***CAPITOLO 2***

Introduzione (?)

In questo capitolo verranno affrontate le tematiche necessarie per conoscere e comprendere la metodologia DevOps.

DevOps è un fenomeno nuovo, nato nel 2011, perciò utilizzerò un approccio top-down. Penso infatti che sia fondamentale comprendere le dinamiche che muovono il mondo della produzione nel campo IT per apprezzare a pieno l’innovazione e il cambiamento che il DevOps sta portando all’interno del settore.

Le sezioni che comporranno il capitolo partiranno da un’analisi generale del contesto della produzione, agganciandosi successivamente al campo IT per arrivare infine a parlare di DevOps.

PRODUZIONE DI MASSA

*La seconda rivoluzione industriale*

Nel periodo compreso tra il 1856 e il 1878 possiamo collocare la seconda rivoluzione industriale. Anni di fuoco che hanno portato trasformazioni economiche e sociali sconvolgenti.  
Le macchine e gli apparati produttivi raggiungevano la maturità all’interno del contesto industriale. Scienza e tecnica continuavano a fornire macchinari sempre più veloci e potenti portando la produzione a livelli mai visti prima.

I progressi tecnologici si accompagnarono al progressivo ingrandimento dei complessi industriali. Il numero di fabbriche che superavano il migliaio di dipendenti cominciò a farsi più frequente verso fine 1800, anche grazie a fusioni tra imprese che imboccavano sempre più velocemente la strada del gigantismo industriale.  
La concentrazione di manodopera in grandi stabilimenti, se da un lato era destinata ad apparire come la risposta più logica alla esigenza di una produzione su larga scala, dall'altro lato cominciava a porre problemi organizzativi nuovi per i quali non potevano più valere le vecchie soluzioni di origine artigianale.

Il crescente bisogno di forza lavoro viene soddisfatto attraverso il reclutamento di masse di estradizione contadina, i paesi più industrializzati furono travolti dalle dimensioni imponenti di tale fenomeno, e soprattutto negli Stati Uniti affluirono, tra fine ‘800 e inizio ‘900, milioni di immigrati (tra cui anche numerosi italiani)

*Taylor, Ford e la catena di montaggio*

Il nuovo concetto di fabbrica pone nuovi problemi alle organizzazioni. Con lo svolgimento del lavoro su scala molto più ampia da parte di un più grande numero di lavoratori, si cominciò a pensare a come progettare e svolgere il lavoro in modo da incrementare la produttività raggiungendo al contempo la massima efficienza.

Alla fine del 1800, Frederik Taylor delinea l’insieme di metodi di gestione e direzione, noto con il nome di “organizzazione scientifica del lavoro”, basato sul principio del one best way, secondo cui esiste sempre e comunque un metodo unico e migliore per risolvere i problemi o compiere azioni di vario genere.   
Il motivo storico che spiega il nascere di questi principi risiede nella contraddizione tra le potenzialità produttive di un’industria ormai alle soglie della produzione di massa e i metodi ancora arcaici della produzione.

La figura di spicco in questo periodo è rappresentata sicuramente da Henry Ford. Con lui si raggiunge probabilmente la vera prima produzione di massa della storia grazie alla produzione dell’iconica Ford Model T: fu la prima vettura prodotta in grande serie utilizzando la tecnica della catena di montaggio riducendo i tempi di assemblaggio da 12 ore a soli 93 minuti, fu prodotta in 15.007.033 esemplari fino al maggio del 1927 arrivando a costare un terzo rispetto alle vetture della concorrenza, per velocizzare la produzione e diminuire i costi era disponibile in un solo colore e poco personalizzabile.

La fabbrica di Ford applicava i principi dell’organizzazione scientifica del lavoro di Taylor ed è stata tra le prime fabbriche a utilizzare il concetto di catena di montaggio: inizialmente l’esecuzione del progetto e delle sue parti era affidata alle capacità professionali dell’operaio, con la catena di montaggio, invece, i pezzi da lavorare venivano portati di fronte all’operaio, che eseguiva il suo compito, ripetendo sempre la stessa operazione. Ne consegue un deciso aumento della produttività.

