



UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

*Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica*

Elaborato teorico di Progettazione e Sviluppo di Sistemi Software

DevOps

Anno Accademico 2018/2019

A cura di:

Formisano Raffaele M63000912

Napolano Giuseppe M63000856

Pugliese Nicola Alessandro M63000847

Romito Giuseppe M63000936

Indice

1 Metodologie Agili	1
1.1 Introduzione al modello agile	1
1.2 Esempi di metodologie agili	4
1.2.1 Extreme programming	4
1.2.2 Scrum	5
1.3 Miti da sfatare	7
2 DevOps	9
2.1 Perchè DevOps	9
2.2 Architettura di riferimento	11
2.3 DevOps e microservizi	13
2.4 DevOps e cloud computing	15
2.5 Adottare DevOps:tecniche e strumenti	17
2.5.1 Tecniche	17
2.5.2 Configuration Management	18
2.5.3 Approfondimento GIT	20
Conclusioni	23

Elenco delle figure

1.1	I valori del manifesto agile	2
1.2	I principi del manifesto agile	2
1.3	Modello di sviluppo Agile	3
1.4	Valori di XP	4
1.5	Attività ed artefatti di Scrum	6
2.1	DevOps	10
2.2	Architettura di riferimento DevOps	11
2.3	Registrazione istanze e load balancing	13
2.4	Configuration Management	19
2.5	VC centralizzato	20
2.6	VC Distribuiti	20
2.7	Versioni di componenti come liste di Delta	21
2.8	Versioni di componenti gestite da Git	21
2.9	I tre stati dei file git	22

Capitolo 1

Metodologie Agili

1.1 Introduzione al modello agile

Primo compito dell'ingegneria del software è quello di portare ad un approccio disciplinato nella produzione di software mediante la definizione di "modelli di sviluppo" che descrivano le attività e stabiliscano l'ordine in cui esse devono essere svolte, garantendo in tal modo la qualità del prodotto finale. I modelli di sviluppo tradizionali, tra cui ricordiamo il Waterfall ed i suoi derivati, con la loro eccessiva rigidità e richiesta di documentazione formale, sarebbero poco efficaci nel caso di un progetto che presenta requisiti poco chiari e instabili e pertanto porterebbero con buona probabilità ad un fallimento con impossibilità di rispettare deadlines e budget. Con la loro linea più leggera e flessibile, i modelli agili, sono preferibili quando il lavoro non è soltanto **complicato** ma è **complesso** poiché prevede un'incertezza non eliminabile sui requisiti e richiede pertanto un approccio adattivo più che predittivo in grado di accogliere e rispondere adeguatamente ai cambiamenti. Il manifesto proposto di seguito elenca i quattro valori e i dodici principi base su cui si fonda la metodologia agile e quindi anche tutti gli specifici metodi da essa derivati tra cui XP e Scrum che analizzeremo in dettaglio nel seguito.

<http://agilemanifesto.org/>

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck - Mike Beedle - Arie van Bennekum - Alistair Cockburn - Ward Cunningham - Martin Fowler - James Grenning - Jim Highsmith - Andrew Hunt - Ron Jeffries - Jon Kern - Brian Marick - Robert C. Martin - Steve Mellor - Ken Schwaber - Jeff Sutherland - Dave Thomas

FIGURA 1.1: I valori del manifesto agile

Principles behind the Agile Manifesto

<http://agilemanifesto.org>

We follow these principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

FIGURA 1.2: I principi del manifesto agile

Si può quindi ridefinire l'Agile come un mindset che con i suoi valori e le sue pratiche punta a massimizzare il valore più che la mole di quanto prodotto, portando ad uno sviluppo di tipo

iterativo ed incrementale che promette di ridurre i rischi creando prodotti distribuibili e valutabili già durante la fase di sviluppo. La parola chiave del modello agile è **comunicazione**. Agile incoraggia infatti la comunicazione diretta tra i membri dei team di sviluppo per realizzare una condivisione delle esperienze e della conoscenza, ma anche la comunicazione diretta col cliente. È previsto infatti un alto grado di coinvolgimento e collaborazione col cliente che permette di massimizzare il feedback grazie all'elevata frequenza dei rilasci di versioni intermedie. Minore importanza è attribuita alla documentazione che spesso è creata a partire dal codice già scritto, mentre si punta ad ottenere il prima possibile software funzionante che sarà poi soggetto ad un continuo lavoro di *refactoring*. Il personale non è più suddiviso in una tradizionale struttura piramidale, ma è organizzato in team di dimensioni più piccole in cui non ha particolare risalto il concetto di gerarchia, ma in cui si incoraggia compattezza e affiatamento e si stimola la motivazione degli individui. Talvolta è prevista inoltre una programmazione di coppia. Essenziale è la semplicità in tutte le attività di progettazione, modellazione e soprattutto durante la produzione di codice, per aumentare la leggibilità e agevolare le successive fasi di correzione, modifica e quindi conseguentemente anche l'adattamento ai nuovi requisiti. La figura successiva mostra la natura iterativa, incrementale e interattiva della metodologia agile.

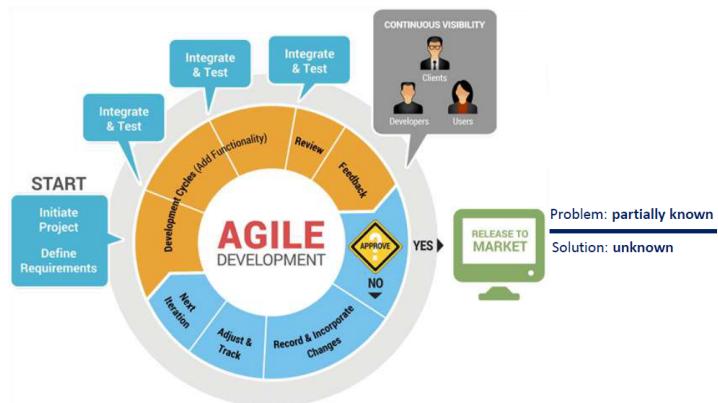


FIGURA 1.3: Modello di sviluppo Agile

1.2 Esempi di metodologie agili

1.2.1 Extreme programming

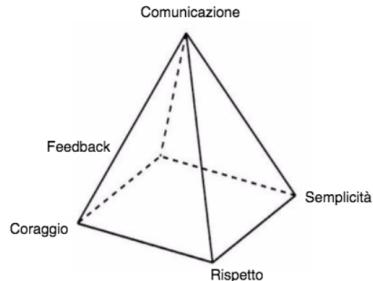


FIGURA 1.4: Valori di XP

Come suggerisce il nome, l'attività di codifica è al centro di questa metodologia che si basa sul continuo refactoring, ad ogni iterazione, di una porzione codice e senza doversi preoccupare delle fasi successive. I valori su cui si basa sono:

