POLITECNICO DI MILANO

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica

Dipartimento di Elettronica, Informazione e Bioingegneria

INGEGNERIA DEL SOFTWARE BASATA SU DEVOPS: UNO STUDIO PRELIMINARE TRAMITE APPROCCI "MODEL-DRIVEN"

Relatore: Prof. Raffaela Mirandola

Correlatore: Dr. Damian Andrew Tamburri

Tesi di laurea di:

Giuseppe Vergori

Matr. 824298

Anno accademico 2015-2016

Dedico questo lavoro ai miei affetti più cari.

RINGRAZIAMENTI

Desidero ringraziare tutti coloro che mi hanno aiutato nella stesura della tesi con suggerimenti, critiche ed osservazioni. A loro va la mia gratitudine, anche se a me spetta la responsabilità per ogni errore contenuto in questa tesi.

Ringrazio anzitutto la professoressa Raffaela Mirandola, Relatore, il dottor Damian Andrew Tamburri, Co-relatore, e il dottor Diego Perez-Palacin. Senza il loro supporto e la loro guida sapiente questa tesi non esisterebbe.

Vorrei infine ringraziare la mia famiglia e le persone a me più care che, indipendentemente dalle distanze, mi hanno sempre sostenuto, incoraggiato e spronato.

INDICE

ELENCO DELLE FIGURE

SOMMARIO

Questo lavoro si pone nell'ambito dello studio dei modelli di sviluppo e del ciclo di vita del software DevOps, una metodologia di sviluppo software che di recente ha avuto un forte impatto nelle realtà aziendali. Tuttavia l'adozione di DevOps risulta essere carente sfruttando risorse e approcci che un'azienda già possiede. Abbiamo studiato una possibile soluzione per tentare di mitigare questo problema, attraverso un approccio che sia in grado di pianificare e guidare i processi che costituiscono il ciclo di vita del software tramite un metodo DevOps, sfruttando i modelli consolidati nel campo dell'ingegneria del software, come i diagrammi UML e le reti di Petri. In questo modo potrebbe essere meno faticoso per le aziende adottare DevOps usando strumenti di modellazione che già possiedono.

1 INTRODUZIONE

Un modello di sviluppo del software è la descrizione delle attività condotte in un progetto di ingegneria del software, tenendo conto dell'ordine in cui sono svolte [1]. Il processo di sviluppo e supporto del software spesso richiede di svolgere compiti ben precisi, eseguiti da persone differenti in un ordine stabilito, al fine di evitare ritardi e problemi che potrebbero comportare il fallimento dell'intero progetto. Il primo modello sviluppato in ordine storico è il "modello a cascata", che ha subito diverse evoluzioni nel corso degli anni. Tuttavia, questi modelli di sviluppo sono privi di processi di comunicazione e collaborazione tra le diverse divisioni aziendali, che generalmente operano in maniera isolata. In questo contesto si inserisce DevOps [7] [8], abbreviazione dei termini in inglese di "Development" e "Operations", che indicano rispettivamente lo "sviluppo" e la "messa in produzione" di un prodotto software, è una metodologia di sviluppo del software "snella" e "agile", che di recente si sta diffondendo nelle realtà aziendali. Essa mira alla comunicazione e collaborazione tra un'azienda e i dipartimenti da cui è composta, come sviluppo, produzione e controllo qualità, con lo scopo di aiutare un'organizzazione a sviluppare in modo continuo, rapido ed efficiente, prodotti e servizi software, garantendo alta qualità e reagendo immediatamente alle richieste del mercato e dei consumatori.

Nonostante all'inizio DevOps ha suscitato molta attenzione tra le aziende dato i benefici che comporta, tuttavia risultano esserci ancora delle carenze su come adottare questa nuova metodologia sfruttando le risorse e strumenti che un'organizzazione già possiede. Assumendo che questa consapevolezza esista, le aziende considerano DevOps solo come "un'altra strategia". Quest'indifferenza nasce dal fatto che gli

approcci e i modelli precedenti possiedono procedure valide, che permettono di sviluppare un software in maniera efficiente.

Non esistono quindi ad oggi modelli e metodi che consentono di verificare effettivamente che i processi che costituiscono il ciclo di vita del software migliorino se si adotta un approccio DevOps. In questo contesto si inserisce EDEN [2], acronimo di "End-to-end Devops ENgineering", ovvero un approccio "model-based" in grado di pianificare e guidare i processi che costituiscono il ciclo di vita del software tramite un metodo DevOps, sfruttando i modelli consolidati nel campo dell'ingegneria del software, come i diagrammi UML [15] e le reti di Petri [16].

Grazie a EDEN, potrebbe essere meno faticoso per le aziende adottare DevOps usando strumenti di modellazione che già possiedono. Consentirebbe alle aziende di combinare metodologie di sviluppo agili con strumenti e modelli provati dell'ingegneria del software, dato che l'azienda già li possiede. Ad esempio, i diagrammi di sequenza UML possono essere usati come input per strumenti che li trasformano, in maniera automatica, in reti di Petri, dove possono essere condotte delle analisi più accurate che verificano l'affidabilità e la stabilità dei processi che compongono il sistema aziendale.

In questo lavoro esamineremo un caso di studio DevOps tratto dal libro "The Phoenix Project" [3], che tratta degli sforzi compiuti da un'azienda nell'adottare questa nuova metodologia. Assumeremo che i dipendenti dell'organizzazione usino EDEN, che fornirà informazioni accurate sullo stato dei processi e preziose raccomandazioni. Gli scenari analizzati saranno modellati tramite i diagrammi di sequenza UML, per identificare il comportamento di un sistema e mostrare le comunicazioni tra i diversi partecipanti, e le reti di Petri, in quanto sono modelli formali basati su precise teorie matematiche e appropriati per modellizzare e analizzare sistemi. Inoltre, molti metodi di analisi e verifica sono stati sviluppati per le reti di Petri e, tra questi, considereremo quelli che trasformano i diagrammi di sequenza in reti di Petri.

1.1 STRUTTURA DEL DOCUMENTO

In questo documento presentiamo il lavoro svolto, organizzato con la seguente struttura:

- *Cap.2:*L'EVOLUZIONE DEI MODELLI DI SVILUPPO DEL SOFTWARE. In questo capitolo esamineremo in generale come i modelli di sviluppo del software si sono evoluti negli anni, individuando le principali differenze che li contraddistinguono, partendo dal "modello a cascata" fino ad arrivare alla metodologia DevOps, descritta con maggiore dettaglio. Vedremo che, sebbene il nome sembra soltanto riferirsi ai team di sviluppo e operativi, in realtà coinvolge tutti gli elementi che costituiscono un'azienda, tra cui i dirigenti, l'architettura, il design, lo sviluppo, controllo qualità, produzione, sicurezza, partner e fornitori. Esamineremo l'architettura di riferimento. Essa aiuta i professionisti ad utilizzare le linee guida, le direttive e altro materiale necessario per il design di una piattaforma Devops che ospita persone, processi e tecnologie.

- *Cap. 3:*EDEN. In questo capitolo, discuteremo dei problemi riscontrati dalle aziende che adottano una metodologia DevOps e come abbiamo impostato il processo di studio. Presenteremo EDEN, focalizzandoci nel contesto in cui si inserisce e tracciandone le caratteristiche. Descriveremo le informazioni date in input a EDEN e i modelli che rilascia in output, che utilizzeremo per condurre l'analisi di un caso di studio DevOps.

- Cap. 4:UN CASO DI STUDIO DEVOPS: "THE PHOENIX PROJECT". In questo capitolo sfrutteremo EDEN per analizzare un caso di studio DevOps, generando i modelli che ci permetteranno di esaminare gli scenari in questione. I risultati ottenuti saranno discussi e valutati.

- *Cap. 5:*CONCLUSIONI. In questo capitolo presenteremo le nostre conclusioni in merito al lavoro svolto e discuteremo degli aspetti da studiare e delle possibili direzioni future da poter intraprendere per proseguire e migliorare questo lavoro.

2 L'EVOLUZIONE DEI MODELLI DI SVILUPPO DEL SOFTWARE

Nel capitolo precedente abbiamo parlato di DevOps [7] [8], una metodologia di sviluppo del software "snella" e "agile" che mira alla comunicazione e collaborazione tra i team di sviluppo, operativi e controllo qualità, con lo scopo di aiutare un'organizzazione a sviluppare in modo continuo, rapido ed efficiente, prodotti e servizi software, garantendo alta qualità e reagendo immediatamente alle richieste del mercato e dei consumatori. Per capire da dove deriva questa metodologia, analizzeremo i modelli di sviluppo software adottati dalle aziende nel corso degli anni.
Il capitolo presenta un "background" sui modelli di processo di sviluppo del software tradizionali presentati in [1]. Sono riportate le caratteristiche principali che li contraddistinguono, differenze ed evoluzioni di ognuno.
Nel paragrafo ***2.4*** è descritta la metodologia DevOps nel dettaglio. Vedremo che, sebbene il nome sembra soltanto riferirsi ai team di sviluppo e operativi, in realtà coinvolge tutti gli elementi che costituiscono un'azienda, tra cui i dirigenti, l'architettura, il design, lo sviluppo, controllo qualità, produzione, sicurezza, partner e fornitori. Escludere uno qualsiasi di questi attori, interno o esterno all'organizzazione, porta ad una implementazione errata di DevOps.
Vedremo che DevOps non è concepito solo come una capacità IT, ma come un vero e proprio processo di business. Con un investimento in aree ben precise e automatizzando procedure standard, si eviteranno incidenti e quindi interruzioni, garantendo tempi di consegna più rapidi. Questo movimento ha prodotto diversi principi che si sono sviluppati nel tempo e si stanno ancora evolvendo, tra cui:

- Rispondere velocemente alle esigenze dei clienti, catturando continuamente i loro feedback.
- Aumentare la capacità per innovare, eliminando lavoro inutile dal sistema aziendale.
- Fare valutazioni in tempi più rapidi.

Fissati gli obbiettivi del business, un'azienda deve studiare quali cambiamenti deve apportare prima di adottare un approccio DevOps. Nel paragrafo ***2.4.1*** esamineremo l'architettura di riferimento. Essa aiuta i professionisti ad utilizzare le linee guida, le direttive e altro materiale necessario per il design di una piattaforma Devops che ospita persone, processi e tecnologie.

In principio, DevOps nasce come un movimento culturale, che coinvolge prima di tutto le persone. Nel paragrafo ***2.4.2***, vedremo che costruire una cultura è l'aspetto principale di DevOps. Esamineremo le persone, i processi e gli aspetti tecnologici del metodo.

Il ciclo di vita del software è un processo continuo e prende il nome di "Delivery Pipeline", analizzata nel dettaglio nel paragrafo ***2.4.4***. Rappresenta le fasi che lo sviluppo di un'applicazione percorre partendo dall'idea inziale e continuando dopo l'effettiva messa in produzione. I passi principali sono:

- Individuare bottlenecks, in modo da proteggerle ed evitare che si accumuli lavoro nel sistema.
- Le risorse aziendali migliori devono essere schierate al fronte, in quanto devono essere loro a dettare la linea di produzione;
- Permettere agli sviluppatori di implementare il codice in un ambiente simile a quello di produzione.
- Stabilire i "business goal", che si ridefiniscono continuamente a seconda dei feedback dei clienti.
- Collaborazione, grazie alla quale si integra continuamente il lavoro con altri reparti.

- Monitoraggio, una fase continua che, grazie alla tracciabilità del processo, qualora si verifichi un problema, si interviene immediatamente.