Nonostante le resistenze contro questi metodi, che impoverivano di contenuto il lavoro privandolo di ogni creatività, il taylorismo e il fordismo si diffusero ampiamente nell’industria statunitense tra le due guerre, e anche in quella europea. I nuovi principi dell’organizzazione scientifica del lavoro divennero la base del moderno sistema di fabbrica.

(immagine Ford Modello T con frase Ford: "Avranno l'auto che vorranno, del colore che vorranno, purché sia nera.")

PRODUZIONE SNELLA

*Nascita*

I concetti che hanno caratterizzato la nascita e lo sviluppo della produzione di massa sono stati rivoluzionari per l’epoca e ancora adesso, specialmente nel contesto occidentale queste linee guida sono alla base delle metodologie e delle strategie di moltissime aziende.

La grande rigidità e la visione fortemente utilitaristica della produzione di massa non tiene però conto di aspetti legati alla qualità, al prezzo e al servizio al cliente; aspetti che, negli ultimi decenni, sono diventati di fondamentale importanza per sopraffare la concorrenza feroce che un’azienda di qualunque settore è costretta ad affrontare. Dopo quasi un secolo infatti, le teorie di Taylor e Ford venivano utilizzate e applicate da tantissime compagnie, un metodo sempre valido e funzionante, ma che, considerando anche dinamiche sempre più esigenti in tutti e tre i settori, non necessariamente risultava essere la migliore soluzione in tutti i contesti, anzi.

Dalla seconda metà del 1900 si iniziarono a vedere i primi cambiamenti di pensiero in ambito industriale. Tutto partì dal Giappone, in modo particolare dalla Toyota Motor Corporation. Dopo la seconda guerra mondiale la profonda crisi di quegli anni spinse verso una nuova ottica industriale che permettesse la riduzione dei costi a fronte di un grande aumento di produttività.

Sotto la guida di Taichii Ohno venne definito un nuovo modello di produzione industriale, che permettesse di rispondere alla necessità di flessibilità della produzione mantenendo comunque elevata la produttività aziendale. L'ascesa di Toyota fu possibile grazie al Toyota Production System (TPS), una metodologia innovativa incentrata sulla continua caccia agli sprechi e sull'importanza del coinvolgimento di tutti all'interno dell'azienda.

*Caratteristiche e benefici*

Il TPS è uno dei principali precursori della produzione snella. Alla base del TPS si trova l'idea di utilizzare le poche risorse disponibili nel modo più produttivo possibile con l'obiettivo di incrementare drasticamente la produttività della fabbrica. La disastrosa situazione del Giappone dopo la Seconda Guerra Mondiale spiega il perché ci sia stata la necessità di trovare una soluzione diversa rispetto a quelle già presenti all’epoca, le risorse pressoché inesistenti hanno obbligato la Toyota a muoversi in un’altra direzione, rivelatasi successivamente geniale e rivoluzionaria.

Nacque così il concetto di produzione snella (Lean Production), una filosofia che cambia completamente il punto di vista aziendale e sposta di gran lunga l'attenzione verso il cliente piuttosto che sulla produttività.

La Lean Production si focalizza sul concetto di valore e di spreco, inteso quest’ultimo come tutto ciò che consuma risorse senza creare valore per il cliente, e pertanto da eliminare.

La produzione snella è funzionale a rispondere a un cliente molto esigente, in un mercato in cui le gamme di prodotti iniziano a essere molto ampie e perciò i processi di produzione devono essere flessibili, rapidi e confrontarsi con volumi ridotti.