- **Comunicazione:** tra i membri del team e con il cliente.
- **Semplicità:** mantenere il design più semplice possibile in modo da evitare sprechi ed agevolare il lavoro di modifica.
- **Feedback:** feedback continuo indispensabile per il continuo miglioramento e per avere maggiore flessibilità nei confronti del cliente.
- **Coraggio:** coraggio nel fare scelte e nel prendere importanti decisioni, ma anche nell'essere pronti ad ammettere quelle sbagliate.
- **Rispetto:** tutti gli attori coinvolti nel progetto hanno uguale importanza e ad ognuno deve essere data la possibilità di esprimere liberamente le proprie opinioni. Questa metodologia prevede di adottare una serie di pratiche di codifica, di design e organizzative che puntano a rendere il processo di sviluppo più efficiente e veloce possibile:
- **Coding standards:** la codifica deve essere omogenea a degli standard adottati per semplificare la leggibilità. Il codice assume infatti il ruolo di importante veicolo di comunicazione e per questo deve essere facile per ogni sviluppatore comprendere e modificare il codice scritto da altri.
- **Testing:** E' previsto un testing rapido e continuo. Talvolta si parla di Test Driven Development, pratica che prevede di derivare i casi di test dai requisiti raccolti e successivamente procedere alla produzione di codice che superi il test.

- **Refactoring:** la riprogettazione ed eliminazione di parti superflue richiede una rielaborazione adattiva della struttura del codice.
- **Simple design:** la soluzione adottata è sempre la più semplice e prevede la realizzazione solo di quanto strettamente necessario.
- **Short releases:** ogni iterazione si conclude con il rilascio di una versione del software funzionante che implementa un nuovo scenario d'uso (stories).
- **Continuous integration:** con l'utilizzo di una piattaforma di versioning che provvede ad allineare frequentemente il lavoro indipendente degli sviluppatori.
- **Pair programming:** su una sola postazione lavorano due programmatore che assumono alternativamente il ruolo di conducente e osservatore.
- **Onsite customer:** il cliente è coinvolto come fonte principale per stabilire i criteri di validazione e quindi partecipa alla stesura dei casi di test.
- **Metaphor:** strumento che descrive e guida la storia di sviluppo del sistema.
- **Collective ownership:** tutti gli sviluppatori possono mettere mano a qualsiasi porzione di codice rispettando le regole condivise.
- **40 hour week:** limite di 40 ore a settimana per preservare la qualità del lavoro.
- **Planning game:** sulla base degli obiettivi da raggiungere descritti dagli utenti, gli sviluppatori stimano un tempo realizzativo ed una priorità per ogni caso d'uso. Sulla base di ciò si procede ad una accurata pianificazione del lavoro e ad una continua misurazione dei progressi fatti confrontando le attività eseguite e il piano stesso.

1.2.2 Scrum

Scrum è un approccio Agile di tipo iterativo incrementale nato negli anni 90 che mira ad ottenere il massimo valore nel più breve tempo possibile. Il cliente specifica le funzionalità da realizzare, o aggiungere al sistema, con le relative priorità e ogni tre o quattro settimane ha la possibilità di ispezionare una nuova versione del software. La versione esaminata permette di decidere se iterare un nuovo ciclo (sprint) o se procedere al rilascio. Si basa su tre pilastri: trasparenza, ispezione, adattamento; e cinque valori: impegno, coraggio decisionale, focus sugli obiettivi a breve termine, apertura, rispetto. La **trasparenza** tra i membri del team e col cliente prevede di non celare problematiche ma di comunicare con chiarezza l'andamento delle attività in modo da ottenere non solo maggiore coesione nel gruppo di lavoro ma anche un rapporto di fiducia col cliente. L'**ispezione** continua degli artefatti permette di rilevare in anticipo rischi e criticità evitando problemi nel rispettare scadenze e consegne.

Conseguentemente un lavoro di **adattamento** deve essere fatto ogni volta emerge che qualcosa sta andando fuori progetto. Lo scrum team è un gruppo auto-organizzato composto dalle figure del *Product Owner*, *Scrum Master*, e *team di sviluppo*. Il **P.O.** come recita la "Scrum Guide", ha la responsabilità di massimizzare il valore del prodotto e del lavoro svolto dal team. Egli infatti ha il compito di gestire e interfacciarsi con gli *stakeholders* e definire una linea di sviluppo che permetta di realizzare un prodotto che risponda alle necessità degli utenti finali. Comunica quindi al team quali sono le funzionalità da implementare, le priorità e i tempi di rilascio.

Lo **Scrum Master** ha il compito di guidare l'adozione e l'applicazione del metodo scrum aiutando a comprenderne i valori, rimuovendone gli ostacoli e facilitandone gli eventi del processo come le riunioni periodiche di aggiornamento e confronto. Assolve al ruolo di coach per il team di sviluppo, motivandolo, proteggendolo da influenze esterne e supportando la cooperazione.

Il **Team di sviluppo** è un gruppo auto-organizzato che decide in autonomia di trasformare il *Product Backlog* in incrementi rilasciabili. È un gruppo cross-funzionale che deve saper svolgere tutte le attività necessarie allo sviluppo e atomico poiché non permette suddivisione in sottogruppi. È prevista totale parità tra i membri e anche se alcuni possono avere competenze specialistiche, l'intero team condivide in egual misura le responsabilità del lavoro di sviluppo nel complesso.

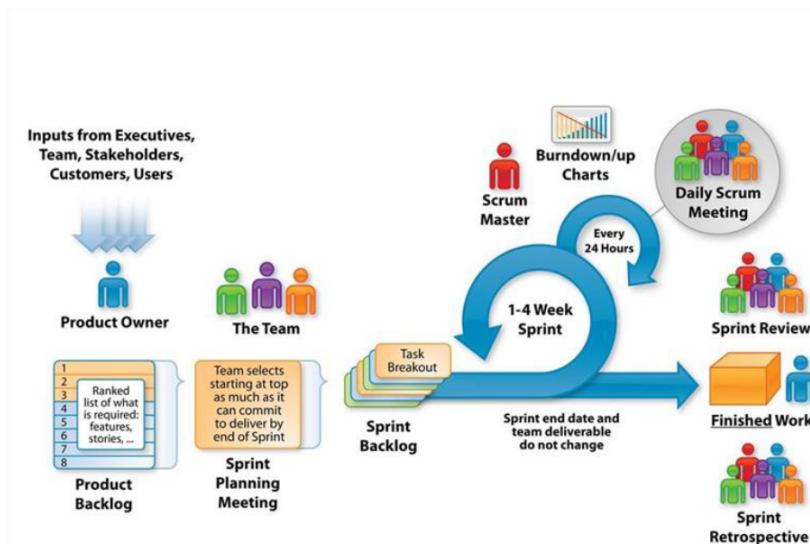


FIGURA 1.5: Attività ed artefatti di Scrum

La figura precedente mostra le attività e gli artefatti del metodo scrum. Un processo scrum si configura come una serie di **Sprint**, ovvero periodi di durata costante che varia da due a quattro settimane. In ogni sprint si eseguono le attività di design codifica e test necessarie per estrarre una funzionalità implementabile dal product backlog e realizzarne un codice.

finito eventualmente rilascibile con una nuova versione del software. Il processo è guidato dai seguenti artefatti:

- **Product Backlog:** una lista di requisiti espressi in forma di user stories ad ognuno dei quali è associata una priorità dal P.O. che è ridefinita all'inizio di ogni sprint.
- **Sprint Backlog:** breve descrizione del lavoro da fare durante un singolo sprint, modificabile da ogni membro del team che su base volontaria può scegliere e prenotare un'attività da svolgere.