Nella sezione dedicata al CLOUD, paragrafo ***2.4.5***, vedremo come questa tecnologia può essere una risorsa eccezionale per DevOps. Infatti può garantire un ambiente di lavoro sempre disponibile, gestione delle risorse "On Demand" e configurazioni automatiche, che consentono un flusso di lavoro più fluido e con meno problemi.
Infine, nel paragrafo ***2.4.6***, individueremo le principali differenze tra DevOps e i modelli di sviluppo software tradizionali.

2.1 MODELLI DI PROCESSO DI SVILUPPO DEL SOFTWARE TRADIZIONALI

L'obiettivo di un modello di processo di sviluppo del software è quello di fornire una guida per coordinare e controllare sistematicamente un task o un'attività che deve essere svolta per raggiungere un prodotto finito e, quindi, gli obiettivi di un progetto software. Per un modello di processo di sviluppo del software vengono definiti:

- Un insieme di task che necessitano di essere svolti.
- L'input e l'output di ogni task.
- Le precondizioni e le post condizioni per ogni task.
- L'ordine di esecuzione dei task.
- Il flusso di esecuzione di ogni task.

Con l'aumentare della complessità di un progetto, a sua volta aumentano i task da eseguire e, di conseguenza, aumenta anche il numero delle persone coinvolte. Grazie ai modelli di processo di sviluppo del software, si può gestire e coordinare tempestivamente ogni situazione durante l'evoluzione del processo di sviluppo del software.

Non esiste un modello da applicare per lo sviluppo di un qualsiasi progetto software. Per questo sono stati creati diversi modelli, che si possono adattare e combinare a

seconda del progetto che si deve sviluppare. In questo paragrafo sono presentati, in maniera generale, i modelli di processo di sviluppo del software più tradizionali. Ognuno di essi può essere modificato in base alle specifiche circostanze.

2.1.1 MODELLO A CASCATA

Il modello a cascata, in inglese "Waterfall Model", rappresentato in ***Figura 2.1***, è il modello di processo di sviluppo del software più tradizionale. Prevede una sequenza di fasi, svolte in maniera sequenziale, ciascuna delle quali produce un ben preciso output che viene utilizzato come input per la fase successiva. Ogni fase del processo viene documentata perché necessaria alla fase successiva.

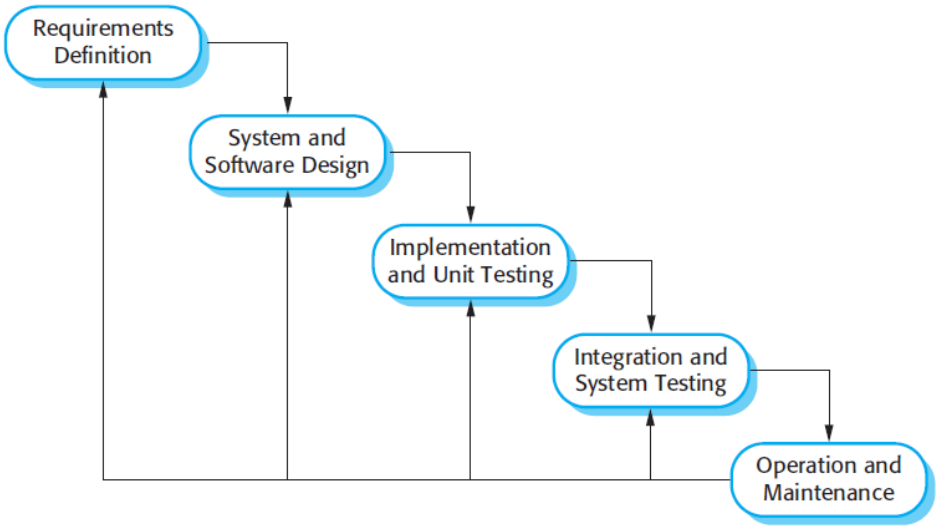

Figura 2.1 : Rappresentazione del Modello a Cascata [1].

Il modello a cascata è il modello di riferimento rispetto al quale gli altri modelli risultano delle varianti piuttosto che delle alternative. Esso prevede le fasi rappresentate in ***Figura 2.1***, ovvero:

Studio di fattibilità: consente di decidere se intraprendere o meno lo sviluppo del progetto, sia da un punto di vista tecnico che economico.

Analisi dei requisiti: si determina in dettaglio cosa farà il sistema, documentando in forma chiara e precisa i requisiti delle funzionalità richieste.

Design: si definiscono le specifiche dei requisiti, l'architettura del sistema nel dettaglio, la sua suddivisione in moduli e le relazioni tra di essi.

Sviluppo: costruzione dei moduli del sistema con un linguaggio di programmazione.

Testing: esecuzione di prove per verificare la correttezza dell'implementazione dei singoli moduli. Nello specifico avremo:

- Test di unità: prove eseguite dai programmatori per verificare la correttezza dell'implementazione dei singoli moduli.
- Test di integrazione: prove eseguite dagli analisti funzionali per verificare la correttezza del funzionamento complessivo del sistema.
- Test di accettazione: prove finali del funzionamento del sistema eseguite dagli utenti finali allo scopo di verificare l'aderenza ai requisiti e, quindi, di accettarne l'implementazione.

Rilascio: attività di consegna del sistema e sua installazione nell'ambiente di produzione. Comprende tutte le azioni necessarie che consentono di utilizzare il nuovo sistema, dalla predisposizione dell'ambiente infrastrutturale sino all'utilizzo da parte degli utenti finali;

Manutenzione: comprende tutte le attività volte a migliorare, estendere e correggere il software nel tempo.

Il limite del modello a cascata è rappresentato dall' eccessiva rigidità del modello che non facilita i miglioramenti in corso d'opera. Con l'evoluzione del software e dei linguaggi di programmazione, il modello è stato sottoposto a profonde critiche e revisioni. Ancora oggi il ciclo di vita a cascata continua a rimanere un punto di riferimento importante. Infatti rappresenta sempre il modello "canonico" rispetto al quale vengono solitamente descritte le "variazioni" moderne, come il "modello a cascata con ritorni", rappresentato in ***Figura 2.1***, dove l'errore viene propagato verso l'alto fino al modulo che lo ha generato.

Un'altra variante prende il nome di “modello a cascata con prototipazione”, caratterizzato dalla creazione di un prototipo “usa e getta”, che funge da meccanismo per identificare i requisiti. Prima di iniziare a lavorare sul sistema vero e proprio, viene costruito velocemente un prototipo in grado di simulare in scala ridotta il funzionamento del sistema, in modo da fornire agli utenti una base concreta per migliorare le specifiche. Il prototipo viene valutato dal cliente e il suo feedback viene usato per raffinare i requisiti e produrre un nuovo sistema, utilizzando le parti funzionanti del primo prototipo.

2.1.2 MODELLO EVOLUTIVO

Il modello evolutivo, chiamato anche “prototipizzazione”, è uno dei modelli di processo di sviluppo del software che cerca di superare i limiti principali del modello a cascata. Si basa sulla costruzione di prototipi, realizzati con strumenti software che permettono la rapida realizzazione di versioni semplificate. I prototipi permettono di sperimentare le funzionalità, di verificare i requisiti e conseguentemente di revisionare facilmente il progetto.

La prima versione di un software spesso presenta degli errori o delle limitazioni che costringono a rifare gran parte dell’applicazione. Il modello evolutivo considera la prima versione come un “throw-away”, cioè come un prototipo "cestinabile", che serve a fornire al progettista un feedback di analisi e verifica. Successivamente si procede alla realizzazione dell'applicazione vera e propria.

Il modello evolutivo è costituito principalmente dalle seguenti fasi:

a) Costruzione del prototipo sulla base dell’analisi dei requisiti.
b) Valutazione del prototipo.
c) Verifica del prototipo con il cliente.
d) Elaborazione del progetto sulla base delle valutazioni.

La seconda versione del software può essere poi sviluppata seguendo il modello a cascata, attraverso le fasi precedentemente descritte di “realizzazione” e “collaudo”.

Questo approccio evolutivo però fornisce solo una soluzione parziale ai problemi del modello a cascata, in quanto elimina gli errori generati nella fase di "analisi dei requisiti", è utile per valutare i costi, i tempi di realizzazione, la risposta del cliente e altro ancora, ma non riduce i tempi di realizzazione durante il ciclo di sviluppo.

2.1.3 MODELLO INCREMENTALE

La necessità di ridurre i tempi di sviluppo, ha comportato la trasformazione del modello evolutivo in incrementale. Tale modello prevede la realizzazione di un "prototipo parziale" capace di implementare un insieme significativo di funzionalità valutabili dal cliente, rimandando il resto a fasi successive. All'utente vengono forniti una serie di prototipi che integrano i feedback ricevuti in maniera incrementale. Questa tipologia viene detta "modello di sviluppo a rilascio incrementale". Tuttavia, le fasi del processo possono anche entrare in concorrenza. Ad esempio, mentre si sta integrando una versione può accadere che sia stata già avviata la progettazione di quella successiva, senza aver ancora valutato completamente i feedback della soluzione precedente. Con questo modello si riducono notevolmente i tempi, ma aumenta il rischio di perdere il controllo delle attività. Per evitare problemi è indispensabile definire degli standard di processo seguendo il modello a cascata. Nel modello incrementale la fase di "manutenzione", poiché è vista come attività di trasformazione continua, viene gestita come un'evoluzione del prototipo durante il progetto. In alcuni casi il "prototipo cestinabile" può essere sostituito con un "prototipo evolutivo" che, poco per volta, si trasforma nell'applicazione finale. Anche nell'uso del modello incrementale, il modello evolutivo rimane comunque molto utile per verificare alcune componenti del software come le interfacce, tali da essere create e adattate velocemente all'utente.

2.1.4 MODELLO A SPIRALE

Il modello a spirale di Boehm, ***Figura 2.2***, è un modello che abbina la natura iterativa della prototipazione e gli aspetti controllati del modello a cascata consentendo un rapido sviluppo di versioni successive sempre più complete del software.

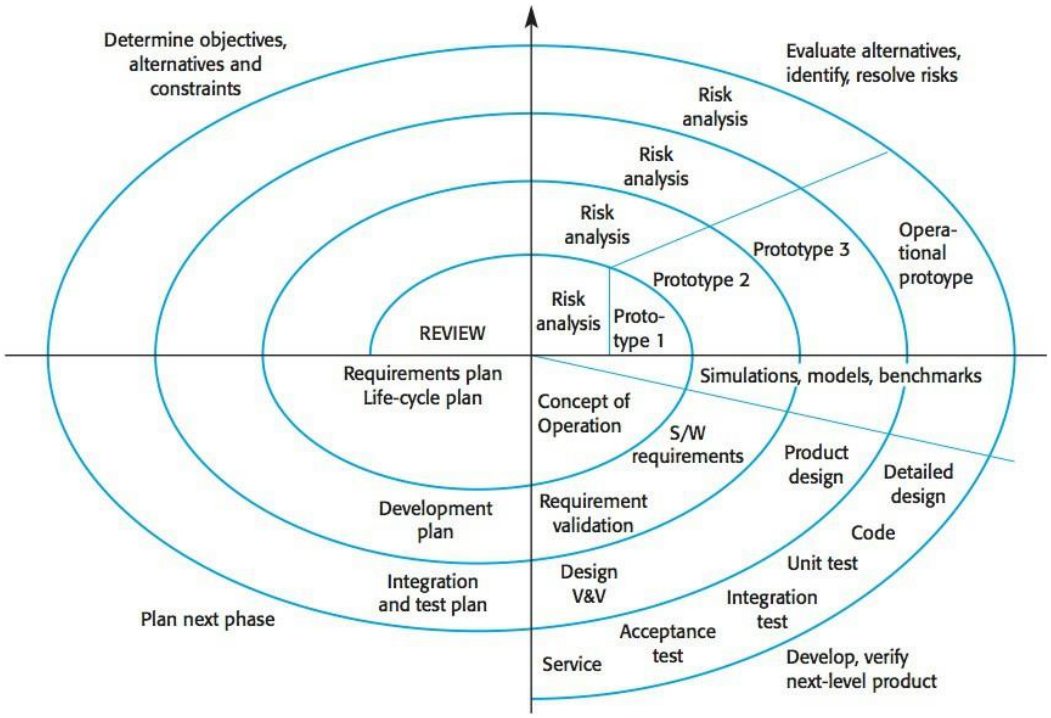

Figura 2.2: Modello a Spirale di Boehm [1].