Un’analisi accurata della Lean Production meriterebbe un lavoro dedicato. L’intento, in questo elaborato, è presentarne i concetti base a livello generale, senza scendere nei dettagli di come questi vengano applicati al mondo della produzione industriale. Nel prosieguo di questo Capitolo, infatti, verrà mostrato come queste logiche siano state adottate, con risultati altrettanto vincenti, nel settore IT.

Verranno perciò presentati di seguito i principi fondamentali alla base del Lean Thinking che rivoluzionano la cultura e il modo di operare all’interno dell’azienda:

* Attenzione verso il cliente: la centralità del cliente è il punto di partenza e di arrivo di tutte le attività ed azioni messe in campo. Il dialogo con il cliente è fondamentale per identificarne i fabbisogni e definirne il valore.
* Importanza di tutto il personale: tutti i membri dell’azienda contribuiscono alla sua crescita. Il miglioramento del singolo si riflette su un miglioramento del sistema. Viene rivalutata la figura dell'operaio, non più mero esecutore di ordini, ma attivo all'interno dell'impresa, potendo anche intervenire sulla produzione stessa e modificarne l'andamento. Ciò richiede elevata capacità professionale da parte degli operai. Lo sviluppo e il sostegno della competitività aziendale, con l’ottenimento di risultati duraturi, si raggiunge con il continuo e costante allineamento del management e di tutte le persone che lavorano nell’azienda verso un obiettivo comune. Ampiamente superato il concetto dell’estrema divisione del lavoro proposta da Taylor.
* Lotta agli sprechi: I MUDA (termine giapponese per “sprechi”) consistono in tutte le attività, che impegnano risorse ed energie, che non aggiungono valore al prodotto o al servizio e non danno quindi valore al cliente. È tra i più importanti capisaldi del Lean Thinking, ed è inoltre uno dei più ritrovabili anche nell’applicazione di tali concetti al mondo IT.
* Lotti piccoli: chiamata in gergo tecnico SMED (Single Minute Exchange of Die), attraverso una produzione in piccoli lotti si risparmia tempo e denaro; permette inoltre di rispondere con molta più velocità e flessibilità alle variazioni di mercato, requisito fondamentale al giorno d’oggi. Applicando questo concetto al mondo IT vedremo come sarà possibile avere una visione completa del progetto in qualsiasi momento.

Il metodo risultò essere rivoluzionario, permise a Toyota di rialzarsi e iniziare una crescita continua che la portò, nel 2008, a essere la prima azienda automobilistica al mondo per numero di veicoli e fatturato.

LA PRODUZIONE NEL MONDO SOFTWARE

È possibile adottare il «Lean Thinking» anche nell’industria software? (da inserire prima di iniziare a parlare di Agile)

*Nascita e contesto*

Il termine “software” inizia a avere una divulgazione consistente sulla carta stampata solo agli inizi degli anni 50. A quel tempo però non vi era ancora un mercato corrispondente: i programmi erano sviluppati principalmente da utenti, non esisteva un movimento e un’industria di riferimento.  
Solo verso la fine degli anni 60 si può iniziare a intravedere la transizione che porterà la pratica dello sviluppo software da mera attività artigianale ad un approccio industriale e ingegneristico.  
Questo mutamento di prospettiva si può ricondurre all’aumento della complessità dei sistemi con cui abbiamo a che fare ogni giorno che ha portato alla necessità di utilizzare sistemi informatici sempre più evoluti anche in contesti critici (sistemi aerospaziali, strumenti per la difesa militare, ecc).  
Da allora, la ricerca su questi temi è stata estremamente prolifica sia in ambito industriale che accademico.

L’approccio iniziale è stato fortemente condizionato e ispirato dai concetti dell’industria meccanica e automobilistica. Il settore IT era agli albori ed è stato quindi naturale prendere spunto dalle idee che, dalla seconda rivoluzione industriale in poi, hanno visto i principali settori dell’industria occidentale fiorire.  
Cambia il campo da gioco, ma l’obiettivo rimane sempre quello: creare prodotti di qualità, contenendo i costi e generando profitto.