Sono inoltre necessari dei meetings periodici di pianificazione e coordinamento ad ognuno dei quali è assegnato un tetto di ore prestabilito:

- **Sprint Planning :** fase preliminare ad uno sprint in cui si decide cosa deve essere fatto consultando e aggiornando il product backlog e in cui si definisce lo sprint backlog.
- **Daily Scrum :** meeting giornaliero di circa quindici minuti in cui ogni dipendente spiega cosa ha fatto dal meeting precedente, cosa intende fare nelle prossime 24 ore e quali sono le problematiche che prevede di dover affrontare.
- **Sprint Review:** si ispeziona l'incremento prodotto nello sprint e se necessario si aggiorna il product backlog.
- **Sprint Retrospective:** non ispeziona il prodotto bensì il processo. Il team esamina quelle che sono state le proprie difficoltà e criticità per migliorare la produttività nel prossimo sprint.

1.3 Miti da sfatare

In base a quanto detto sull'approccio Agile e dopo aver visto in dettaglio due esempi di metodologie agili, si può ritenere utile raccogliere e sfatare i principali falsi miti o luoghi comuni :

1. **Agile = nessuna documentazione.** L'agile non fornisce indicazioni prescrittive per cui non esclude la creazione di documentazione. Il processo è infatti orientato alla produzione di valore e quindi la documentazione va prodotta se e quando rappresenta un valore per il cliente.
2. **Agile = Anarchia.** Agile richiede infatti maggiore disciplina poichè, come visto, porta ad una redistribuzione delle responsabilità nel gruppo di lavoro. Non è più l'autorità, bensì l'insieme dei valori e dei principi ad "autodisciplinare" i membri dei team.

3. **Agile = Silver Bullet.** Agile non rappresenta la risposta definitiva per tutte le problematiche di produzione e gestione di un software, sebbene migliora la collaborazione, fornisce un maggior controllo sullo stato di avanzamento e riduce i rischi del processo di sviluppo.
4. **Agile = No Pianificazione.** Agile evita una fase iniziale di iper-pianificazione, ma prevede un raffinato processo di pianificazione durante tutto il ciclo di sviluppo. Infatti si realizzano piani che riguardano diversi traguardi temporali aggiornati costantemente che costituiscono la “Planning Onion”
5. **Agile = No Progettazione.** La progettazione non è limitata ad una sola fase, ma è distribuita durante tutto il processo di sviluppo a supporto del continuo lavoro di refactoring.
6. **Agile = Solo progetti di piccole dimensioni.** Agile è infatti scalabile su progetti di qualsiasi dimensione, ed in particolare consente di affrontare un progetto di grandi dimensioni scomponendolo in piccole iterazioni che lo rendono più facile da gestire.

Capitolo 2

DevOps

2.1 Perchè DevOps

Nei capitoli precedenti si è visto come l’evoluzione dai lenti e ingombranti modelli tradizionali verso il modello agile ha permesso una velocizzazione del processo di sviluppo e soprattutto un ormai indispensabile miglioramento in termini di capacità di accogliere e gestire i cambiamenti di requisiti. Tuttavia i modelli di sviluppo agili tra cui gli stessi XP e Scrum visti prima, apportano benefici circoscritti solo alla fase di sviluppo non considerando e coinvolgendo in alcun modo le attività di messa in produzione. In tal contesto, DevOps è una metodologia che estende gli ideali dell’approccio agile all’operato di tutti i dipartimenti dell’azienda e mira a ridurre quindi la distanza e le difficoltà di collaborazione tra gli ingegneri del lato *Development* e quelli del lato *Operations*.

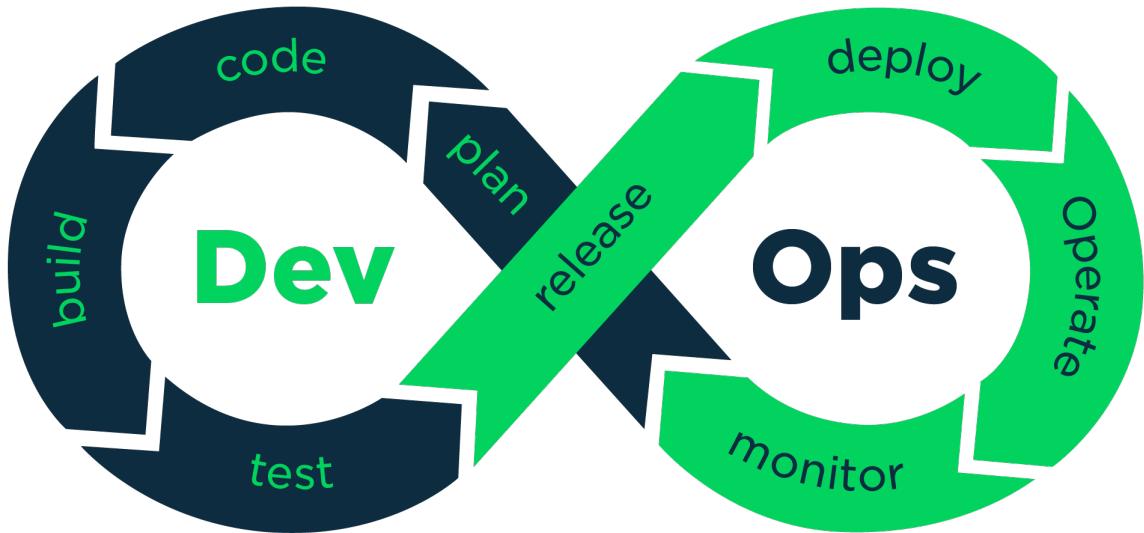


FIGURA 2.1: DevOps

Ma cosa si intende per *Operations*? Attività di cui i team “operativi” di un’azienda sono tradizionalmente responsabili sono per esempio:

- **Installazione e gestione dell’infrastruttura hardware**
- **Configurazione di server e reti**
- **Monitoraggio**
- **Controllo e gestione della sicurezza del sistema**
- **Acquisto di risorse IT**
- **Applicazione di politiche di backup e ripristino**
- **Risposta alle interruzioni**
- **Tracciamento delle risorse**

In aziende con impostazione tradizionale possono addirittura verificarsi conflitti di interessi tra i team di sviluppo e i team operativi. I primi infatti sono portati a voler sviluppare nuove funzionalità per il sistema, i secondi invece non accettano di buon grado i cambiamenti che possono aumentare il rischio di guasti in produzione. Adottare DevOps non significa quindi adottare semplicemente un programma che migliori e velocizzi i processi di sviluppo, bensì introdurre una cultura aziendale che coinvolga tutti i dipendenti, allineando l’operato dei dipartimenti verso obiettivi e interessi comuni . In particolare la figura 2.1 mostra come in un contesto DevOps il lavoro del team di sviluppo non possa prescindere da quello del team operativo e viceversa.

