connected add-ons below*" Test-Reports, come riportato nella figura sottostante:

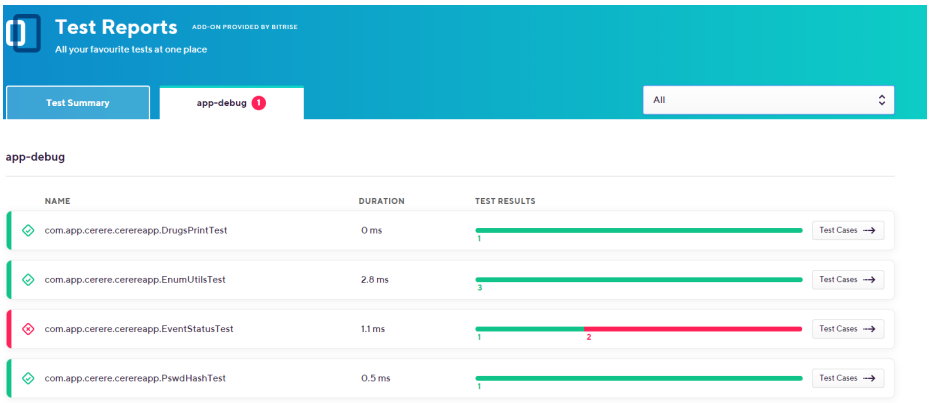

Figura 4.8: Esito dei test. [66]

Se andiamo nello specifico dei test che hanno dato errore, cliccando semplicemente sul bottone "*Test Case*" presente in figura, troveremo le classi e i metodi coinvolti, nel nostro caso ad esempio "*com.app.cerere.cerereapp.EventStatusTest*" ha dato esito negativo in quanto *java.lang.AssertionError: expected:<PARTIAL> but was:<EMPTY>*. Man mano sono state quindi sistemate le varie classi di test che sono risultate problematiche, ed è stato possibile, grazie allo step *Virtual Device Testing for Android*, vedere sia l'esito dei test, che proprio parte dei nostri test. Infatti sempre in Test Report, vi è una sezione apposita dedicata a test denominati *Firebase TestLab*, ci sarà anche una sezione video dov'è possibile vedere direttamente parte di alcuni nostri test, come nella figura:

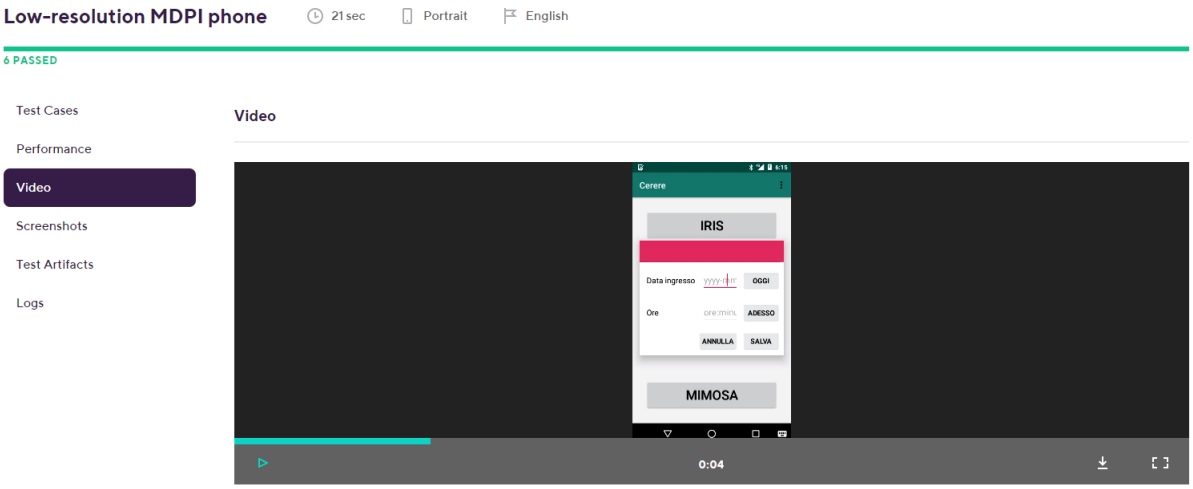

Figura 4.9: Video dei test. [67]