Inizialmente non è stato immediato applicare procedimenti industriali già conosciuti e sviluppati in un contesto completamente nuovo come quello della produzione software, ci sono ovviamente delle differenze strutturali (come l’immaterialità del prodotto principale, ovvero il codice eseguibile) che rendono unico, nell’ambito della produzione, il settore informatico (lato software), ma con il tempo si sono sviluppate alcune metodologie che riprendono i concetti sviluppati da Taylor, Ford o dalla Toyota.

*Metodologie*

In informatica si intende come “metodologia di sviluppo software” una suddivisione del processo di sviluppo del software per ottimizzarne il completamento.

Si sono sviluppate negli anni diverse metodologie che utilizzano procedimenti e strategie talvolta molto diversi tra loro. In questo elaborato saranno presentati il “modello a cascata” e la “metodologia agile”, che sono i modelli più utilizzati al giorno d’oggi. Si noteranno delle analogie con i modelli di produzione industriale proposti da Ford e da Toyota, motivo per cui ho ritenuto importante includerli nell’elaborato.

*Modello a cascata*

Il modello a cascata, waterfall model, è il modello di produzione software più tradizionale. Il processo di realizzazione è suddiviso in una serie di fasi da eseguire sequenzialmente.

La teorizzazione di tale processo, da parte di Winston Royce nel 1970, rappresenta un importante mutamento di prospettiva nella pratica dello sviluppo del software, concepito come processo industriale e non più con un approccio “code and fix” (tentativo a errori), sicuramente non adatto a un contesto di medie e grandi dimensioni.

Questo modello riprende la sequenza tipica della produzione di massa che si era sviluppata un secolo prima nell’ambito dell’industria meccanica e automobilistica. I passi che costituiscono il ciclo di vita del software, da eseguire sequenzialmente, riprendono il concetto della catena di montaggio delle fabbriche. Un approccio rigido e monolitico che prevede che l’output di una fase diventi l’input di quella successiva.

Possiamo identificare 5 fasi tipiche del modello a cascata:

1. Analisi: ha come scopo la definizione delle esigenze espresse dai clienti. Talvolta viene sottovalutata, ma è una fase fondamentale del progetto. Identificare in modo chiaro i requisiti che il software deve possedere evita future incomprensioni con il cliente che portano solitamente ad un ritardo nella consegna. La fase si conclude con la redazione di un documento, il Software Requirements Specification, contenente le funzionalità e le caratteristiche che il software deve avere per soddisfare le esigenze espresse dal cliente.
2. Design: partendo dal documento di Software Requirements Specification si realizza un progetto dell’intera architettura del software e anche dei singoli componenti che la costituiscono. Questa fase, che vede lavorare fianco a fianco membri di team diversi (programmatori, sistemisti, esperti di architetture di rete), è una di quelle che beneficia maggiormente di un approccio DevOps, come verrà meglio spiegato nel seguito di questo Capitolo.
3. Sviluppo: è la fase in cui comincia a prendere vita il software attraverso la scrittura del codice da parte degli sviluppatori. Si parte dall’implementazione dei singoli moduli per poi integrarli nel software finale.
4. Test: Finita la parte di sviluppo, il software deve essere sottoposto a diverse tipologie di test per verificare che le specifiche espresse dal cliente vengano soddisfatte dal funzionamento del programma. Le attività di verifica vengono svolte sia sui singoli moduli che sull’intera soluzione, per assicurarsi il corretto funzionamento del software complessivo a seguito dell’integrazione dei singoli componenti. Gli sviluppatori vengono informati di eventuali problemi e si attivano per risolverli.
5. Deployment: È la fase di rilascio in cui il software che ha superato la fase di test viene reso disponibile agli utilizzatori finali. In questa fase, conosciuta anche come deployment, il software, che ha superato con successo la fase di collaudo, viene rilasciato e reso disponibile agli utilizzatori finali. Ciò comporta anche la fase di installazione e di formazione che può essere più o meno elaborata a seconda della tipologia di software realizzato (applicazione da installare su ogni client o applicazione web).