2.2 Architettura di riferimento

In questo capitolo si analizzerà un template che può aiutare i professionisti a disegnare una piattaforma DevOps che includa persone, processi e tecnologie. In particolare verrà presentata un'architettura di riferimento che distribuisce le sue funzionalità a dei componenti che possono essere concepiti per lavorare indipendentemente oppure organizzati in gruppi. Si può osservare l'architettura proposta in figura 2.2 dal punto di vista delle capacità di base che intende fornire, che nell'evoluzione dal piano astratto verso una forma più concreta, coinvolgono un set di pratiche, personale competente, e strumenti di automazione.

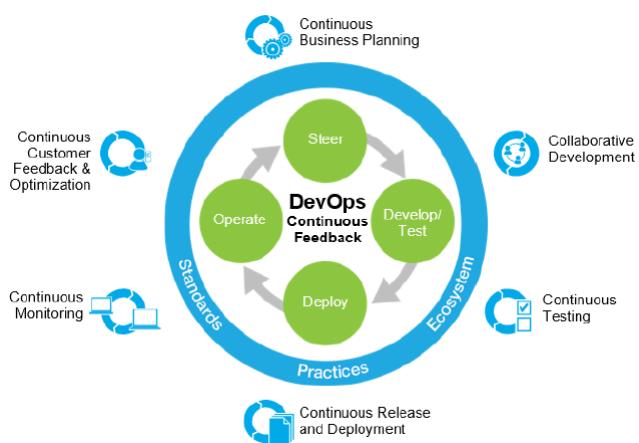


FIGURA 2.2: **Architettura di riferimento DevOps**

Analizziamo in dettaglio le componenti:

Steer: è la fase di orientamento che consiste nello stabilire e focalizzarsi sugli obiettivi di business, aggiornandoli costantemente sulla base del feedback ricevuto dal cliente. Gli approcci tradizionali sono troppo lenti per permettere un continuo ed agile adattamento ai fabbisogni del cliente anche a causa del fatto che i team tendono a lavorare indipendentemente. Alcuni team inoltre, abituati a puntare alla quantità piuttosto che alla qualità, tendono a considerare la pianificazione come un inutile rallentamento del proprio flusso di lavoro piuttosto che come un'attività che guida alla produzione di valore con tempi rapidi. DevOps invece prevede un alto grado di cooperazione tra i team per stabilire collaborativamente gli obiettivi di business e poterli adattare ai feedback mediante un processo di “*continuous business planning*”. Questa attività permette inoltre di identificare gli sprechi nel processo di sviluppo permettendo ai team di lavorare in maniera più efficiente ed ottenere in tal modo un miglior bilancio di tempi e costi.

Develop/test: coinvolge due pratiche che servono ad assicurare la qualità dello sviluppo:

- **Sviluppo collaborativo:** Il lavoro da compiere che porta al rilascio di un software coinvolge un ampio numero di team cross-funzionali. Essi sono composti da professionisti con diverse competenze specifiche che si trovano a dover lavorare su varie piattaforme e distribuiti in locazioni diverse. Lo sviluppo collaborativo permette loro di lavorare insieme poiché prescrive un set di pratiche e una piattaforma condivisa che essi possono usare per creare e rilasciare software. Una pratica fondamentale che caratterizza lo sviluppo collaborativo è la “*continuous integration*” con cui i dipendenti, aiutati dall'utilizzo della piattaforma, integrano frequentemente il loro lavoro con quello degli altri membri. Questa pratica, eredità del movimento agile, permette in caso di progetti complessi, di compiere una routine di test ogni qualvolta il sistema viene integrato con una nuova funzionalità riuscendo in questo modo a localizzare e risolvere immediatamente gli errori.
- **Continuous Testing:** durante tutto il ciclo di vita sono eseguiti frequenti e rapidi cicli di test che permettono di avere una costante garanzia della qualità di quanto prodotto. Il testing che non è più relegato e circoscritto a una fase finale del ciclo di vita, è agevolato da strumenti che ne permettono l'automatizzazione e da strumenti di service virtualization in grado di ricreare un ambiente simile a quello di produzione.

Deploy: dopo aver definito la continuous integration, si trova, come passo successivo e complementare “*continuous release and deployment*” con utilizzo di tecniche che permettono di realizzare e gestire una delivery pipeline che facilita il rilascio in maniera efficiente ed automatica. Questa fase prevede l'utilizzo di strumenti di “deploy automatizzato” e di gestione del rilascio che permettono di tener traccia delle diverse versioni dell'applicazione distribuite su ogni nodo, velocizzare la configurazione dei diversi ambienti e condividere informazioni e risorse.

Operate: In questa fase è prevista l'adozione di due pratiche :

- **Continuous Monitoring:** prevede di monitorare l'applicazione non solo nell'ambiente di produzione, ma ad ogni fase del ciclo di vita raccogliendo dati che permettano agli stakeholders di migliorare o cambiare le funzionalità da rilasciare e sulla base di essi orientare i prossimi piani di business.
- **Continuous Customer Feedback and Optimization:** i dati più importanti da raccogliere sono ovviamente quelli incentrati sull'esperienza del cliente. In particolare è importante raccogliere dati sull'utilizzo che il cliente fa dell'applicazione e dati di feedback con cui egli stesso può segnalare problemi e migliorie da apportare. Il processo di feedback continuo serve dunque ad alimentare quello di *continuous optimization* che consente all'azienda di migliorare oltre che il prodotto anche il processo produttivo, rispondendo con agilità alle esigenze del mercato.