Proposto per supportare l'analisi dei rischi, la caratteristica principale del modello è quella di essere ciclico e non lineare. Ogni ciclo di spirale si compone di quattro fasi e il raggio rappresenta il costo sostenuto sino a quel momento, mentre l'angolo rappresenta sia la fase in esecuzione, che il livello di avanzamento del processo all'interno di essa. Il modello si compone delle seguenti fasi:

1. Prima fase: identificazione degli obiettivi e delle soluzioni. Si determinano anche alternative e vincoli associati al progetto. Il committente e il fornitore del sistema interagiscono allo scopo di definire in maniera sufficientemente univoca cosa deve essere realizzato e come. In questa fase è buona norma comporre, o completare, dei

documenti, in principio non eccessivamente dettagliati, che fissino i punti fondamentali della pianificazione del lavoro futuro.

2. Seconda fase: valutazione delle soluzioni e rilevazione delle potenziali aree di rischio. Si identificano e analizzano i problemi e i rischi associati al progetto, al fine di determinare delle strategie per controllarli. Tra i rischi che devono essere presi in considerazione si annoverano i fattori di costo, di tempo e di variazione delle specifiche. I rischi più evidenti da valutare sono quelli di carattere economico, facendo riferimento ai costi di realizzazione, di gestione e di esercizio.

3. Terza fase: sviluppo e verifica del prodotto. È la fase che impiega più tempo, in quanto si procede alla vera e propria realizzazione. Si adotterà un approccio evolutivo, se i rischi riguardanti l'interfaccia utente sono dominanti; a cascata se è l’integrabilità del sistema il rischio maggiore; trasformazionale, se la sicurezza è più importante.

4. Quarta fase: revisione dei risultati delle fasi precedenti. Si verifica che il prodotto soddisfi effettivamente i requisiti richiesti oppure è necessario un altro ciclo di spirale.

In questo modello quindi il software viene sviluppato attraverso versioni successive e crescenti mediante la realizzazione di fasi successive che caratterizzano ogni giro della spirale. La spirale ha il pregio di considerare tutto il ciclo di vita sino alla consegna del software e oltre, in quanto permette di strutturare e programmare anche l’attività successiva all'installazione, cioè la manutenzione, che molti altri modelli trascurano. Il modello a spirale è consigliato per progetti di grandi dimensioni perché permette di raffinare, a ogni giro, il materiale precedentemente elaborato e approvato dal cliente: si parte dalla definizione degli intenti e dello scopo finale, poi si passa alla costruzione, verifica e validazione dei prototipi, fino a giungere al prodotto conclusivo.

2.2 UN PROCESSO PIÙ MODERNO: IL "RATIONAL UNIFIED PROCESS"

Il "Rational Unified Process" (RUP) è un framework di processo piuttosto che un singolo processo, da adattare alle diverse tipologie di progetto. Le sue caratteristiche principali sono:

- È guidato dai casi d'uso, usati principalmente per catturare i requisiti, ma possono anche essere usati per descrivere qualsiasi interazione con un utente esterno al sistema.
- Incentrato sull'architettura, che descrive ciò che è statico e dinamico nell'intero sistema.
- Approccio iterativo e incrementale.

RUP si suddivide nelle seguenti fasi, rappresentate in ***Figura 2.3***:

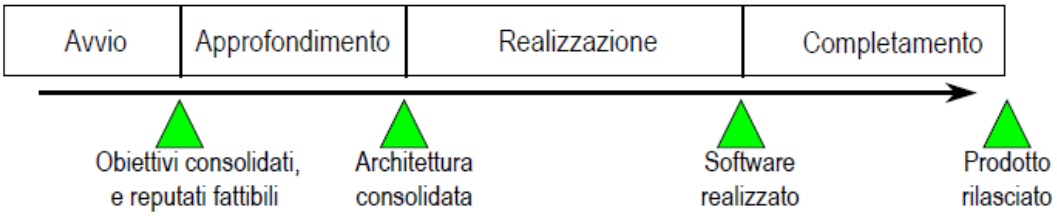

Figura 2.3: Evoluzione delle fasi di RUP.

• **Avvio**

La "fase di avvio" è simile alla "fase di analisi di fattibilità" del modello a cascata. Lo scopo principale è quello di comprendere nel modo più accurato possibile il tipo di mercato al quale il progetto afferisce e identificare gli elementi importanti affinché esso conduca a un successo commerciale.

• **Approfondimento**

La fase di approfondimento o elaborazione definisce la struttura complessiva del sistema. Durante questa fase si approfondiscono i requisiti stimati in precedenza e si progetta una prima versione dell'architettura del sistema, in modo da analizzarne i possibili rischi.

• **Realizzazione**

La fase di realizzazione o costruzione serve a sviluppare le specifiche del sistema stimate nelle fasi precedenti. La fase può essere considerata chiusa quando il committente e tutte le parti interessate concordano che il prodotto software è già completamente realizzato, integrato e testato.

• **Completamento**

Nella fase di completamento o di transizione, il sistema passa dall'ambiente di sviluppo a quello del cliente finale. Vengono condotte attività di test in modo da eliminare errori e ridurre il rischio di rilasciare un prodotto non accettabile.

Tipicamente un progetto gestito usando RUP viene suddiviso in iterazioni, come mostrato in ***Figura 2.4***.

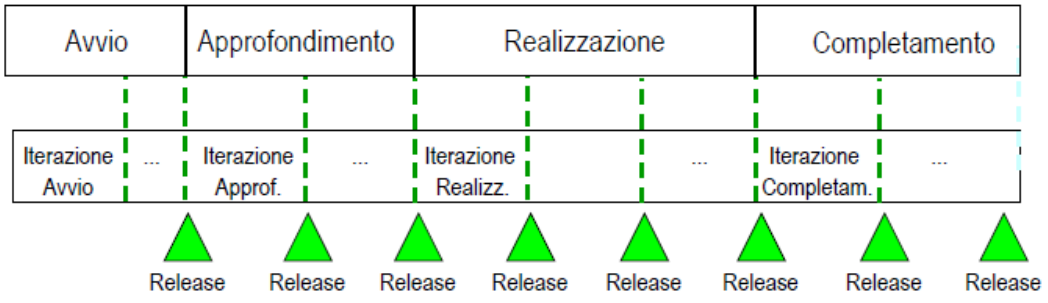

Figura 2.4: Fasi di RUP suddivise in iterazioni.

Ogni iterazione ha un piano, obiettivi e criteri di valutazione. In ognuna di essa si svolgono attività di gestione dei requisiti, analisi, progettazione, codifica e test. Ogni iterazione produce un risultato concreto.

Il metamodello applicato da RUP per descrivere e controllare un processo utilizza quattro concetti cosiddetti "statici", ovvero che sono definiti nello stesso modo per tutti i processi:

- **Ruoli**: un ruolo identifica un certo insieme di responsabilità attribuite a un certo insieme di partecipanti al progetto.
- **Artefatti**: sono il prodotto delle attività del processo; includono documenti, modelli, componenti e altre parti inerenti allo sviluppo del software.
- **Flussi di lavoro**: rappresenta il flusso di esecuzione di un task, il quale identifica lo stato di avanzamento di un'attività.
- **Attività**: sono i compiti specifici portati a termine dai partecipanti del progetto.

2.3 MODELLO AGILE

Il modello agile prevede un continuo contatto con il cliente in modo da facilitare la richiesta di requisiti e la soluzione di alcuni problemi emersi durante sviluppo. I principi su cui si basa, seguendo il Manifesto Agile [4], sono i seguenti:

- Le persone e le loro interazioni sono più importanti dei processi e degli strumenti, ossia le relazioni e la comunicazione tra gli attori di un progetto software sono la miglior risorsa del progetto.
- È più importante avere un software funzionante che la documentazione, è opportuno rilasciare nuove versioni del software a intervalli frequenti, bisogna mantenere il codice semplice e avanzato tecnicamente riducendo la documentazione al minimo indispensabile.
- Bisogna collaborare con i clienti al di là del contratto perché la collaborazione diretta offre risultati migliori dei rapporti contrattuali.

- Bisogna essere pronti a reagire immediatamente alla domanda del mercato e di conseguenza il team di sviluppo dovrebbe essere autorizzato a suggerire modifiche al progetto in ogni momento.

I metodi che derivano dall'applicazione della metodologia agile sono molteplici e dipendono fortemente dal contesto in cui questi devono essere applicati. Benché ciascuno dei metodi agili sia unico nel suo approccio specifico, tutti condividono una visione comune e una serie di valori di base come chiaramente esposto nel Manifesto Agile [4]. Tutte le metodologie incorporano infatti il concetto di iterazione e feedback continuo allo scopo di rilasciare e successivamente raffinare un sistema software. Tutte le metodologie coinvolgono le attività di pianificazione, test e integrazione continua assieme ad altre forme di evoluzione, allo scopo di perfezionare qualsiasi aspetto relativo sia al progetto, sia al software. Tutte le tecniche più diffuse sono simili fra loro e possono essere riconducibili a un numero limitato di principi:

Coinvolgimento del cliente: sono previsti differenti gradi di coinvolgimento del cliente.

Comunicazione diretta: comunicazione fra tutti gli attori del progetto e con il cliente prima di tutto.

Consegne frequenti: effettuare rilasci frequenti di versioni intermedie del software consente di ottenere più risultati contemporaneamente. Da una parte si offre al cliente qualcosa con cui lavorare, distraendolo così da eventuali ritardi nella consegna del progetto completo, dall'altra, lo si può impiegare come "tester" dal momento che, utilizzando il software rilasciato, sarà lui stesso a segnalare eventuali anomalie.

Progettazione e documentazione: sebbene l'importanza attribuita a queste due attività venga sensibilmente ridimensionata nella metodologia agile, sarebbe errato credere che le stesse siano del tutto assenti dal processo di sviluppo.

Automazione: le attività problematiche, come ad esempio la documentazione, possono essere automatizzate. La tecnica per ottenere in maniera automatica la

documentazione a partire dal codice già prodotto è detta “retro-ingegneria”. Questa è una delle pratiche più diffuse e più controverse: diffusa perché permette un guadagno enorme in termini di tempo, ma controversa perché, molto spesso, la documentazione prodotta è inutilizzabile e viene conservata solo per motivi burocratici pur senza avere una reale utilità.

Gerarchia: la scelta di creare una struttura gerarchica all'interno del team di sviluppo è delicata. Se si decide per una struttura gerarchica ad albero, si ottiene la possibilità di gestire un numero molto alto di programmatori e di lavorare a diversi aspetti del progetto parallelamente; se viceversa si decide per una totale assenza di gerarchia si avrà un team di sviluppo molto compatto e motivato, ma questo dovrà essere necessariamente piccolo in termini di numero di programmatori.

Programmazione di coppia: è stato dimostrato che i costi di questa scelta sono inferiori ai benefici che comporta, ma ci sono esempi pratici che indicano come questa pratica possa essere insopportabile per alcuni programmatori e quindi controproducente.

Refactoring: riscrittura completa di parti di codice mantenendone invariato l'aspetto esterno.

Miglioramento della conoscenza: Si sfrutta per migliorare il lavoro quotidiano, imparando ad esempio dai fallimenti.