(figura)

Dagli anni 80 il modello ha iniziato a essere soggetto di numerose critiche, molte tra le figure di riferimento del settore IT cominciavano a sostenere che il modello non riuscisse più a stare al passo con l’evoluzione del mondo software e dei linguaggi di programmazione.

Il waterfall model ha infatti alcuni limiti importanti. Limiti non dovuti ad una cattiva esecuzione del modello, ma intrinsechi, causati dalla struttura del modello e dalla logica con cui è stato pensato.

* Scarsa visibilità dell’andamento del progetto: l’approccio molto rigido e monolitico non permette di avere una visione di insieme. Si riesce a valutare e controllare una fase per volta. Questo problema porta spesso a un errore nella valutazione delle tempistiche necessarie per portare a termine il lavoro.
* Poca flessibilità: procedendo verso il fondo della cascata è difficile, se non impossibile, tornare indietro o fermarsi senza incappare in consistenti ritardi. Questo significa che è praticamente nulla la capacità di reagire a eventuali imprevisti o difficoltà a corso d’opera. Se, per esempio, si scopre che l’ambiente hardware per effettuare i test è stato configurato erroneamente solo nel momento in cui si svolgono i test è ovvio che si crei un collo di bottiglia che porta al collasso del sistema
* Difficoltà nel gestire nuove richieste del cliente: abbiamo già parlato di come, nella fase di analisi dei requisiti, si possano facilmente creare incomprensioni con il cliente. Se cambiano anche solo minimamente le richieste del cliente dalla fase di sviluppo in poi è molto facile che per assecondare la richiesta si creino notevoli ritardi. È importante riuscire a gestire in qualunque momento eventuali cambiamenti e rispondere in modo flessibile e rapido.
* Aumento considerevole dei costi: la rigidità di questo modello porta necessariamente a sprechi e cambiamenti di strategia dovuti a iniziali errori di progettazione. Ciò porta a un aumento di costi che in alcuni casi può diventare insostenibile.

Il waterfall model è al giorno d’oggi in parte superato, ma rimane un punto di riferimento importante, lo scheletro e la base di tanti dei modelli contemporanei.

*Agile*

È possibile adottare il Lean Thinking anche all’industria software?

Da quando la produzione di software ha iniziato ad essere affrontata con approccio industriale, anche a causa dei tantissimi settori in cui nasceva la necessità di avere programmi informatici che svolgessero funzioni cruciali, la crescita e l’espansione del settore IT non si sono più fermate. Il problema di dover produrre in tempi sempre più brevi software sempre più grandi e complessi aveva già messo sotto scacco il waterfall model ed è stata chiara la necessità di dover trovare nuovi modelli che rispondessero meglio alle richieste sempre più esigenti del mercato.

Nel corso degli anni 90 si evidenziò quindi un graduale cambiamento di approccio allo sviluppo del software. Si adottarono e perfezionarono diversi metodi light di sviluppo software, che utilizzavano metodologie agili e iterative per avere un miglior controllo sul prodotto. Tra i principali framework nati in quel periodo si ricorda Scrum (1995), ancora oggi molto utilizzato, Unified Process e Dynamic Systems Development Method (1994), Rapid Application Development (1991), Crystal Clear e Extreme Programming (XP) (1996) e Feature-driven Development (1997).