2.3 DevOps e microservizi

Per quanto riguarda l'organizzazione del personale, in un'azienda DevOps è prevista la suddivisione in team composti da poche persone. In un team di piccole dimensioni infatti è più semplice mantenere unità di intenti e coordinazione, mentre durante i meetings è possibile ascoltare e considerare le opinioni di tutti e prendere decisioni più velocemente . Lo svantaggio di tale organizzazione è che per lavori complessi è necessaria una scomposizione in task più piccoli da assegnare a team diversi che lavoreranno ciascuno su una piccola porzione di codice e avranno la necessità di lavorare in coordinamento tra loro. L'obiettivo di DevOps di minimizzare i tempi necessari al coordinamento tra i team può essere ottenuto adottando un'architettura a microservizi dove i meccanismi di coordinamento e di gestione risorse sono specificati dall'architettura stessa richiedendo in tal modo una minima coordinazione tra team. In tale architettura la funzionalità totale del sistema è derivata dalla composizione di più servizi ognuno dei quali è realizzato da un team e fornisce una piccola quantità di funzionalità. L'utente interagisce con un unico componente che fornisce il servizio di interfaccia e che utilizza molteplici altri servizi. Ciascun componente offre un servizio ad altri componenti e si comporta a sua volta da client nel momento in cui necessita di servizi forniti dai componenti sottostanti. L'architettura globale definisce un modello di coordinamento che stabilisce le modalità in cui due servizi possono comunicare tra loro e con cui ad un client viene comunicata la disponibilità di un servizio. Il servizio che diventa disponibile può effettuare una registrazione ad un registro specificando oltre al nome le info su come può essere invocato da un client. Per esempio il registro può contenere un URL associata ad ogni servizio oppure può comportarsi da server DNS locale nel momento in cui viene salvato un indirizzo IP ad ogni registrazione di un servizio. Un client che necessita di un servizio deve semplicemente recuperare dal registro le info che servono ad invocarlo. L'architettura stabilisce un protocollo di comunicazione che deve essere usato solo per la comunicazione tra servizi e che può basarsi su http, rcp etc.

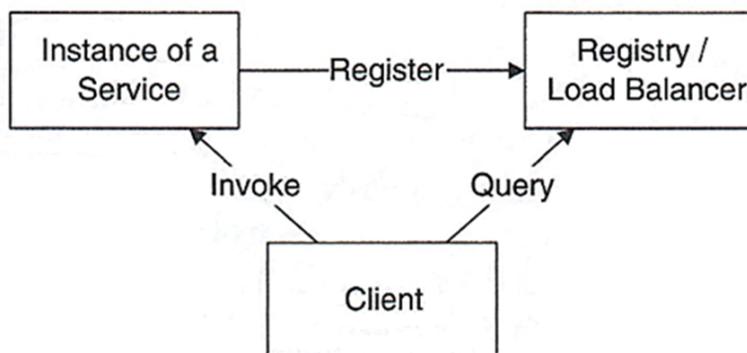


FIGURA 2.3: **Registrazione istanze e load balancing**

Per evitare fallimenti e supportare il carico ci saranno più istanze di uno stesso servizio registrate contemporaneamente e sarà il registro stesso a provvedere al bilanciamento del carico alternando ogni richiesta con politica rotatoria alle diverse istanze. Il registro inoltre si comporta anche da garante contro i possibili fallimenti di un'istanza di servizio. Esso infatti richiede periodicamente ai servizi di rinnovare la propria registrazione e se un'istanza di servizio non riesce a rinnovare la registrazione entro un tempo stabilito, essa verrà rimossa dal registro. Per quanto riguarda le decisioni sulla gestione delle risorse, anche esse possono essere incorporate dall'architettura stessa. In risposta all'aumento di richiesta di un servizio, o al fallimento di esso possono essere istanziate nuove macchine virtuali. E' conveniente usare istanze che non conservino info sullo stato, o stateless, che possano essere messe immediatamente in servizio dopo il provisioning e per le quali il deprovisioning risulta particolarmente semplice. L'istanza può infatti essere semplicemente rimossa allo scadere di un periodo di *cool-down* in cui non ha ricevuto nuove richieste e in cui ha esaurito tutte quelle già in lista. Un servizio stateless inoltre facilita il bilancio del carico poiché qualsiasi messaggio può essere mandato a qualsiasi istanza. Alla luce di ciò si può convenire sul fatto che è conveniente mantenere lo stato esternamente alle istanze di servizio, usando tool o risorse di storage apposite. Il controllo del *provisioning/deprovisioning* delle istanze di un servizio può essere eseguito in più modalità:

- Il servizio è esso stesso responsabile di espandere o ridurre la capacità in relazione alla quantità di domanda. Esso tiene traccia della coda di richieste e della performance di risposta e al superamento di valori di soglia provvede ad istanziare o rimuovere istanze superflue.
- Un componente esterno controlla le performance delle istanze e il carico distribuito su di esse monitorando per esempio l'utilizzo delle CPU e in base a ciò aumentare o diminuire il numero di istanze attive.
- Un componente della catena di client valuta in base alla quantità di domanda se è necessario ridurre o approvvigionare nuove istanze.

L'implementazione di un microservizio richiede la realizzazione e la connessione tra più moduli sviluppabili indipendentemente. Un modo per ridurre i tempi di coordinamento tra team è includere il lavoro di un singolo team in moduli e stabilire interfacce tramite le quali moduli prodotti da team diversi possano comunicare tra loro. Si può inoltre stabilire che la realizzazione di un componente, che costituisce un microservizio, sia responsabilità di un solo team di sviluppo senza precludere la possibilità che un team possa lavorare su più di un componente. Si punta quindi a trasferire il problema della coordinazione tra team imponendo vincoli architettonici di interfacciamento tra componenti. Per quanto concerne l'allocazione, ogni componente deve esistere e poter essere soggetto a deploy

indipendentemente dagli altri e più componenti possono coesistere sulla stessa macchina virtuale. L'upgrade e la redistribuzione di un microservizio non hanno quindi effetti su altri microservizi. L'attuazione delle pratiche di *Continuous Integration*, *Continuous Delivery* e *Continuous Deployment* previste da DevOps rendono impossibile un design tradizionale poiché per un'applicazione sviluppata in maniera monolitica, ogni piccola modifica comporta tempi di integrazione e rilascio estremamente lenti. In tale contesto una naturale risposta è rappresentata dall'architettura a microservizi, che scomponete il prodotto in servizi atomici, sviluppabili e rilasciabili indipendentemente, e comunicanti tra loro tramite API e protocolli comuni. Una tendenza ormai affermata, per lo sviluppo software, è quella di affiancare la filosofia di design della Microservice Architecture (MSA), a quella di development costituita da DevOps. Ciò può avvenire con particolare naturalezza in quanto MSA condivide con DevOps molti principi e valori Agili. Quindi se da un lato DevOps necessita di uno stile architettonico agile come quello dei microservizi, dall'altro il miglior modo per implementare un sistema di microservizi è quello di adottare le pratiche e linee guida di DevOps anche grazie al comune supporto tecnologico fornito dal Cloud.