Oltre alla sezione video, sono presenti anche le sezioni:

- **Test Cases** dove vengono riepilogati e vari esiti: passed, failed.
- **Performance**: in questa sezione, sono riepilogate con un breve grafico le performance dei nostri test, nello specifico viene analizzata: quanto cpu è richiesta, quanta ram viene utilizzata, e quanto sia l'utilizzo a livello di network.
- **Screenshots**: vengono mostrati alcuni screenshots se sono stati eseguiti durante la fase di test.
- **Test Artifacts**: qui vengono riepilogati i vari file di log prodotti dagli step, nello specifico possiamo trovare l'elenco dei bug riscontrati. In realtà il file che viene prodotto è molto più di un semplice file di log, infatti al suo interno troviamo tutti gli stati e tutte le operazioni eseguite dalla macchina virtuale con tutti i vari stati, memoria ecc. Abbiamo anche un file denominato instrumentation.results, dove viene riepilogato l'ordine con cui sono stati chiamati i vari test e il tempo impiegato. Poi troviamo diversi file denominati test_cases/XXXX_logcat, dove vengono brevemente riepilogati i nostri test. Un file xml, test_result_1.xml dove

l'esito dei test è scritto con tag xml. E' anche possibile scaricare il video presente nella sezione video.

- **Logs**: alcuni log generati dal dispositivo virtuale durante l'esecuzione dei test. Come ad esempio il fatto che è partita una Activity Manager e quale nello specifico.

4.5.4.6 Il Continuous deployment

Nonostante non sia previsto in quanto l'applicazione per ora viene installata solo direttamente nei dispositivi e non è presente nell'app store. Tramite Bitrise, si può scegliere un altro Workflow, che non sarà quello *primary*, ma quello *deploy* e rilasciare direttamente la nostra app nell'app store. Molti passi in realtà non differiscono dalla versione di semplice continuous integration, ma anzi sono identici con l'aggiunta invece dei 3 step *Android Sign*, *Android Build*, "*Deploy to Google Play*" e con l'aggiunta della chiave cifrata per poter inserire l'app nello store. Bitrise stesso mostra i passi per firmare il nostro APK. Quindi al nostro precedente worflow aggiungeremo:

1. **Android Sign**: questo step serve per firmare digitalmente il file APK della propria app nel proprio flusso di lavoro. Basta semplicemente caricare il proprio file keystore nella scheda "*Code signing*" del workflow. Infatti sono richiesti: url del keystore, la password del keystore, l'alias del keystore e la propria password privata.

2. **Android Build**: esegue proprio il build della nostra applicazione.

3. **Deploy to Google Play**: questo step permette il caricamento e il rilascio della propria app sul play Store di Google. È possibile distribuire la propria APK, sia tramite la singola APK oppure molteplici APK tramite il Bundle deploys. Questo step per poter funzionare prevede che la propria app sia già stata distribuita almeno una volta nell'app store e che nel proprio app store sia stato configurato l'accesso tramite API. Ovviamente bisognerà aggiungere le credenziali di Google Play, oltre a specificare il nome del package, il path dell'app, ecc.

Dopo aver rilasciato l'app nel proprio store, purtroppo come abbiamo già detto più volte, non possiamo far altro che aspettare eventuali feedback rilasciati dagli utenti, in quanto non ci sono pratiche per il monitoraggio dell'app.