Ma l’inversione di rotta più decisa si ha nel 2001 quando un gruppo di sviluppatori e guru del software pubblicò il “Manifesto per lo sviluppo agile del software” che descriveva una serie di principi in base ai quali i requisiti e le soluzioni proposte si evolvono progressivamente e costantemente durante il ciclo di vita del progetto, attraverso la collaborazione di tutto il team.  
Nei 12 principi del Manifesto Agile ritroviamo concetti che erano alla base dello sviluppo della Lean Production proposta da Toyota:

1. Soddisfazione del cliente 2. Accettare il cambiamento dei requisiti

3. Rilasci frequenti 4. Biz e Dev lavorano insieme

5. Partecipazione attiva di individui motivati

6. Comunicazione chiara 7. Software funzionante

8. Ritmo costante 9. Eccellenza tecnica

10. Semplicità 11. Auto organizzazione del team

12. Miglioramento continuo (da aggiungere a principi Lean Thinking)

L’aggancio culturale al Lean Thinking è evidente, ma l’impatto pratico che i nuovi modelli di stampo agile hanno avuto nel mondo della produzione software è stato, se possibile, ancora più rivoluzionario di quello rilevato nel mondo dell’industria manifatturiera. Nel settore IT, infatti, la necessità di sposare nuovi modelli è stata ancora più vincolante. La produzione di massa viene ancora adottata in alcuni contesti industriali, specialmente occidentali, e in particolari contesti è ancora una soluzione valida; nel mondo della produzione software questo compromesso non ha trovato spazio, le metodologie agili sono imprescindibili per chiunque voglia, al giorno d’oggi, produrre software di qualità rimanendo competitivo.

A differenza dell’approccio lineare del waterfall model, Agile presuppone che il progetto venga sviluppato sin dall’inizio in maniera incrementale e iterativa. Ogni iterazione, chiamata anche sprint, è un piccolo progetto a sé stante e deve contenere tutto ciò che è necessario per rilasciare un piccolo incremento nelle funzionalità del software: analisi, design, sviluppo, test e documentazione. Sostanzialmente le fasi che nel modello a cascata venivano eseguite una sola volta sequenzialmente, con Agile vengono ripetute ad ogni iterazione e non riguardano l’intero progetto ma una o più funzionalità. A fine di ogni sprint vengono consegnate le feature sviluppate al cliente in modo da ricevere costanti feedback, scongiurando scenari tipici dei modelli tradizionali in cui i programmatori si rendevano conto dell’inefficacia del progetto solo a lavoro finito.

(due – tre figure: tabella differenza a-w, slide differenza a-w, foto scaricata)

*DevOps*

*Nuove sfide*

Il paradigma Agile ha consentito alle aziende IT di fronteggiare le sfide sempre più impegnative poste dal mercato software. Ma la crescita delle infrastrutture IT è impressionante, si sta raggiungendo un livello di complessità che, fino a qualche anno fa, era inimmaginabile. L’industria del software è messa alla prova costantemente con sfide sempre più impegnative ed ecco perché i leader del settore, negli ultimi anni, hanno compreso che “fermarsi” ai modelli più nuovi, Agile ad esempio, non sarebbe più stato sufficiente.

È il caso di approfondire la tematica della complessità crescente che spinge a trovare nuove soluzioni. Secondo un sondaggio di Gartner (chi è) le infrastrutture IT esistenti risultano essere “very complex” nel 54% dei casi, le problematiche maggiori sono da rilevare nei processi IT e nel proliferare di tool tecnologici sempre più sofisticati., problematiche che rendono difficoltoso sia il rilascio di software di qualità sia il mantenimento di servizi IT efficienti e efficaci sul piano del business. Secondo i pareri raccolti dal sondaggio una delle aree più critiche risulta essere quella della comunicazione e della relazione tra Development e Operation con il 47% degli intervistati che definisce tale rapporto come “un-collaborative”.

Comprendere le mansioni dei team Development e Operation è imprescindibile per proseguire nel presentare DevOps. Ritengo perciò che sia utile fare questa specificazione. Infatti, se il ruolo di un Developers (i programmatori, ovvero chi sostanzialmente si occupa della parte di sviluppo del codice) è noto praticamente a tutti, non si può fare lo stesso discorso per l’area Operation.   
Possiamo dire che chi lavora nell’area dell’IT Operation cura tutto ciò che sta intorno all’applicazione, ovvero si occupa di creare e mantenere le infrastrutture e le architetture su cui gli sviluppatori potranno poi programmare e su cui il software verrà poi messo in produzione. Per essere ancora meno generici è possibile specificare due aree di competenza degli Operation:

* Network Infrastructure: progettazione, configurazione e manutenzione dell’architettura di rete (router, DNS server, firewall)
* Server and Device Management: gestione e manutenzione di tutti i dispositivi (server, PC) che fanno parte del progetto. Questo prevede, per esempio, l’installazione e la configurazione di sistemi operativi e applicazioni necessarie sui vari devices.