2.4 DevOps e cloud computing

Per le aziende che intendono incrementare la propria competitività nel mercato attuale, le strade migliori sono quelle che prevedono DevOps e cloud computing. Non si tratta di strumenti mutuamente esclusivi, ma si può dire che siano complementari in quanto, mentre DevOps riguarda prettamente i processi aziendali e il miglioramento di essi, cloud computing si focalizza su tecnologia e servizi di supporto. Il cloud computing permette agli sviluppatori di avere un migliore controllo sulle componenti che si traduce in tempi di attesa più brevi. Si ricordi infatti che in un contesto DevOps è previsto che le attività di sviluppo e test siano condotte in ambienti conformi a quello di produzione. La mancata somiglianza degli ambienti di sviluppo a quello di produzione può inficiare la qualità del prodotto, mentre la mancata disponibilità dell'ambiente e i relativi tempi di provisioning e configurazione possono inficiare le prestazioni del processo creando lunghi tempi di attesa. Quindi se da un lato usare cloud computing senza aver adottato la metodologia DevOps significherebbe non avere le giuste linee guida per poterne sfruttare tutti i benefici, è evidente che dall'altro lato associare a DevOps i servizi offerti da una piattaforma cloud significherebbe alleggerirne il carico di lavoro, eliminare bottlenecks dalla delivery pipeline e massimizzarne le prestazioni. Avere i diversi ambienti ospitati da una piattaforma cloud significherebbe per i dipendenti averli sempre a disposizione e poterne richiedere liberamente l'accesso "on demand" mentre la possibilità di un provisioning dinamico, per l'azienda si traduce in una riduzione dei costi che si avrebbero in caso di ambienti di test gestiti in maniera statica e permanente. Con l'utilizzo di alcune tecnologie messe a

disposizione dal cloud, inoltre, è possibile per l'azienda definire e gestire versioni degli ambienti che si adattino meglio alle necessità dei dipendenti oltre che automatizzare le attività di deployment. E' possibile infatti affidarsi ad un unico strumento che gestisca il provisioning degli ambienti cloud e quando necessario distribuisca su di essi le giuste versioni delle applicazioni occupandosi anche delle operazioni di configurazione di entrambi. Strumenti cloud agevolano inoltre la pratica del continuous testing di DevOps permettendo mediante virtualizzazione di eseguire test di un componente anche quando non si dispone di tutti i servizi necessari al suo funzionamento. L'attività di deployment su una piattaforma cloud può essere eseguita con due diversi approcci. Il primo approccio prevede di trattare separatamente l'attività di provisioning/configurazione degli ambienti e successivamente provvedere al deploy dell'applicazione. Con questo approccio lo strumento di automazione del deployment vedrebbe gli ambienti come normali ambienti statici e ciò non massimizza i benefici dell'utilizzo del cloud. Il secondo approccio prevede di usare uno strumento di automatizzazione del deployment come strumento unico per il provisioning degli ambienti e la distribuzione delle applicazioni su di essi. Per fare ciò è necessario creare una mappatura della topologia degli ambienti che consenta di effettuare il matching dei componenti dell'applicazione ai relativi nodi. In base al rapporto tra responsabilità del provider e quelle lasciate al cliente, è possibile categorizzare tre modalità di servizio cloud:

- **Infrastructure as a service:** è la forma di servizio più basilare con la quale viene fornita un'infrastruttura IT di cui il cliente può controllare la configurazione delle reti e di altre risorse base e sulla quale può distribuire e utilizzare software arbitrario come sistemi operativi, risorse di storage e applicazioni.
- **Software as a service:** Il cliente ha la possibilità di utilizzare sull'infrastruttura cloud le applicazioni fornite dal provider stesso. Non ha la possibilità di controllare gli strati sottostanti che includono la configurazione di reti, server, sistemi operativi e applicazioni.
- **Platform as a service:** è una forma di servizio intermedio tra i due precedenti. Il cliente ha la possibilità di scegliere e usare sull'infrastruttura applicazioni acquistate o create da se stesso e di controllare le impostazioni degli ambienti che le ospitano, ma non ha controllo sui livelli sottostanti.

2.5 Adottare DevOps: tecniche e strumenti

2.5.1 Tecniche

Adottare DevOps vuol dire abbracciare una nuova cultura aziendale che evolve dai concetti di lean e agile. Fase preliminare alla realizzazione di un ambiente DevOps è l'analisi e l'eliminazione dei bottlenecks dalla pipeline che portano ad un rallentamento delle prestazioni del sistema produttivo e l'individuazione degli sprechi di lavoro inutile spesso dovuti a scarsa comunicazione e coordinamento tra i dipartimenti. Succesivamente bisogna identificare gli obiettivi di business e, anche riorganizzando il sistema di incentivi e ricompense, riallineare il lavoro di tutti al raggiungimento di goal condivisi tra dipendenti e azienda. I dirigenti devono quindi elaborare un intero processo di business basato su collaborazione, fiducia e condivisione di responsabilità e ciò richiede che venga fornita a tutti i dipendenti una continua visibilità degli obiettivi e dello stato globale del progetto anche mediante l'uso di specifici tool di condivisione e collaborazione che si analizzeranno di seguito. DevOps prevede l'adozione di una serie di tecniche: **Continuous improvement**: l'adozione della metodologia DevOps deve essere intesa come un'attività graduale supportata da un processo di continuo perfezionamento. L'azienda potrebbe incaricare dei team che si occupino di osservare costantemente le attività per identificare le aree di miglioramento e ottimizzare per raffinamenti successivi i processi adottati. **Release planning**: un piano di rilascio serve a far fronte alla necessità di garantire la distribuzione delle funzionalità richieste dai clienti entro le deadlines prestabilite. Tradizionalmente le aziende gestiscono e mantengono tracciabilità delle attività di rilascio organizzando meetings periodici in cui vengono presentati ed esaminati i dati raccolti in presenza di tutti gli stakeholders. DevOps definisce invece accurati processi che con largo uso di automazione raccolgono in maniera automatica dati generando informazioni consultabili in ogni momento. **Continuous integration**: Mediante questa pratica sviluppatori di team e sedi diverse possono aggiungere frequentemente modifiche al codice custodito in un repository centralizzato. Le operazioni di building e test, contestualmente ad ogni integrazione, sono eseguite ed automatizzate da appositi strumenti e ciò permette di identificare e correggere i bug con maggiore tempestività, migliorando la qualità del software e riducendo il tempo di rilascio degli aggiornamenti. **Continuous delivery**: La distribuzione continua è un metodo che estende l'integrazione continua in cui le modifiche al codice vengono applicate a una build e successivamente testate e predisposte al rilascio in modo automatico. L'automatizzazione è favorita dall'utilizzo di tool che velocizzano le operazioni di configurazione, eseguono routine di test automaticamente e collezionano i dati raccolti mantenendo sempre disponibile una versione temporanea e distribuibile che ha già superato i test standardizzati. **Continuous Testing**: come già detto il testing non è più un'attività circoscritta ad una sola fase del ciclo di vita che copre lunghi intervalli