4.5.4 I vantaggi e gli svantaggi ottenuti con Bitrise

Bitrise è risultato un ottimo strumento per la Continuous Integrations, permettendo tutti gli step che questo pratica prevede. Abbiamo sia le operazioni principali quali l'associazione di una chiave SSH e il Webhook, sia la build della nostra applicazione che l'esecuzione delle classi di test, oltretutto tramite una macchina virtuale direttamente configurata e creata da Bitrise. I risultati degli Unit Test però non sono facili da individuare a differenza degli Integration test, in quanto non abbiamo una chiara suddivisione dei test che hanno avuto esito positivo o meno. Per scoprire perché una build non è riuscita, in questo caso, dobbiamo andare nei registri e cercare il motivo dell'errore, questo a volte può richiedere un po' di tempo. Oltre a ciò, essendo comunque relativamente recente (2014) e non molto utilizzato, le documentazioni in tal senso non sono molto esaustive, e nonostante le nostre operazioni non fossero particolarmente difficili, non era ben spiegato come configurare i vari step, o quali usare in caso si volesse utilizzare gli UI test. Nonostante non abbiamo potuto provare direttamente la fase di rilascio, sicuramente se viene aggiunta al flusso di lavoro, Bitrise si trasforma in uno strumento molto potente e completo per poter sviluppare applicazioni Android, senza considerare il fatto che prevede diversi tipi di Account e quello gratuito permette comunque un uso completo di tutte le funzionalità presenti.

Conclusioni

Ciò che è emerso dall'elaborato, è come DevOps sia di fatto diventato l'approccio essenziale ai problemi e allo sviluppo di nuove soluzioni software, e, di come riesce a coprire una gran quantità di casiste, infatti partendo dal più comune IT si riesce a passare al più complesso ambito dell'IoT. Nel nostro caso lo abbiamo applicato ad un progetto già avviato, come CERERE, sviluppato in Android. Per farlo, oltre a seguire le regole base della buona programmazione, abbiamo utilizzato anche uno strumento esterno, come Bitrise, che ci ha permesso di finalizzare il nostro approccio DevOps ed applicare la Continuous Integration. Nonostante la CI, sia solo una delle tante pratiche presenti in DevOps, poterla applicare ad un progetto già avviato, ci fa capire la grande potenzialità di questa pratica. È vero che il progetto preso in considerazione è abbastanza semplice e non di uso comune, ma pensare di poter applicare lo stesso principio ad un altro progetto che richiede continui sviluppi, ci permette di risparmiare una grandissima quantità di tempo, ed evitare spiacevoli inconvenienti. Tramite Bitrise poi, nonostante lo si sia solo mostrato e non veramente applicato, è possibile eseguire anche il Continuous Delivery, che ci permette di risparmiare ulteriore tempo ed è configurato per avere degli step ben precisi, evitando quindi errori imprevisti o errori manuali. Considerando in generale l'evoluzione tecnologica e quanto sia importante sviluppare nel minor tempo possibile, poter disporre di approcci di questo tipo, utilizzando magari dei tools già preconfigurati come Bitrise, Jenkins, ecc., si tramuta in un grande valore aggiunto per il software sviluppato, che lo rende di gran lunga più affidabile, scalabile, e facilmente manipolabile, rispetto ad altri software. In generale se si riesce ad applicare questo tipo di approccio già prima di aver sviluppato il nostro progetto, riusciremmo ad avere dei benefici, che a lungo andare si tramuteranno in maggiore tempo e guadagno. Infatti in

molte aziende, si tende ancora a non aver un approccio ben definito e a considerare questi approcci come "*una perdita di tempo*", continuando ad eseguire rilasci, test e integrazioni manuali, senza che venga considerato il tempo perso e l'alta probabilità di errori per ognuna di queste attività; si arriverà poi al punto in cui il progetto diventerà troppo complesso e coprire tutte le casistiche non sarà più possibile con conseguente perdita di fiducia da parte del cliente finale. Un possibile ulteriore sviluppo del nostro elaborato potrebbe sicuramente essere quello di ultimare il Continuos Delivery, ed integrare e monitorare anche gli altri applicativi coinvolti.