Semplicità: semplicità nel codice, nella documentazione, nella progettazione, nella modellazione. I risultati così ottenuti comportano una migliore leggibilità dell'intero progetto e una conseguente agevolazione nelle fasi di correzione e modifica;

Controllo di versione: necessità di introdurre un modello, un metodo, uno strumento, per il controllo delle versioni del prodotto software rilasciato.

Un esempio di metodologia agile è “Scrum”. Citando la definizione presente in [5], “Scrum non è un processo o una tecnica per costruire prodotti ma piuttosto è un

framework all'interno del quale è possibile utilizzare vari processi e tecniche". In generale, consta delle seguenti fasi:

- **Sprint:** prevede di dividere il progetto in blocchi rapidi di lavoro alla fine dei quali consegnare una versione al cliente.
- **Backlog:** indica come definire i dettagli del lavoro da fare nell'immediato futuro, ovvero quello che deve essere ancora fatto.
- **Scrum Meeting:** sono organizzate riunioni giornaliere del team di sviluppo per verificare cosa si è sviluppato e cosa deve essere ancora sviluppato.

Nel prossimo paragrafo parleremo di DevOps, una metodologia che riesce ad estendere gli ideali dell'approccio agile oltre il codice, concentrandosi sull'intero servizio.

Mentre la metodologia Agile si basa su presupposti che riguardano valori, principi, metodi e pratiche, DevOps aggiunge anche gli "strumenti". Secondo il Manifesto Agile infatti, questi ultimi assumono una minima importanza, e questo ha portato nel tempo a trascurarli notevolmente con conseguenze spesso non in linea con l'obiettivo del cliente.

Mentre il metodo Agile risolve i problemi legati alla tecnologia, DevOps cerca di risolvere più ampiamente un intero problema di business attraverso un insieme di pratiche, cultura e valori che guidano il cambiamento verso un software più solido, che raggiunge la produzione rapidamente e con meno ostacoli durante il suo ciclo di vita.

2.4 DEVOPS

DevOps [7] [8], abbreviazione dei termini in inglese di "Development" e "Operations", che indicano rispettivamente lo "sviluppo" e la "messa in produzione" di un prodotto software, è un approccio basato su principi "lean" e "agili", grazie ai quali viene facilitata la comunicazione tra un'azienda e i dipartimenti da cui è composta, come sviluppo, produzione e controllo qualità. Il termine "lean", tradotto in

italiano con “snello”, si riferisce ad una filosofia che mira a minimizzare gli sprechi di risorse fino ad annullarli. L’obiettivo è quello di reagire in modo rapido ai cambiamenti del mercato e rispondere in maniera tempestiva alle esigenze degli utenti, analizzando in maniera continua i loro feedback.

Dei benefici che comporta DevOps non ne risentirà soltanto l’azienda, ma anche i clienti, fornitori e partner. Quindi non è solo una capacità IT, ma un vero e proprio processo di business.

Con DevOps bisogna investire in tre aree principali:

- **Rispondere velocemente alle esigenze dei clienti.**
  Offrendo una migliore “customer experience”, differenziata e avvincente, costruisce la fedeltà del cliente e fa crescere la fetta di mercato posseduta dall’azienda. Per fare ciò, bisogna continuamente ottenere e rispondere ai feedback del cliente, attraverso un meccanismo che li acquisisca in maniera semplice e veloce, partendo dall’inizio del processo, passando per lo sviluppo e continuando dopo la messa in produzione.
- **Aumentare la capacità per innovare.**
  Le aziende moderne usano degli approcci “lean”, cioè snelli, che mirano a minimizzare gli sprechi di risorse, per migliorare la loro capacità di innovare. Il loro obiettivo è quello di eliminare lavoro inutile dal sistema ed evitare di eseguire una stessa operazione più volte. Ad esempio, una tecnica comune che sfrutta questo principio prende il nome di “A-B testing” [6], con la quale, in generale, un’azienda chiede ad un piccolo gruppo di utenti di testare diverse versioni di un software. La versione migliore sarà messa in produzione, le altre eliminate.
- **Fare valutazioni in tempi più rapidi.**
  Eseguire quest’attività necessita di sviluppare una cultura che coinvolga pratiche e processi automatici che consentano velocità, efficienza e affidabilità. DevOps, quando è adottato come una capacità di business, fornisce gli

strumenti e la cultura richiesta per pianificare e prevedere qualsiasi tipo di scenario, aprendo la strada verso il successo. Il metodo con cui devono essere eseguite queste valutazioni varia a seconda dell'azienda e del progetto, ma l'obiettivo di DevOps è quello di ottenerle in maniera rapida, efficace ed efficiente.

Il movimento DevOps ha prodotto diversi principi che si sono sviluppati nel tempo e stanno ancora evolvendo. Di seguito ne descriviamo i principali:

1. Sviluppare e testare il software in un ambiente simile a quello di produzione.

L'obiettivo è quello di consentire ai team di sviluppo e controllo qualità di sviluppare e testare il software in un ambiente analogo a quello di produzione, in modo da valutare il comportamento dell'applicazione prima di effettuare il deploy, ovvero prima di installarla all'interno di un sistema informatico aziendale. Dal punto di vista dei team di Operativi, ciò ha un valore inestimabile, in quanto loro sapranno già come si comporta l'applicazione in produzione, e potranno supportarla nel migliore dei modi.

2. Deploy con processi ripetibili e affidabili.

Questo principio consente ai team di Sviluppo e Operativi di eseguire il processo di sviluppo del software in maniera iterativa lungo tutto il suo percorso fino alla messa in produzione. L'automazione è essenziale per creare processi che sono iterativi, frequenti, ripetibili e affidabili. Quindi l'azienda deve creare una "Delivery Pipeline", la quale consente di eseguire test e deploy in modo continuo e automatico. Ciò consente inoltre ai team di testare il processo stesso, diminuendo in tal modo il rischio di fallimenti quando viene rilasciato il prodotto software. Esamineremo la "Delivery Pipeline", e i benefici che comporta, in modo dettagliato nel paragrafo ***2.4.4***.

3. Controllo della qualità.

Le aziende tipicamente dispongono di ottime applicazioni e sistemi che catturano informazioni in tempo reale, ma lo fanno in maniera discontinua. Questo principio

serve a monitorare tutte le fasi di ogni processo tramite test automatizzati, monitorando le caratteristiche funzionali e non funzionali di un'applicazione. Un controllo frequente consente di risolvere i problemi che potrebbero verificarsi in produzione. Perciò, è opportuno che il formato con cui sono raccolti questi dati sia comprensibile ad ogni stakeholder, ovvero ad ogni attore coinvolto nel processo di business.

4. Catturare i feedback degli utenti in modo frequente.

Un obiettivo di DevOps è quello di consentire alle aziende di reagire e fare cambiamenti in modo rapido. Per fare ciò, bisogna ottenere i feedback in maniera rapida, creando un canale di comunicazione che consenta a tutti gli stakeholder di accedere e reagire tempestivamente sul feedback ricevuto. Ad esempio:

- Il team di sviluppo aggiusta i suoi piani o le priorità sul progetto.
- Il team operativo migliora l'ambiente di produzione.
- I dirigenti modificano le strategie di business e i piani di rilascio del software.

2.4.1 L'ARCHITETTURA DI RIFERIMENTO DEVOPS

Fissati gli obbiettivi del business, un'azienda deve esaminare quali cambiamenti deve apportare prima di adottare un approccio DevOps. In questo paragrafo esamineremo l'architettura di riferimento di questo nuovo movimento culturale, presentata in [7] e in modo più dettagliato in [8]. Essa aiuta i professionisti ad utilizzare le linee guida, le direttive e altro materiale necessario per il design di una piattaforma Devops che ospita persone, processi e tecnologie. Le funzionalità dell'architettura di riferimento sono espresse attraverso i suoi vari componenti, i quali potrebbero lavorare in modo indipendente oppure tutti insieme. Ogni componente fornisce una funzione ben precisa, che analizzeremo nel dettaglio, e ad ognuno di essi sono assegnate delle persone, a cui sono richieste determinate competenze e svolgeranno operazioni ben precise in modo sistematico tramite strumenti di automazione.

L'architettura di riferimento di DevOps, mostrata in ***Figura 2.5***, è formata dai seguenti componenti:

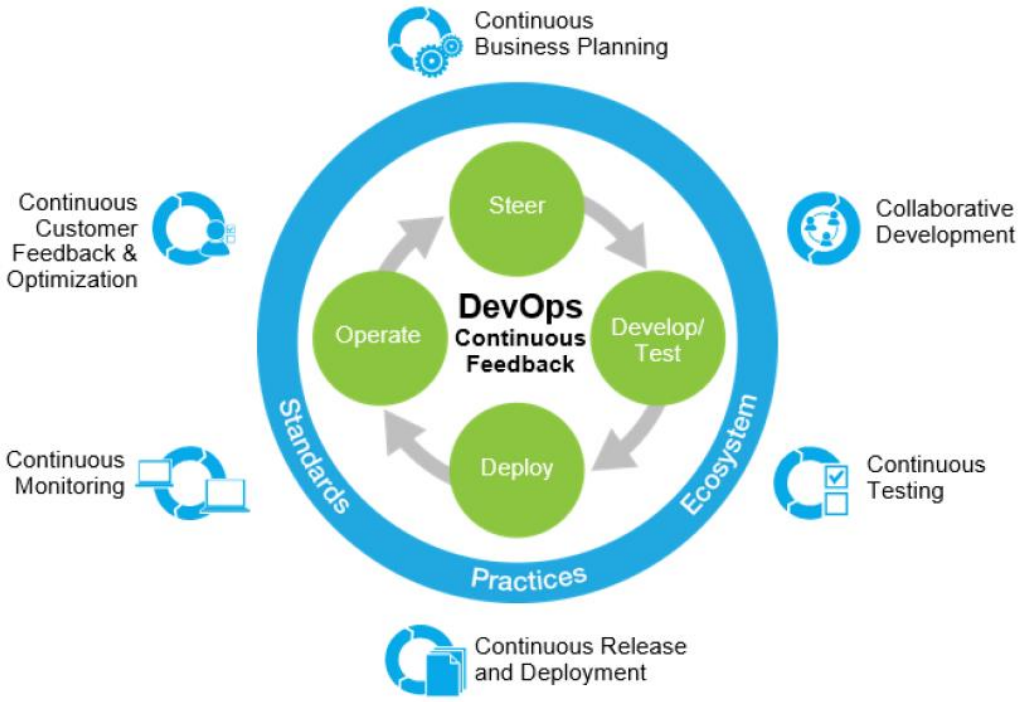

Figura 2.5: Architettura di riferimento DevOps [7].

Steer: il termine, che tradotto in italiano significa "Orientamento", indica la fase che consiste nello stabilire gli obiettivi del business e modificarli in base ai feedback dei clienti. Questa pratica prende il nome di "continuous business planning", ovvero pianificare il business in maniera continua. Al giorno d'oggi, un'impresa necessita di essere agile per poter reagire immediatamente ai fabbisogni dei clienti. Gli approcci tradizionali non consentono di raggiungere questi propositi in quanto, sia i processi e sia le persone, operano in maniera isolata e quindi non hanno la capacità di ripianificare velocemente. Spesso il feedback corretto non è ricevuto tempestivamente e quindi non si può pianificare il corretto investimento a cui dare la priorità. Inoltre, per alcuni team, la pianificazione è vista come un processo che rallenta la produzione piuttosto che come un'attività che permetta loro di eseguire operazioni in maniera più rapida. Rilasci veloci consentono al business di guadagnare una maggiore "agilità", ma bisogna

gestire anche la velocità con cui vengono eseguiti questi rilasci in modo che ogni operazione sia compiuta correttamente. Non si può infatti mettere in produzione un prodotto software che non sia accuratamente testato e non rispetti tutti gli obiettivi di business prefissati in precedenza.