temporali ma continuamente sono eseguite routine di test standardizzate velocizzate da appositi strumenti. In particolare esistono strumenti che automatizzano test di integrazione, funzionalità, performance, sicurezza, agevolano la configurazione degli stessi ambienti di test e permettono una service virtualization nel caso in cui sarebbero necessari i servizi di componenti non ancora disponibili. **Continuous monitoring and feedback:** come si è detto precedentemente le aziende hanno bisogno di raccogliere dati di feedback in grandi quantità per poter pianificare di conseguenza le attività relative ai prossimi rilasci. In particolare, oltre a messaggi esplicativi dell'utente finale, che segnalano suggerimenti o problematiche, l'azienda deve acquisire, categorizzare e analizzare dati di log generati da applicazioni e infrastruttura che permettono di capire come l'esperienza d'uso è influenzata dalle prestazioni del sistema e da possibili sue modifiche. **Infrastructure as a code:** agevola e velocizza il provisioning e la configurazione degli ambienti, attività che spesso possono rallentare le prestazioni della delivery pipeline. *Infrastructure as a code*, a cui è associato il concetto di *software defined environments*, permette di trattare la definizione e configurazione dei diversi nodi tramite codice piuttosto che come operazioni manuali. Inoltre è previsto l'uso di tools che, per sistemi complessi, permettono di definire topologie, ruoli, relazioni e politiche di workload per ognuno dei nodi componenti. Tali strumenti sono categorizzabili in: **Application- or middleware-centric tools:** tools che gestiscono come codice sia i server che le applicazioni attive su di essi. Non possono eseguire attività a basso livello, ma sono capaci di automatizzare totalmente quelle a livello server e a livello applicativo. **Environment and deployment tools:** strumenti che si occupano del deploying del codice applicativo e anche delle configurazioni dell'infrastruttura. **Generic tools:** non sono tools specifici come i precedenti, ma possono eseguire un più vasto range di operazioni dalla configurazione di un OS a quella delle porte del firewall.

2.5.2 Configuration Management

La natura agile di DevOps, con l'attuazione delle pratiche di continuous integration e continuous delivery rende necessaria una particolare attenzione alla gestione delle diverse versioni sviluppate durante il processo evolutivo del software. DevOps, come tutti gli approcci Agile, richiede strumenti di Version Management al fine di controllare le modifiche apportate in maniera indipendente da diversi sviluppatori e supportare l'attività di integrazione eseguita più volte al giorno: le versioni definitive dei componenti sono mantenute in un repository condiviso mentre gli sviluppatori apportano modifiche sui propri workspace e solo dove averle testate eseguono l'update sul repository di progetto. Dal punto di vista della CM ci sono tre fasi di sviluppo:

- **Fase di sviluppo:** gestire la configurazione del software insieme all'aggiunta di nuove funzionalità;

- **Fase di system testing** in cui le modifiche fatte riguardano solo bugs, miglioramenti di prestazioni e sicurezza;
- **Fase di rilascio:** il software è rilasciato ai clienti i quali possono richiedere nuovi cambiamenti (nuove funzionalità, correzione di errori/vulnerabilità).

Per sistemi di grandi dimensioni non esiste una sola versione “funzionante” del sistema. D’altra parte possono esserci diversi team che lavorano sullo sviluppo di diverse versioni.

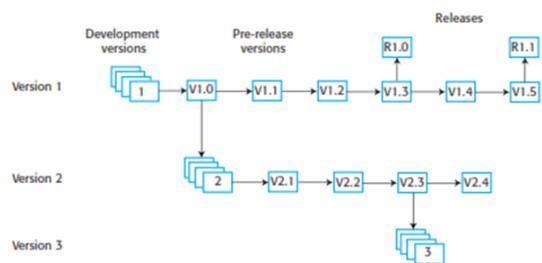


FIGURA 2.4: Configuration Management

La gestione delle versioni (VM) è il processo che tiene traccia delle diverse versioni dei componenti software o degli elementi di configurazione e dei sistemi in cui vengono utilizzati questi componenti. Uno degli obiettivi è quello di fare in modo che modifiche apportate da diversi sviluppatori non vadano in conflitto l’una con l’altra. **Codeline**: sequenza di versione di componenti, dove la versione successiva è derivata dalle precedenti. **Baseline**: definizione di uno specifico sistema -> specifica le versioni dei componenti incluse nel sistema (insieme alle librerie utilizzate, i file di configurazione, etc.). N.B. Le baseline sono usate per ricreare una versione specifica del prodotto software. I **Version control systems (VC)** identificano, conservano e controllano l’accesso alle diverse versioni dei componenti. Esistono due tipi di sistemi: **Centralizzati**: unico repository master da cui gli sviluppatori scaricano i componenti che vogliono modificare; **Distribuiti**: esistono più versioni del repository allo stesso tempo.

Per supportare lo sviluppo indipendente senza interferenze i VC utilizzano i concetti di repository di progetto e spazio di lavoro privato: il repository mantiene la versione “principale” di tutti i componenti ed è usato per creare le baseline, gli sviluppatori eseguono le modifiche nel loro spazio di lavoro lavorando su copie.

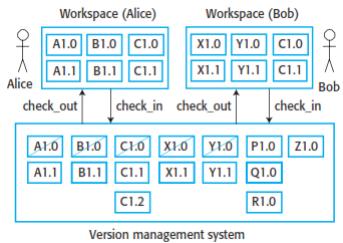


FIGURA 2.5: VC centralizzato

VC centralizzati: operazioni di check-in/check-out da parte degli sviluppatori per caricare/scaricare i file che vogliono modificare/hanno modificato. Se più persone stanno lavorando su un componente allo stesso tempo, ognuno fa il check-out dal repository. Quando un componente è stato estratto il sistema VC avvisa altri utenti che quel componente lo sta modificando anche qualcun altro. Dalla figura si vede che quando Bob farà il check-in di C1.1, il VC crea una nuova versione C1.2 per non interferire con quella di Alice.

VC distribuiti: ogni sviluppatore clona il repository di progetto e lo installa sul suo computer. Al termine delle modifiche eseguono il commit e aggiornano il proprio repository del server privato, possono quindi decidere di fare push di queste modifiche nel repository di progetto.

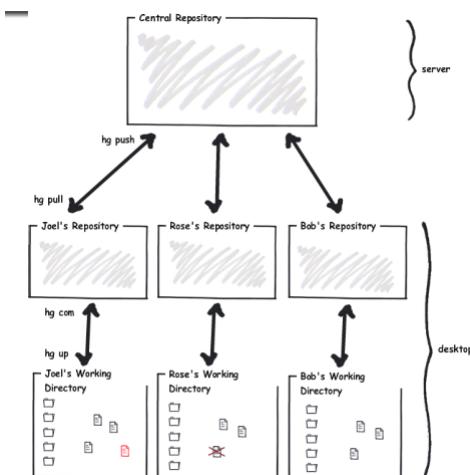


FIGURA 2.6: VC Distribuiti

2.5.3 Approfondimento GIT

Tra gli strumenti per il Version Management ricordiamo Git. Esso pur avendo un interfaccia utente simile a quella degli altri Version Control System, presenta notevoli differenze rispetto agli altri VCS.