Bibliografia

[1] Len Bas, Ingo Weber,Liming Zhu (2015). Chapter 1 "What is DevOps?" *"DevOps A Software Architect's Perspective"*

[2] SDLC Models Explained: Agile, Waterfall, V-Shaped, Iterative, Spiral `https://existek.com/blog/sdlc-models/`

[3] Prakriti Trivedi ; Ashwani Sharma *A comparative study between iterative waterfall and incremental software development life cycle model for optimizing the resources using computer simulation* `https://ieeexplore-ieee-org.ezproxy.unibo.it/document/6915096`

[4] Alan M. Davis, Edward H. Bersoff, Edward R. Comer *A Strategy for Comparing Alternative Software Development Life Cycle Models* `https://ieeexplore-ieee-org.ezproxy.unibo.it/document/6190`

[5] Adel Alshamrani and Abdullah Bahattab *A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model* `https://www.academia.edu/10793943/A_Comparison_Between_Three_SDLC_Models_Waterfall_Model_Spiral_Model_and_Incremental_Iterative_Model`

[6] Nabil Mohammed Ali Munassar1 and A. Govardhan *A Comparison Between Five Models Of Software Engineering* `https://www.researchgate.net/publication/258959806_A_Comparison_Between_Five_Models_Of_Software_Engineering`

[7] Barry W. Boehm, TRW Defense Systems Group *A Spiral Model of Software Development and Enhancement* `https://ieeexplore-ieee-org.ezproxy.unibo.it/document/59`

[8] *What is domain Driven Design* https://dddcommunity.org/learning-ddd/what_is_ddd/

[9] *Domain-Driven Design – What is it and how do you use it?* https://airbrake.io/blog/software-design/domain-driven-design

[10] Intro to Lean Startup http://leanstartup.pbworks.com/w/page/65946049/Intro%20to%20Lean%20Startup

[11] 5 Reasons Not to Follow the Lean Startup Process for Your Next Idea https://www.entrepreneur.com/article/320508

[12] Eric Evans (2003). Chapter 1. *Domain-Driven Design, Tackling Complexity in the Heart of Software*

[13] John Willis, (2012) The Convergence Of DevOps https://itrevolution.com/the-convergence-of-devops/

[14] Patrick Debois on the State of DevOps, interview. https://www.infoq.com/interviews/debois-devops/

[15] What does ITIL mean? https://www.ibm.com/cloud/learn/it-infrastructure-library

[16] Len Bas, Ingo Weber,Liming Zhu (2015). Chapter 5 *"DevOps A Software Architect's Perspective"*

[17] Kent Beck and Cynthia Andres. 2004. *Extreme Programming Explained: Embrace Change. Addison-Wesley Professional.*

[18] Sameer S Paradkar (2019) .IT .Chapter 11 *"Interview Guide for Freshers"*

[19] Manish Virmani *"Understanding DevOps & Bridging the gap from Continuous Integration to Continuous Delivery* https://ieeexplore-ieee-org.ezproxy.unibo.it/document/7173368

[20] Sam Guckenheimer. *"What are Microservices?"* https://docs.microsoft.com/en-us/azure/devops/learn/what-are-microservices

[21] Len Bas, Ingo Weber,Liming Zhu (2015). Chapter 5 Introduction *"DevOps A Software Architect's Perspective"*

[22] Andrej Dyck, Ralf Penners, and Horst Lichter. 2015. *Towards definitions for release engineering and DevOps.* https://www2.swc.rwth-aachen.de/docs/RELENG2015/DevOpsVsRelEng.pdf

[23] What is Nagios? https://www.nagios.org/about/

[24] CIO Business Technology Leadership (2007). *Strategy planning in the real world*

[25] Antonio Terceiro, Joenio Costa, João Miranda, Paulo Meirelles, Luiz Romário Rios, Lucianna Almeida, Christina Chavez, and Fabio Kon. 2010. *Analizo: An extensible multi-language source code analysis and visualization toolkit.*

[26] Knight Capital Says Trading Glitch Cost It $440 Million https://dealbook.nytimes.com/2012/08/02/knight-capital-says-trading-mishap-cost-it-440-million/

[27] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, Paulo Meirelles. *A Survey of DevOps Concepts and Challenges.* https://dl.acm.org/doi/abs/10.1145/3359981