DevOps aiuta i team a collaborare e a reagire ai feedback dei clienti in maniera continua, migliorando sia l'agilità che i risultati di business. Allo stesso tempo, bisogna gestire i costi. Identificando ed eliminando lavoro inutile dal sistema durante il processo di sviluppo, i team diventano più efficienti e, quindi, diminuiscono i costi.

Develop/Test: l'architettura di riferimento coinvolge due pratiche:

- *Sviluppo collaborativo.* Le operazioni da svolgere per effettuare il rilascio del software coinvolge un ampio numero di team come quello di sviluppo, operativo, controllo qualità, sicurezza, fornitori, partner e altri ancora. I dipendenti di questi team lavorano su più piattaforme e potrebbero operare in luoghi differenti. Lo sviluppo collaborativo consente ai dipendenti di lavorare insieme tramite pratiche, strumenti e piattaforme che possono usare per creare e rilasciare il software.

  La caratteristica principale dello sviluppo collaborativo è la "continuos integration", ovvero "integrazione continua", mostrata in ***Figura 2.6***, una tecnica con la quale gli sviluppatori software continuamente integrano il proprio lavoro con gli altri membri del team di sviluppo. Quindi, l'obiettivo è quello di integrare e testare, regolarmente e frequentemente, il lavoro di ogni membro del gruppo. In caso di lavori complessi, sviluppati su più sistemi, il risultato di questo tipo di approccio fa emergere i problemi immediatamente, sia quelli conosciuti che quelli sconosciuti, in modo da intervenire subito per risolverli.

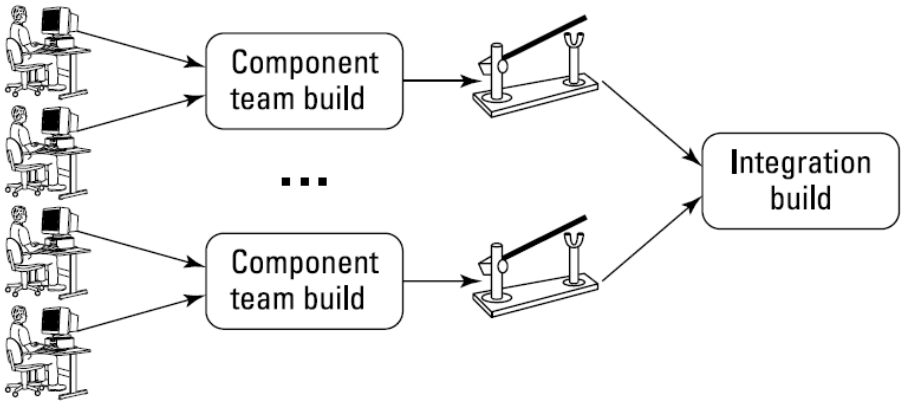

Figura 2.6: Collaborazione tramite "continuous integration" [7].

- *Continuos Testing.* Significa testare l'applicazione continuamente durante tutto il ciclo vita. Il risultato di quest'attività, oltre a ridurre i costi, permette di ottenere un feedback continuo sulla qualità del lavoro svolto. Questo processo è facilitato adottando pratiche come test automatizzati e "Service Virtualization", in modo tale da riprodurre un ambiente simile a quello di produzione e, se si presentano dei problemi, si risolvono prima di effettuare il deploy. La virtualizzazione del servizio verrà trattata in modo più dettagliato nel paragrafo ***2.4.5***.

Deploy: questa parte dell'architettura può essere considerata come la "radice" da cui ha avuto inizio lo sviluppo della metodologia DevOps. Le tecniche che consentono il rilascio e il deploy di un prodotto software consentono anche la creazione di una "Delivery Pipeline". Come vedremo nel dettaglio nel paragrafo ***2.4.4***, questa pipeline facilita il "rilascio continuo" del software, in maniera efficiente ed automatica. L'obiettivo è quello di consegnare nuove caratteristiche di un'applicazione agli utenti quanto prima è possibile.

Gestire l'automazione del deploy di un'applicazione da una fase alla successiva richiede strumenti specializzati, come:

- *Deploy automatico*: gli strumenti per l'automazione del rilascio sono fondamentali nello spazio DevOps. Questi programmi, oltre a gestire l'intero

processo, tengono traccia di quale versione dell'applicazione è stato effettuato il rilascio e su quale nodo fisico o virtuale, in ogni stadio della delivery pipeline. Gestiscono inoltre le configurazioni degli ambienti di tutte le fasi per le quali deve essere fatto il deploy dei componenti dell'applicazione.

- *Gestione del rilascio*: gestire i piani di rilascio con i deploy associati richiede un coordinamento in tutta l'azienda, in particolare tra i team di sviluppo, operativi e controllo qualità. Gli strumenti per la gestione dei rilasci consentono alle aziende:
    - Di pianificare ed eseguire rilasci.
    - La collaborazione e lo scambio di informazioni tra tutti gli stakeholder durante il rilascio.
    - Di tracciare tutte le componenti utilizzate, cioè dipendenti e risorse, sia durante la fase di rilascio e sia lungo le fasi che compongono la delivery pipeline.

Operate: questa parte dell'architettura si divide in due pratiche, che consentono di monitorare l'applicazione in produzione e ricevere i feedback dei clienti. L'acquisizione di questi dati consente di reagire immediatamente a dei cambiamenti e a rivedere i piani di business a seconda delle necessità.

- *Continuous Monitoring:* il "Monitoraggio Continuo" fornisce dati in ogni stadio del ciclo di vita. Non si limita solo a raccogliere dati in produzione, ma coinvolge tutte le fasi del processo.
- *Continuous Customer feedback & Optimization:* il termine indica i due principali tipi di informazione che si possono ottenere, ovvero: i dati riguardanti su come il cliente usa l'applicazione e i feedback forniti dall'utente stesso per migliorarla. Le nuove tecnologie consentono, oltre a catturare il comportamento del cliente, di vedere anche dove non è soddisfatto mentre usa l'applicazione. Questi feedback permettono di intraprendere azioni appropriate

in modo da migliorare l'applicazione. Ad esempio, i dirigenti potrebbero modificare i piani del business, gli sviluppatori aggiungere o migliorare funzionalità di un'applicazione, il team operativo migliorare l'ambiente in cui l'applicazione è eseguita. Il "continuos feedback", ovvero l'acquisizione continua delle opinioni degli utenti, è un componente essenziale di DevOps, grazie al quale consente al business di guadagnare agilità e rispondere tempestivamente alle esigenze del cliente.

2.4.2 SCEGLIERE DI ADOTTARE DEVOPS

Adottare qualsiasi nuova metodologia tipicamente richiede un piano che coinvolga persone, processi e tecnologie. Non si può sperare di avere successo, specialmente in ambito aziendale dove sono coinvolti più stakeholder, senza coinvolgere questi tre aspetti fondamentali. In questo paragrafo saranno discussi le persone, i processi e gli aspetti tecnologici di DevOps.

Sebbene il nome DevOps fa pensare solo ai team di sviluppo e operativi, in realtà coinvolge tutti gli elementi che costituiscono un'azienda, tra cui i dirigenti, l'architettura, il design, lo sviluppo, controllo qualità, produzione, sicurezza, partner e fornitori. Escludendo uno qualsiasi di questi attori, interno o esterno all'organizzazione, porta ad una implementazione errata di DevOps.

2.4.2.1 PREREQUISITI

Questa sezione fornisce una guida sui prerequisiti da possedere prima di adottare una metodologia DevOps, al fine di essere in grado di creare una giusta cultura, identificare gli obiettivi di business ed eliminare i "bottleneck", tradotto in italiano con il termine "collo di bottiglia", che sono la causa principale di ritardi e irregolarità. Il termine si riferisce ad un fenomeno che si verifica quando le prestazioni di un sistema o le sue capacità sono fortemente vincolate da un singolo componente, limitando la

performance del componente stesso e dell'intero sistema. Discuteremo di ciò più avanti nel corso del paragrafo.

Tornando a parlare di DevOps, prima di adottare questa nuova metodologia è necessario:

- *Identificare gli obiettivi di business*.
  Il primo compito nel creare una cultura richiede che ognuno prosegua nella stessa direzione e lavori per raggiungere lo stesso scopo. Ciò significa identificare obiettivi di business comuni sia per i dipendenti che per l'azienda. È importante che l'intero team sia motivato a raggiungere gli obiettivi aziendali, i quali non devono essere in conflitto con gli incentivi che hanno i team nel raggiungere determinati traguardi. Quando le persone conoscono per che cosa stanno lavorando e come i progressi verso il raggiungimento della meta comune sono valutati, diminuiscono i problemi e ogni individuo può gestire le proprie priorità. DevOps non è l'obiettivo, ma è ciò che aiuta a raggiungere gli obiettivi.
- *Identificare i bottleneck nella delivery pipeline.*
  All'interno della "delivery pipeline", descritta nel dettaglio nel paragrafo ***2.4.4***, possiamo avere tre tipi di inefficienze:
  - Sovraccarico, quando si comunica ad esempio la stessa informazione.
  - "Rework", che possiamo tradurre in italiano con il termine "correzione", si verifica, ad esempio, quando si scoprono difetti, in fase di test o in produzione, riassegnando lavoro al team di sviluppo.
  - Sovrapproduzione, che si verifica ad esempio quando si sviluppa di una funzionalità di un'applicazione non necessaria.

Uno dei maggiori ostacoli che si incontrano nella "delivery pipeline" è lo sviluppo dell'infrastruttura su cui verrà eseguita l'applicazione. L'adozione di un approccio

DevOps aumenta la velocità di rilascio dell’applicazione e mette pressione all’infrastruttura per rispondere più velocemente alle varie richieste. Per garantire ciò, è necessaria un'automazione rapida e basata su criteri comuni delle attività di sviluppo, test e produzione correlate ad applicazioni e infrastruttura. Per implementare rapidamente l'infrastruttura, i professionisti DevOps stanno cercando di gestirla come codice. L’infrastruttura come codice [9], in inglese “Infrastructure As A Code”, è la risorsa chiave di DevOps, che consente alle aziende di gestire la quantità e la velocità con cui gli ambienti devono essere approvvigionati e configurati per consentire il rilascio continuo del software. In questo modo hanno la possibilità di implementarla, controllarne la versione e disconnetterla, esattamente come avviene per la gestione del software applicativo che sviluppano. Questo potente strumento consente quindi un livello di astrazione completo, configurabile e programmabile per tutte le risorse del sistema aziendale.

Ritornando alla “delivery pipeline”, bisogna cercare di ottimizzare l’intero processo “end-to-end”, ovvero dalla concezione iniziale fino al supporto della produzione. Il “Throghput” di ogni processo, ovvero una stima di efficienza che si ottiene selezionando una risorsa e il numero di task completati su di essa in un determinato intervallo di tempo, deve essere uguale in modo da evitare che si accumuli lavoro in uno stadio qualsiasi della pipeline. Per raggiungere il giusto equilibrio, bisogna individuare le fasi chiave così da poter minimizzare il tempo di attesa ad ogni centro di lavoro della pipeline, stimato in base al tempo in cui la risorsa risulta occupata e inattiva, ottimizzare il lavoro in corso e fare modifiche al flusso di esecuzione in modo da evitare ritardi.

2.4.2.2 LE PERSONE IN DEVOPS

Alla radice, DevOps nasce come un movimento culturale, che coinvolge prima di tutto le persone. Un’organizzazione potrebbe adottare i migliori processi o strumenti possibili, inutili se le persone non eseguissero correttamente i propri compiti. Perciò costruire una cultura è l’aspetto principale di DevOps.