L’importanza dell’area Operation è cresciuta e sta crescendo proporzionalmente all’aumento della complessità delle soluzioni software che le aziende devono produrre. Nel contesto odierno, infatti, non è neanche lontanamente immaginabile creare un prodotto di qualità servendosi esclusivamente di bravi e veloci programmatori, che magari utilizzano anche Agile. È fondamentale che i servizi e le infrastrutture che creano il sistema su cui programmatori (e non solo) lavorano per il corso del progetto siano ottimizzate, efficaci ed efficienti.

In molte organizzazioni lo sviluppo del software e la gestione dei sistemi sono separati, non c’è collaborazione tra le due aree, come testimonia anche il sondaggio di Gartner. Questo crea una discontinuità tra i metodi di lavoro e le finalità dei due team: gli sviluppatori si concentrano su produrre continuamente del codice in modo da consegnare il più velocemente possibile al cliente del nuovo materiale, mentre gli operation puntano sulla stabilità e la qualità delle infrastrutture. La mancanza di comunicazione tra le due aree fa sì che nessuno dei due team riesca a svolgere in modo adeguato il lavoro.  
Comincia a essere più chiaro che un’azienda IT di successo debba puntare sulla collaborazione di queste due aree.

È da queste necessità, ma non solo, che nasce DevOps.

*Nascita e contesto*

Il “movimento” DevOps comincia a prendere piede nel 2009 in Belgio dopo una serie di DevOps day, ma è nel 2011 che si colloca storicamente la nascita vera e propria di questa metodologia.

DevOps è una metodologia che, attraverso l’interdipendenza e la collaborazione del team Dev (Development) e Ops (Operations), consente di automatizzare il rilascio del software all’interno della catena di produzione col fine di ottenere un software di qualità superiore in modo più rapido, riducendo il time-to-market e venendo il più incontro possibile al cliente.

Come si inserisce DevOps nel contesto IT di quegli anni?

Pensare a DevOps come opzione alternativa a Agile è profondamente sbagliato. DevOps si basa sugli ideali di Agile e li estende oltre al codice concentrandosi sull’intero servizio offrendo una soluzione più completa a livello di business. Se Agile si fondava su valori, metodi e pratiche, DevOps aggiunge gli strumenti; se con Agile viene teorizzato che sottoporre il codice a continui test produce un risultato qualitativamente migliore, DevOps mi fornisce gli strumenti per implementare un sistema automatizzato di testing all’interno della mia catena.  
Non solo, Agile è dentro DevOps: sul fronte dello sviluppo si sposano pratiche tipiche di Agile, scegliendo un framework di sviluppo come Scrum ad esempio.

*Le pratiche del DevOps*

Come affermato precedentemente, DevOps aggiunge gli strumenti alle pratiche di stampo Lean in modo da applicarle all’intero processo e creare una catena di produzione sempre più efficace. Proseguendo in questo Capitolo si andranno perciò ad analizzare le principali best-practice che caratterizzano questa metodologia.

*Utilizzo di un Version Control System*

L’utilizzo di un sistema per il controllo di versione è una delle pratiche più comuni e utilizzate della produzione software da molto prima che emergesse il fenomeno DevOps.

Il controllo di versione è una pratica che permette di tracciare i cambiamenti prodotti sul codice sorgente da chiunque stia lavorando al progetto. Vengono registrate tutte le versioni del codice che viene salvato in modo da poter facilmente tornare a una versione precedente per necessità varie.