Snapshot vs Delta

La principale differenza tra Git e gli altri VCS (inclusi Subversion e compagni), è come Git considera i suoi dati. Come si vede in figura, gli altri sistemi tendono ad immagazzinare i dati in termini di cambiamenti rispetto alla versione base di ogni file.

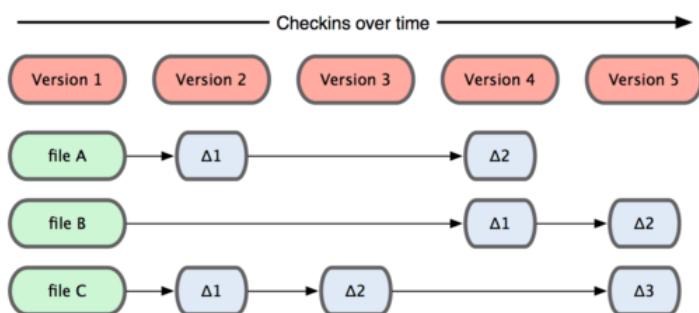


FIGURA 2.7: Versioni di componenti come liste di Delta

Git usa un approccio più veloce: i file memorizzati vengono prima compressi insieme alle meta-informationi ad essi associate. Il recupero del file quindi è una semplice decompressione. In particolare Git considera i propri dati più come una serie di istantanee (**snapshot**) di un mini filesystem (figura). Ogni volta che si effettua un committ, Git crea un'immagine di tutti i file in quel momento, salvando un riferimento allo **snapshot**. Inoltre Git non risalva i file immutati dall'ultimo committ, bensì crea un collegamento al file precedente già salvato.

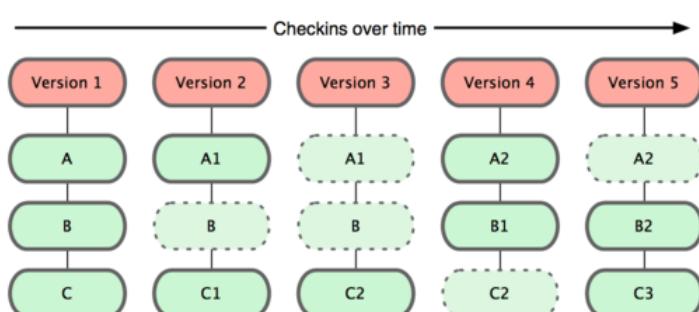


FIGURA 2.8: Versioni di componenti gestite da Git

Tutte le operazioni sono locali

A differenza degli altri VCS in cui la maggior parte delle operazioni sono soggette alle latenze della rete, in Git la maggior parte delle operazioni necessitano solo di file e risorse locali. Per esempio, per navigare la storia di un progetto, Git non ha bisogno di connettersi al server per scaricarla e per poi mostrarla ma può leggerla direttamente dal proprio database locale. Questo significa che si può risalire a vecchie versioni del progetto quasi istantaneamente. Inoltre anche quando si è offline si può modificare e fare committ della propria copia locale per poi poter effettuare l'upload in un secondo momento.

Checksum

Qualsiasi cosa in Git è controllata, tramite checksum, prima di essere salvata ed è referenziata da un checksum. Ciò comporta che è impossibile che il contenuto di un file o directory sia modificato, perso o corrotto senza che Git se ne accorga. Questa funzionalità di checksum è basata su un hash chiamato SHA-1 calcolata in base al contenuto di file o della struttura della directory in Git e il riferimento ad un file non è basato sul nome del file, ma sull'hash del suo contenuto.

Tre stati

Git gestisce i file assegnando ad essi uno tra tre stati possibili:

- I file **committati** sono quelli salvati in maniera persistente nel database
- I file **modificati** sono quelli che hanno subito modifiche ma non sono stati ancora committati
- I file **in stage** sono quei file modificati che sono stati contrassegnati per entrare a far parte nello snapshot del prossimo commit.

Questo ci porta alle tre sezioni principali di un progetto Git: la directory di Git, la directory di lavoro e l'area di stage. Questi tre stati definiscono tre aree di un progetto Git:

- **Git Directory:** è l'area relativa al database dei dati e dei relativi metadati. E' in quest'area che vengono salvati i repository clonati da altri pc.
- **Directory di Lavoro:** è un checkout di una versione specifica del progetto. Questi file vengono estratti dal database compresso nella directory di Git, e salvati sul disco per essere usati o modificati.
- **Stage Directory:** è un file, contenuto generalmente nella directory di Git, con tutte le informazioni necessarie alla prossima commit.

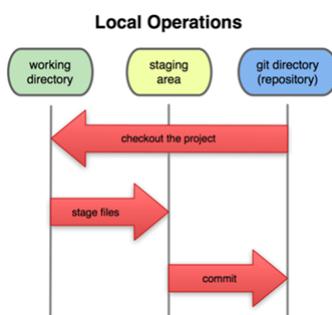


FIGURA 2.9: I tre stati dei file git

Conclusioni

Alla luce di quanto detto nel testo di questo elaborato, DevOps si configura come un approccio essenziale ed efficace per le aziende che intendono assimilare i principi lean e agili e acquisire le capacità necessarie per poter rispondere con rapidità ai cambiamenti delle richieste di mercato. DevOps fornisce alle aziende gli strumenti per favorire continuous integration e delivery accelerando così il rilascio di nuove funzionalità software, migliora il monitoraggio del sistema e l'esecuzione delle attività del ramo production. Se da un lato può sembrare evidente che DevOps sia la strada che facilita il raggiungimento del successo nel nuovo contesto di mercato, di contro si trovano diverse difficoltà che ne ostacolano l'adozione da parte di un'azienda. Non si tratta infatti di un simple switch che prevede semplicemente di assimilare e usare all'occorrenza processi e pratiche dell'ennesimo modello di sviluppo emergente, poichè per l'azienda tradizionale, l'adozione di DevOps richiede un totale stravolgimento del modo consolidato di fare business. Si tratta di un cambiamento che non può riguardare solo i team di sviluppo ma che coinvolge tutta l'azienda e che richiede la propensione da parte di dirigenti e dipartimenti ad accogliere e adattare il proprio operato alla nuova cultura aziendale. Nonostante persistano difficoltà ed una certa componente di scetticismo, la diffusione di articoli, materiale che descriva come adottare DevOps, strumenti di supporto ed esempi di esperienze d'uso da parte di importanti aziende innovatrici quali Google, Netflix, Amazon, agevolerà la transizione verso l'adozione mainstream di questa nuova metodologia.