Una cultura DevOps è caratterizzata da un alto grado di collaborazione e fiducia tra i vari ruoli, concentrandosi a raggiungere gli obiettivi di business aziendali piuttosto che quelli dipartimentali, cercando di imparare dai fallimenti, in modo che non si verifichino mai più. Costruire una cultura è diverso rispetto ad adottare un nuovo processo o programma. Occorre una sorta di "ingegneria sociale" tra le persone che compongono i diversi team per garantire che viaggino nella stessa direzione. A causa delle diversità presenti nelle varie squadre di lavoro, costruire una cultura può essere un'impresa ardua. Richiede che la direzione aziendale lavori con ogni team presente nell'organizzazione per creare un ambiente e abitudini di collaborazione e condivisione, eliminando tutte le barriere tecniche e organizzative che impediscono ciò.

Ad esempio, consideriamo come vengono gestite tipicamente le ricompense. I team Operativi ci guadagnano a mantenere il sistema operativo e stabile, quelli di sviluppo invece fornendo nuove funzionalità. Ciò può generare dei contrasti in quanto i gruppi Operativi sono consapevoli che meno cambiamenti sono accettati, minore sarà la probabilità che si verifichino guasti in produzione. Dall'altra parte invece, gli sviluppatori hanno poco incentivo a puntare sulla qualità piuttosto che sulla quantità. Bisogna sostituire ciò con una responsabilità condivisa in modo da rilasciare nuove capacità in modo rapido e sicuro.

Quindi i dirigenti di un'azienda, per consentire la costruzione di una cultura DevOps basata sulla fiducia e collaborazione, devono offrire a tutte le parti interessate la visibilità, sia degli obiettivi da raggiungere, sia sullo stato del progetto a cui si riferiscono, e fornire un insieme di strumenti di condivisione comune, essenziale soprattutto quando i team sono geograficamente distribuiti e non possono lavorare insieme di persona. Spesso, per costruire una cultura DevOps è richiesto alle persone di cambiare. Per coloro che non sono disposti a fare ciò potrebbe essere necessario prendere dei provvedimenti, come ad esempio riassegnare dei ruoli all'interno di un team.

Infine, se l'azienda sceglie di avere un team DevOps, l'obiettivo principale deve essere quello di garantire che operi come un centro di eccellenza, che semplifichi la collaborazione, senza aggiungere un nuovo livello di burocrazia o diventi il team che risolva tutti i problemi, altrimenti sarebbe un modo inappropriato di adottare la cultura DevOps.

2.4.2.3 I PROCESSI IN DEVOPS

In generale, i processi definiscono che cosa le persone devono fare. Anche se un'organizzazione possiede una discreta cultura incentrata sulla collaborazione, se le persone stanno eseguendo compiti errati o stanno facendo la cosa giusta in maniera sbagliata, la probabilità di fallimento è ancora alta.

Un vasto numero di processi sono identificati con DevOps e, di seguito, ne discuteremo i principali.

DevOps come un processo business: come abbiamo ripetuto più volte nel corso del capitolo, DevOps influenza l'intero business, rendendolo più agile e migliorando la capacità di rilascio di un'applicazione. È possibile estendere ulteriormente il concetto di questa metodologia considerandolo come un processo di business, ovvero come un insieme di task che producono uno specifico risultato, che può essere un servizio o un prodotto software, per i clienti. Un'azienda che decide di adottare DevOps deve esaminare il processo che l'azienda usa per effettuare rilasci e identificare le aree di miglioramento, ottimizzando tutte le attività coinvolte e introducendo automazione.

Processo di gestione del cambiamento: il processo di "gestione del cambiamento", è un insieme di attività che ha lo scopo di controllare, gestire e tenere traccia di un cambiamento identificando i prodotti sviluppati che potrebbero necessitare di modifiche, e i processi usati per realizzarlo. Questo procedimento guida il modo in cui i processi DevOps assorbono e reagiscono alle richieste di cambiamento e al feedback dei clienti. Gli approcci di gestione del cambiamento "tradizionali" si riducono solo alla gestione di una richiesta di modifica o riscontro di un difetto, con la limitata

capacità di tracciare gli eventi tra la richiesta di una modifica e il codice associato o ai requisisti richiesti. Questi approcci non consentono di gestire il lavoro in maniera integrata in tutto il ciclo di vita del software e non possiedono la capacità di tenere traccia di tutte le attività coinvolte. DevOps richiede che tutti gli stakeholder, ovvero tutte le parti interessate, siano in grado di visualizzare e collaborare su tutti i cambiamenti da apportare durante tutto il ciclo di vita del software. La gestione del cambiamento in DevOps comprende i processi che consentono la gestione degli elementi di lavoro per tutti i progetti, attività e risorse associate, non solo quelle coinvolte nella richiesta di cambiamento. Esso comprende anche i processi che consentono all'azienda di collegare ogni task, che viene creato o modificato, con lo specifico dipendente che ha eseguito quell'attività, e a tutti i mezzi che ha usato per completarla. Questi processi consentono ad ogni membro del team l'accesso a tutte le informazioni che riguardano il cambiamento, come ad esempio lo stato di avanzamento del lavoro. Supportano quindi una metodologia di sviluppo agile e iterativa.

2.4.3 TECNICHE DEVOPS

Di seguito sono riportate alcune tecniche specifiche che è necessario utilizzare quando si adotta una metodologia DevOps.

Miglioramento continuo: l'adozione di un nuovo processo non è un'azione immediata, ma graduale. Un'azienda dovrebbe sfruttare delle procedure che identifichino le aree dove apportare miglioramenti e, allo stesso tempo, acquisire una conoscenza da ciò che si è deciso di sostituire, sia in positivo che in negativo. Molte aziende possiedono dei team che lavorano appositamente sull'ottimizzazione dei processi, osservando sia il loro funzionamento, sia l'emergere di nuove tecnologie che potrebbero perfezionarli. Altre organizzazioni invece, consentono ai dipendenti di fornire delle autovalutazioni e suggerimenti, che possono migliorare il processo di cui fanno parte. Indipendentemente dal metodo utilizzato, l'obiettivo è quello di permettere il miglioramento continuo dell'azienda.

Pianificazione del rilascio: la pianificazione del rilascio rappresenta un'operazione critica, guidata dalle necessità di business di offrire le funzionalità richieste dai clienti nei tempi prestabiliti. Pertanto, le aziende hanno bisogno di un piano di rilascio ben definito e di processi che lo gestiscano, consentendone la tracciabilità. La maggior parte delle aziende svolge questo compito organizzando meeting con tutti gli stakeholder, a cui vengono presentati i dati raccolti fino a quel momento, di solito in forma di tabulati, ed esaminano tutte le necessità del business, tutte le applicazioni coinvolte durante la fase di sviluppo, lo stato del progetto e i piani di rilascio. Processi ben definiti e l'automazione, tuttavia, eliminano la necessità di questi meeting, in quanto i dati raccolti durante lo svolgimento di qualsiasi attività sono catturati e ordinati in maniera automatica, oltre ad essere consultabili in ogni momento, consentendo dei piani di rilascio più rapidi e prevedibili. Quindi, sfruttando pratiche "lean" e "agili", comporterà consegne più brevi e frequenti che permettono un maggiore controllo sulla qualità.

Integrazione continua: l'integrazione continua, discussa precedentemente nel paragrafo ***2.4.1***, costituisce una caratteristica fondamentale di DevOps, che consente ad ampi team di sviluppatori di eseguire i propri compiti utilizzando strumentazioni e/o tecnologie differenti, anche in sedi diverse. Garantisce inoltre che il lavoro di ogni team sia continuamente integrato con quello svolto da un team differente, e successivamente validato. In tal modo si riducono i rischi in quanto i problemi vengono identificati tempestivamente durante il ciclo di vita del software.

Distribuzione continua: la distribuzione continua è una conseguenza dell'integrazione continua. Indica il processo con cui viene automatizzato il rilascio del software, per collaudarlo, testare l'intero sistema e trasferirlo nell'ambiente di produzione. Le aziende che usano DevOps di solito utilizzano lo stesso processo automatizzato in tutti gli ambienti, per migliorare l'efficienza e ridurre i rischi introdotti da processi inconsistenti. In ambienti di test, automatizzando la configurazione e aggiornando continuamente i dati raccolti, appena si effettua il

rilascio del software vengono subito eseguiti dei test in maniera automatica, velocizzando in questo modo la ricezione del riscontro sul lavoro svolto, grazie al quale si può intervenire immediatamente in caso di problemi.

Test continui: i test continui sono stati introdotti precedentemente nel paragrafo ***2.4.1***. Per consentire il corretto funzionamento di un processo bisogna tenere sotto controllo continuamente aree specifiche, ed eseguire:

- Test dell'ambiente selezionato e configurato.
- Test sui dati gestiti.
- Test di integrazione, funzionalità, performance e sicurezza.

In un'azienda, i team di controllo qualità devono determinare quali processi adottare in ogni area, che possono variare in base al progetto, alle esigenze richieste per eseguire ogni test e sui requisiti dell'accordo sul livello del servizio. Per le applicazioni rivolte ai clienti, ad esempio, può essere indispensabile condurre maggiori test sulla sicurezza rispetto a quelli che vengono normalmente svolti per le applicazioni interne.

Eseguire i test nelle aree sopra elencate richiede uno sforzo maggiore per i progetti che sfruttano una metodologia agile con integrazione continua rispetto a quelli che utilizzano, ad esempio, un modello a cascata, dove i test vengono eseguiti con intervalli temporali più lunghi, tipicamente circa ogni 2-3 mesi. Allo stesso modo, i test di funzionalità sui requisiti e sulle performance per applicazioni complesse, costituite da componenti che hanno diversi cicli di consegna, sono differenti rispetto a quelle più semplici, composte da un solo componente.

Feedback e monitoraggio continuo: le opinioni dei clienti possono essere catturate in diverse circostanze, come ad esempio acquisti, richieste di cambiamento, reclami e valutazioni. Specialmente grazie alla popolarità dei social media, le imprese hanno bisogno di processi ben definiti che registrano i feedback da una grande quantità di fonti, che a loro volta saranno utilizzati per pianificare successivi rilasci del software.

Questi processi necessitano anche di essere abbastanza agili per adattarsi ai cambiamenti del mercato e normativi. I feedback derivano anche dai dati monitorati, che provengono dai server che eseguono l'applicazione, dai team operativi, di sviluppo e controllo qualità, o da strumenti integrati che catturano le azioni degli utenti. Data l'enorme quantità di dati, le aziende hanno bisogno di processi che catturino e usino questi dati per migliorare le loro applicazioni e l'ambiente in cui sono eseguite, evitando di sovraccaricare il sistema.

2.4.4 DELIVERY PIPELINE

La "delivery pipeline" è costituita dalle fasi che compongono il ciclo di vita di un software, dallo sviluppo fino all'effettiva messa in produzione. Queste fasi possono variare a seconda dell'azienda e del progetto, così come il loro livello di automazione.

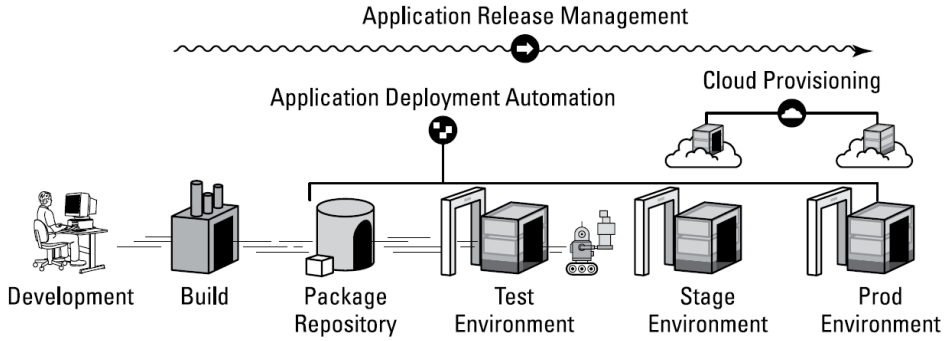

Figura 2.7: Fasi di una tipica DevOps "delivery pipeline" [7].