Le dimensioni crescenti dei progetti software hanno portato ad avere sempre più sviluppatori lavorare sullo stesso progetto. Il controllo di versione permette ad un team, i cui membri sono geograficamente distanti, di programmare contemporaneamente senza collisioni: ciò è possibile grazie ai branch, che permettono di creare un ramo diverso dal master del progetto per apportare le modifiche e solo alla fine avviene l’integrazione con il master.

Ovviamente anche DevOps sposa questa pratica, estendendone addirittura gli utilizzi: infatti non ci si limita più a mettere sotto controllo di versione il codice sorgente, ma anche altre informazioni utili ad altre parti del progetto, ad esempio script contenenti la configurazione dei vari ambienti. Questo aspetto sarà approfondito nel corso del Capitolo parlando di Infrastructure as a Code.

*Concetto di Deployment Pipeline*

Uno dei primi concetti chiave del DevOps è quello di Deployment Pipeline. Si intende la catena di produzione del software, più nello specifico delle sue singole feature, dalla progettazione, fino alla consegna al cliente. La si può definire il campo da gioco del DevOps. Composta da vari stage, l’obiettivo è rendere il percorso lungo la pipeline il più rapido a automatizzato possibile in modo da velocizzare la realizzazione del lavoro.

“Automazione” è uno dei mantra del DevOps, la leva che si vuole utilizzare, grazie ai sofisticati strumenti attuali, per rendere la produzione software più snella possibile. Si vuole creare un production-like environment (definizione production-like environment?) in ogni stage del processo attraverso un build mechanism che crei tutti gli ambienti (sviluppo, test, produzione, ecc) velocemente (attraverso il configuration management, approfondito più avanti) e sin da subito senza la necessità di doverli richiedere aprendo un ticket che può far perdere anche settimane di lavoro. Un simile approccio porta evidenti vantaggi: ad esempio, se l’ambiente per il testing è a disposizione degli sviluppatori dall’inizio essi potranno testare giornalmente il loro lavoro e risolvere sul nascere eventuali problemi, evitando di trovarsi con grosse parti non funzionanti in là con il tempo. Un concetto ancora più estremo degli sprint Agile.  
  
(figura deployment pipeline)

Come creare una Deployment Pipeline efficiente?

*Continuous Integration*

Negli ultimi anni, con l’avvento di nuovi framework per lo sviluppo software, l’integrazione continua (Continuous Integration, abbreviata CI) è diventata una pratica diffusa in diversi contesti, in primis nell’ambito del DevOps che ne fa una delle sue principali best-practice.

La CI si occupa di risolvere il problema della divergenza tra versioni di codice sviluppate da diversi membri del team. Fino a qualche anno fa, infatti, le componenti software venivano sviluppate in modo indipendente e quando era il momento di integrarle tra loro si scoprivano bug e numerose problematiche nell’unire pezzi di codice incompatibili con l’insieme del software.  
C’è la necessità di risolvere questo problema sul nascere, evitando di riscontrare difficoltà notevoli in fasi più avanzate del progetto; perciò ogni volta che inseriamo del codice nel nostro Version Control System esso dev’essere integrato il prima possibile e automaticamente da una macchina appositamente configurata che eseguirà la compilazione del codice e lancerà la suite di test. Per favorire questa operazione gli sviluppatori devono imparare a lavorare con piccole porzioni di codice alla volta (small batch size) integrandolo il più spesso possibile nella deployment pipeline. Il servizio utilizzato per ottenere la Continuous Integration riceve il codice, crea automaticamente una build e avvia i test.  
(figura continuous integration)

*Creazione sistema automatizzato di test*

Parlare del fatto che si è visto che è necessario automatizzare dei test e far vedere che ciò è necessario in diverse fasi della catena. Presentare varie tipologie di test

*Continuous Delivery*

Estensione del concetto di Continuous Integration