[28] Goldman Sachs. *Technical error causes erroneous U.S. option trades* https://www.reuters.com/article/us-nasdaq-trades/goldman-sachs-technical-error-causes-erroneous-u-s-option-trades-idUSBRE97J0R920130820

[29] Eueung Mulyana, Rifqy Hakimi, and Hendrawan. 2018.*Bringing automation to the classroom: A ChatOps-based approach.*

[30] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems.*

[31] *"Internet of Things Global Standards Initiative"*. Retrieved 26 June 2015. http://www.itu.int/en/ITU-T/gsi/iot

[32] James Manyika and Michael Chui *By 2025, Internet of things applications could have $11 trillion impact* https://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internet-of-things-applications-could-have-11-trillion-impact

[33] Stephen Cobb. 24 Oct 2016 *10 things to know about the October 21 IoT DDoS attacks* https://www.welivesecurity.com/2016/10/24/10-things-know-october-21-iot-ddos-attacks/

[34] Mark Weiser. Sept 1991 *The Computer of the 21st Century. Scientific American* https://www.lri.fr/~mbl/Stanford/CS477/papers/Weiser-SciAm.pdf

[35] Mike Loukides. *What is DevOps?*

[36] Jason Wong, Gartner. *Addressing DevOps for Mobile App Development* https://blogs.gartner.com/jason-wong/addressing-devops-mobile-app-development/

[37] About the platform, *Platform Architecture* https://developer.android.com/guide/platform

[38] APK Expansion Files https://developer.android.com/google/play/expansion-files

[39] About Android App Bundles https://developer.android.com/guide/app-bundle/

[40] Tony Hoare "Bilion-dollar mistake" https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

[41] The Lean StartUp http://theleanstartup.com/principles

[42] Toyota Production System https://www.creativesafetysupply.com/articles/understanding-the-toyota-production-system/

[43] *SDLC LifeCycle* https://www.vskills.in/certification/blog/software-development-life-cycle-sdlc

[44] *DevOps lifecycle in IT* https://www.embedded.com/applying-devops-to-iot-solution-development/

[45] *Waterfall SDLC Model* https://existek.com/blog/sdlc-models/

[46] *Iterative and Incremental SDLC Model* https://medium.com/@srisayi.bhavani/agile-methodology-zomato-case-study-311da3388518

[47] *Spiral SDLC Model* https://ieeexplore-ieee-org.ezproxy.unibo.it/document/59

[48] *V-shape Model* https://upload.wikimedia.org/wikipedia/commons/9/96/V-model.jpg

[49] *Agile Model* https://existek.com/blog/sdlc-models/

[50] *DevOps lifecycle* https://existek.com/blog/sdlc-models/

[51] *DevOps Lifecycle* https://dzone.com/articles/life-cycle-of-devops

[52] *Deployment pipeline* https://books.google.it/books?id=fcwkCQAAQBAJ&printsec=frontcover&hl=it&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[53] *Figura-DevopsLifecycleIT* https://www.embedded.com/applying-devops-to-iot-solution-development/

[54] *Continuous DevOps* https://www.redhat.com/it/topics/devops/what-is-ci-cd

[55] *CI and CD Flow*

[56] *DevOps lifecycle in IT.* https://www.embedded.com/applying-devops-to-iot-solution-development/

[57] *Mobile And Ubitous Computing*

[58] *DevOps Mobile*

[59] *Architettura Android* https://developer.android.com/guide/platform

[60] *Android App Bundle* https://developer.android.com/guide/app-bundle/

[61] *Settaggio Bitrise Step 1* https://www.bitrise.io/features/android-features

[62] *Settaggio Bitrise Step2* https://www.bitrise.io/features/android-features

[63] *Settaggio Bitrise Step 2 B* https://www.bitrise.io/features/android-features

[64] *Settaggio Bitrise Step invalido* https://www.bitrise.io/features/android-features

[65] *Settaggio Bitrise Step Configurazione* https://www.bitrise.io/features/android-features

[66] *Esito delle classi di Test* https://www.bitrise.io/features/android-features

[67] *Risultati Test Bitrise* https://www.bitrise.io/features/android-features