Una tipica delivery pipeline, mostrata in ***Figura 2.7***, è costituita dalle seguenti fasi:

- **Ambiente di sviluppo:** fornisce agli sviluppatori gli strumenti necessari per scrivere e testare il codice. Oltre all'ambiente di sviluppo integrato che gli sviluppatori useranno per scrivere il codice, questa fase include anche gli strumenti che consentono lo "sviluppo collaborativo", come: gestione del codice, test e pianificazione. Le applicazioni in questa fase possono essere utilizzate su qualsiasi piattaforma e sviluppate con qualsiasi tecnologia, scelta a seconda delle caratteristiche del progetto.

- **Build:** indica la fase in cui il codice implementato viene compilato. Se non si sono verificati problemi, una prima versione del progetto viene creata e testata.
- **Package repository:** è un meccanismo di archiviazione creato durante la fase di compilazione. Nel repository sono memorizzate anche le risorse associate al codice per facilitare il deploy, come ad esempio i file di configurazione.
- **Test environment:** si riferisce all'ambiente dove i team di sviluppo e controllo qualità eseguono dei test. Molteplici sono gli strumenti utilizzati in questa fase, a seconda delle esigenze richieste. Alcuni esempi di applicazioni usate per questo scopo sono:
    - *Gestione dell'ambiente di test:* questi strumenti software facilitano la scelta e la configurazione degli ambienti di test.
    - *Gestione dei dati di test:* è una funzione essenziale per qualunque azienda che adotta pratiche di "continuous testing". Il numero di test che possono essere condotti, e la frequenza con cui vengono eseguiti, sono limitati dalla quantità di dati che sono disponibili per i test e dalla velocità con cui questi dati possono essere aggiornati.
    - *Test di integrazione, funzionalità, performance e sicurezza:* strumenti automatici sono disponibili per ognuno di questi tipi di test. Queste applicazioni dovrebbero essere integrati con un altro strumento che gestisce i test sulla risorsa impiegata, dove sono memorizzati tutti gli scenari dei test condotti e i risultati ottenuti, in modo da facilitarne la tracciabilità e intervenire nel punto esatto quando si verifica un'anomalia.
    - *Virtualizzazione dei servizi:* le applicazioni moderne sono sistemi complessi che dipendono da altre componenti, come applicazioni, server, database e così via. Purtroppo, quando vengono eseguiti i test, queste componenti potrebbero non essere disponibili oppure costose.

La virtualizzazione dei servizi simula il comportamento, in termini di funzionalità e prestazioni, delle componenti non disponibili all'interno dell'applicazione, permettendo di testarla completamente. Daremo maggiori informazioni a riguardo nel paragrafo ***2.4.5***.

- **Stage and production environment:** in questa fase viene eseguito il deploy del software. Gli strumenti utilizzati comprendono le applicazioni per la scelta e la gestione dell'ambiente. Con la nascita delle tecnologie di virtualizzazione e il Cloud, l'ambiente di produzione può essere costituito da centinaia o anche migliaia di server. Strumenti di monitoraggio consentono alle aziende di controllare, dopo il rilascio, le applicazioni in produzione.

2.4.5 DEVOPS E IL CLOUD

DevOps e il Cloud semplificano l'uno il compito dell'altro. Se un'azienda decide di sfruttare il Cloud per ospitare un carico di lavoro DevOps i benefici risultano evidenti. La flessibilità, elasticità, agilità e i servizi offerti da una piattaforma Cloud consentono di "alleggerire" il carico di lavoro nella delivery pipeline di un'applicazione. Gli ambienti per lo sviluppo, test e produzione possono essere forniti e configurati come e quando necessario, minimizzando i bottlenecks. Le aziende infatti stanno cercando di sfruttare questo servizio sia per ridurre i costi degli ambienti di sviluppo e test, sia per fornire ai propri dipendenti un ambiente di lavoro moderno in continua evoluzione. Quindi grazie alla sinergia tra le due parti si crea un'idea di business avvincente.

Come ampliamente affermato nel corso del capitolo, l'obiettivo principale di DevOps è quello di rendere il processo che gestisce il ciclo di vita del software più efficiente e "snello", eliminando qualsiasi tipo di lavoro inutile dal sistema aziendale. Uno dei maggiori ostacoli per le aziende è la disponibilità e configurazione di un ambiente, che può introdurre lunghe fasi di stallo durante il ciclo di vita del software. Uno dei principi su cui si basa DevOps è quello di sviluppare e testare in uno spazio che sia simile a quello di produzione. La mancanza di disponibilità degli ambienti comporta la

formazione di tempi di attesa per i dipendenti. Invece la mancata corrispondenza tra gli ambienti di sviluppo e produzione può introdurre problemi legati alla qualità in quanto gli sviluppatori, nella fase di sviluppo, non possono verificare correttamente come l'applicazione si comporterà in produzione.

Il Cloud affronta questi problemi nei seguenti modi:

- La velocità con cui le piattaforme cloud forniscono un ambiente consentono ai dipendenti di avere uno spazio sempre disponibile e di richiedere l'accesso "On Demand".

- La capacità di selezionare e deselezionare dinamicamente questi ambienti, in base alle esigenze, consente una migliore gestione degli ambienti stessi e riduce i costi, che si avrebbero in caso di ambienti statici e permanenti.

- Consente di sfruttare tecnologie che permettono all'azienda di definire o di creare una nuova versione dell'ambiente, che si adatti perfettamente alle esigenze dei propri dipendenti, come ad esempio creare un ambiente simile a quello di produzione.

- Sono disponibili tecnologie che consentono il deploy automatico di un'applicazione. Esse individuano l'ambiente corretto e effettuano il rilascio dell'applicazione quando necessario, configurando automaticamente anche ambiente e applicazione in base alle esigenze.

- La disponibilità della tecnologia di virtualizzazione dei servizi, che operano in sinergia con gli ambienti Cloud, consente la simulazione dei servizi che sono necessari per i test senza dover disporre delle istanze reali dei servizi stessi.

La ***Figura 2.8*** mostra come gli ambienti Cloud lavorano in combinazione con le tecnologie per eseguire il rilascio automatico e la virtualizzazione del servizio per fornire ambienti di sviluppo o di test durante tutto il ciclo di vita del software.

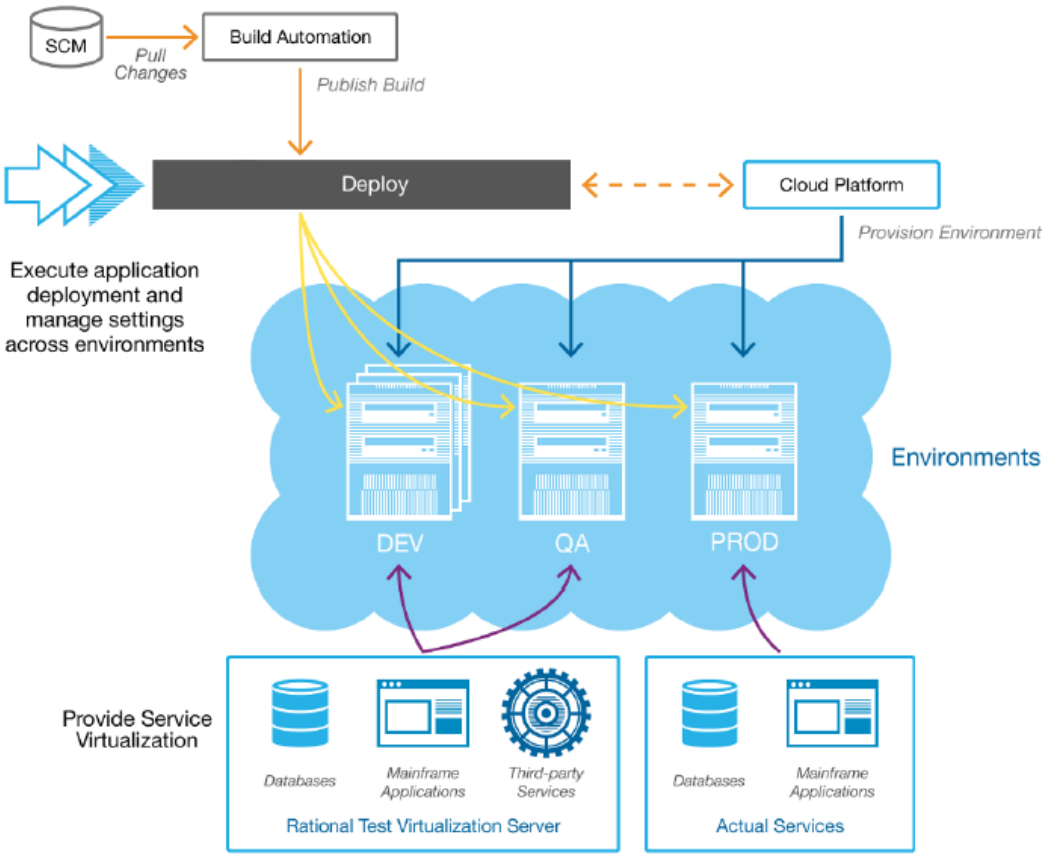

Figura 2.8: Fasi di sviluppo e Test in ambiente Cloud [7].

2.4.6 DIFFERENZE CHIAVE TRA DEVOPS E I MODELLI DI SVILUPPO SOFTWARE TRADIZIONALI

Principalmente, DevOps si differenzia dagli altri approcci IT tradizionali in questi punti [10]:

Frequenza dei rilasci: gli approcci IT tradizionali privilegiano i rilasci lunghi. Primo, perché le aziende adottano principalmente un modello a cascata, che per sua natura richiede molto tempo. Secondo, dato che i rilasci sono costosi, il team operativo consente agli sviluppatori poche finestre di rilascio del software all'anno. Terzo, vi è una mentalità che se "fai le cose in grande" è un modo per ottenere una promozione. Come risultato, le aziende di sviluppo massimizzano la produttività attraverso la

pianificazione di grandi progetti, la quale comporta la scrittura di grandi quantità di codice implementato in maniera frettolosa in un solo rilascio, che risulta bloccato in produzione.

Un'organizzazione DevOps possiede invece un punto di vista opposto. Sono consapevoli che grandi rilasci sono complessi, rischiosi e difficili da coordinare. Invece rilasci più brevi sono più semplici, facili da gestire, testati rigorosamente e meno rischiosi. Se si verifica un problema, l'impatto è minimo e può essere risolto immediatamente. Quindi, le aziende sono in grado di effettuare rilasci più frequenti e rispondere subito alle esigenze dei clienti.

Collaborazione: le aziende che adottano approcci IT tradizionali generalmente sono suddivise al loro interno in dipartimenti di lavoro isolati, in cui ogni dipendente possiede le stesse abilità. Grazie a ciò l'esecuzione di un lavoro viene effettuata gestendo in maniera appropriata le risorse disponibili, beneficiando di economie di scala. Tuttavia, questa ottimizzazione dei costi non sempre porta ai benefici sperati. In un tipico ambiente IT, lo sviluppo di una nuova caratteristica di un software transita nei vari reparti più volte prima di essere effettivamente rilasciata al cliente. Non è raro infatti che il codice implementato trascorra la maggior parte del tempo in attesa o "rimbalzi" da un reparto ad un altro.

Un'azienda DevOps è costituita invece da gruppi composti da sviluppatori, tester, analisti e operatori. Un task è completato senza che transiti da un reparto ad un altro. In questo modo viene incrementata la produttività e di conseguenza abbattuti costi.

Pianificazione: fornire una programmazione delle attività efficiente è l'obiettivo principale di un'azienda IT tradizionale. Date le risorse aziendali, bisogna capire come allocarle per ogni progetto. Per fare ciò, le aziende hanno investito in sofisticati sistemi di pianificazione, spesso imprecisi, che richiedono troppo tempo per gestirli e finiscono per diventare le bottleneck che stanno cercando di alleviare.

In DevOps, la programmazione è gestita in maniera semplice grazie alla combinazione tra i rilasci brevi, gruppi dedicati e processi automatici. Primo perché la previsione è limitata ad un futuro molto prossimo (circa 2-3 settimane). Secondo, non vi sono conflitti di interessi all'interno del team dato che si tratta di un gruppo dedicato al completamento di uno specifico task. Terzo, non vi è nessuna attesa per l'infrastruttura in quanto viene fornita automaticamente. Quarto, grazie all'automazione, il tempo per gestire o prendere delle decisioni diminuisce drasticamente.

Rilascio: in un'azienda IT tradizionale, rilasciare il software in produzione è un evento ad alto rischio. Quando si verifica un problema, bisogna risolverlo immediatamente. Questo processo è gestito strettamente dagli alti livelli aziendali e richiede la partecipazione di tutta l'azienda. In alcuni casi vengono allestite delle "war rooms" che monitorano il processo continuamente prima e dopo il rilascio.

Le organizzazioni DevOps invece considerano il rilascio del software come un "non evento". Riducono i rischi integrando quotidianamente il codice, automatizzando i test, garantendo che tutti gli ambienti siano sincronizzati e riducendo le dimensioni dei rilasci. In altre parole, il codice passerà ad una fase successiva solo quando si ha la certezza che lavorerà in produzione. In questo modo, anche se si dovessero presentare dei problemi sarebbero risolti immediatamente. Inoltre sono in grado di introdurre rapidamente nuove funzionalità all'interno del sistema.

Informazione: entrambi i tipi di organizzazione generano e condividono una quantità enorme di dati. La differenza sta nel come questi dati vengono utilizzati. Nell'IT tradizionale, ogni tipo di informazione generata viene inserita in un report che non sarà mai consultato semplicemente perché contiene troppi dati e non vi è abbastanza tempo per fare questo tipo di lavoro. Quindi le informazioni non sono sfruttate per intraprendere determinate azioni.

Un'azienda DevOps invece, i dati vengono creati e gestiti all'interno dei team, che raccolgono solo le informazioni necessarie, anche in maniera automatica.

Cultura: la priorità delle aziende IT tradizionali è quella di evitare di danneggiare il business. Questo è il motivo principale per il quale investe in processi in grado di prevenire fallimenti. Nonostante ciò, si hanno ritardi nei rilasci, problemi di qualità e di infrastruttura.

Un'organizzazione DevOps cerca anch'essa di prevenire fallimenti, ma riesce a capire anche quando è inevitabile. Pertanto, invece di cercare di eliminare il fallimento, preferisce scegliere quando e come fallire. Preferisce fallire presto e riportare il sistema in uno stato funzionante al più presto.

Metriche: le metriche tradizionali cercano di capire cosa si può ottenere con il minor costo. Cercando di mantenere una capacità IT costante con il minor costo, ha comportato tagli nel settore e la crescita dell'outsourcing.

Le aziende DevOps hanno aggiunto invece una nuova metrica: il "flusso" di esecuzione del lavoro, grazie al quale riescono ad avere sotto controllo l'intero ciclo di vita del software, identificando le aree problematiche, migliorando la produttività e la qualità, eliminando lavoro inutile nel sistema e concentrandosi sulle attività che aggiungono valore all'azienda.

Esecuzione di un lavoro: nelle aziende IT tradizionali significa rispettare la "deadline" senza preoccuparsi della qualità.

Con DevOps invece, grazie alla formazione di team dedicati, qualora dovesse verificarsi un problema il responsabile non sarebbe un singolo membro, ma l'intero gruppo. Ciò garantisce alta qualità.

2.5 RIASSUNTO

In questo capitolo abbiamo visto come la metodologia DevOps, se utilizzata nel modo giusto, può essere uno strumento potente per le aziende. Con investimenti in aree ben precise, il processo che gestisce il ciclo di vita del software può raggiungere la giusta maturità consentendo sia rilasci veloci, e sia di reagire immediatamente ai

cambiamenti del mercato. DevOps è prima di tutto un movimento culturale e, come tale, necessita che le persone siano disposte a cambiare. L’azienda deve eliminare le barriere che ostacolano la comunicazione e la collaborazione, cercando di far coincidere gli obiettivi aziendali con quelli individuali.

Nel prossimo capitolo parleremo di EDEN [2], acronimo di “End-to-end Devops ENgeneering”, ovvero un approccio “model-based” in grado di pianificare e guidare i processi che costituiscono il ciclo di vita del software tramite un metodo DevOps, sfruttando modelli consolidati nel campo dell’ingegneria del software, come i diagrammi UML [15] e le reti di Petri [16].

3 EDEN

Abbiamo visto che DevOps può essere classificata come una strategia di ingegneria del software che consente soprattutto agilità, continuità e collaborazione tra i vari membri di un'azienda, come ad esempio tra i team operativi, di sviluppo e controllo qualità. Sebbene i benefici di questa nuova cultura siano chiari, non esistono ad oggi modelli e metodi in grado di verificare effettivamente che i processi, che costituiscono il ciclo di vita del software, migliorino se si adotta un approccio DevOps. In questo contesto si inserisce EDEN [2], acronimo di "End-to-end Devops ENgineering", ovvero un approccio "model-based", in grado di pianificare e guidare i processi che costituiscono il ciclo di vita del software tramite un metodo DevOps, sfruttando modelli consolidati nel campo dell'ingegneria del software, come i diagrammi UML [15] e le reti di Petri [16].

Il paragrafo ***3.1*** introduce EDEN, focalizzandosi nel contesto in cui si inserisce, mentre il paragrafo ***3.2*** ne descrive le caratteristiche. Infine il paragrafo ***3.3*** descrive i modelli generati da EDEN e le analisi che si possono condurre. Ci soffermeremo in generale sui diagrammi di sequenza e le reti di Petri, rispettivamente nei paragrafi ***3.3.1*** e ***3.3.2***, e nel paragrafo ***3.4*** illustreremo le regole per trasformare il primo diagramma nel secondo. Utilizzeremo questi modelli per condurre l'analisi di un caso di studio DevOps nel prossimo capitolo.

3.1 INTRODUZIONE

Prendendo come riferimento la definizione di [8], DevOps è un insieme di pratiche che consentono di ridurre il tempo che intercorre tra sviluppare un cambiamento per un

sistema e inserirlo nell'ambiente di produzione, garantendo alta qualità. Nonostante all'inizio DevOps ha suscitato un forte interesse nelle aziende dato i benefici che comporta, tuttavia risultano esserci ancora delle carenze su come adottare questa nuova metodologia sfruttando le risorse che un'organizzazione già possiede. Assumendo che questa consapevolezza esista, le aziende considerano DevOps solo come "un'altra strategia". Quest'indifferenza nasce dal fatto che gli approcci e i modelli precedenti possiedono procedure valide, che permettono di sviluppare un software in maniera efficiente.

Tramite un'indagine condotta in [2], è stato riscontrato che:

- È ancora difficile per un'azienda adottare DevOps facendo uso delle risorse che possiede senza causare problemi.
- Il passaggio a DevOps avviene applicando piccoli cambiamenti.
- Non è detto che adottando DevOps si ottengano i benefici sperati.

Da questi punti, in [2] si sostiene che:

- Gli approcci di modellizzazione sono abbastanza agili e possono essere usati per astrarre tutti i processi esistenti e gestire DevOps.
- Gli approcci di modellizzazione possono essere costruiti in maniera incrementale ed essere raffinati in maniera automatica in modo da velocizzare il passaggio da un modello classico a DevOps.
- I modelli possono perfettamente rappresentare i processi di DevOps.

In risposta a questi problemi e assunzioni, la domanda sorge spontanea: "quale tipo di modello ci aiuta ad adottare una metodologia DevOps?". È qui che entra in gioco EDEN [2], acronimo di "End-to-end Devops ENgeneering", cioè un approccio che

riusa i processi di modellazione esistenti con l'obiettivo di pianificare e gestire gli sforzi sostenuti per adottare DevOps sin dall'inizio.

A prima vista, l'idea di usare EDEN per gestire DevOps risulta abbastanza semplice. Primo, produce un modello dell'architettura software, usato per rendere visibili i processi che la compongono e capire come gestire la suddivisione del lavoro. Ad esempio un abbozzo dell'architettura può essere ottenuto tramite "Function-Points" [11], ovvero un tipico modello che i professionisti software adottano per stimare i costi e gli sforzi per sviluppare un software, che deve anche considerare le questioni operative. Similmente, l'architettura che rappresenta l'organizzazione di un'azienda e le risorse impiegate può essere dedotta dalle "Kanban Boards" [12] e da strumenti che supportano modelli agili come ad esempio "Trello"[1] o "Asana"[2], ovvero applicazioni progettate per consentire ai team di tracciare il loro lavoro.

Secondo, trasforma il modello precedente in modelli di processo DevOps. Ad esempio, le informazioni codificate nei modelli che rappresentano l'architettura, e le attività da cui è composta, possono essere inseriti e valutati in una rete di Petri per decidere quale combinazione di task consente lo sviluppo del progetto rispettando i requisiti richiesti. Questo passaggio può essere fatto in maniera:

- Automatica, trasformando un modello con un altro.
- Trasparente, grazie all'utilizzo di strumenti specifici.
- Semplice, in quanto i modelli rappresentano i processi di cui dispone l'azienda.

Come conseguenza, EDEN può essere usato per rappresentare il principio di "analisi continua" di DevOps di cui abbiamo parlato nel capito precedente nel paragrafo ***2.4***, che sfrutta questi modelli per pianificare efficientemente la suddivisione del lavoro, i task richiesti e, allo stesso tempo, la configurazione migliore per svolgerli, in accordo con le richieste da soddisfare sviluppando il progetto.

[1]https://trello.com/
[2]https://asana.com/

EDEN mostra grandi promesse:

- Potrebbe essere meno faticoso per le aziende adottare DevOps usando strumenti di modellazione che già possiedono.
- Consentirebbe alle aziende di pensare in maniera DevOps. Ad esempio, valutando come distribuire i task tra i suoi dipendenti nella fase iniziale del progetto, consentirebbe alle aziende di combinare metodologie di sviluppo agili con strumenti e modelli provati di ingegneria del software, dato che l'azienda già li possiede.

3.2 CARATTERISTICHE

Sebbene DevOps sia un approccio basato sulla collaborazione, assumiamo che sia rappresentato con il processo in ***Figura 3.1***, dove:

a) La fase iniziale è dedicata alla pianificazione e alla stima dei costi.

b) Il concetto di task è l'astrazione dominante in un approccio DevOps.

c) I task e le attività, siano esse svolte dai team di sviluppo o operativi, hanno vita propria, cioè sono assegnati, sviluppati, testati, completati, riassegnati e così via.

d) Ogni task è subordinato al precedente. Ad esempio un membro del team operativo può svolgere il task solo se il corrispondente task svolto da un membro del team di sviluppo è stato completato.

e) Da una prospettiva di modellazione, qualsiasi attività chiave in un processo DevOps potrebbe essere ulteriormente elaborata sfruttando EDEN con altri modelli più concreti. Ad esempio il riquadro in ***Figura 3.1*** mostra un modello DevOps elaborato mediante il modello agile Scrum [5].

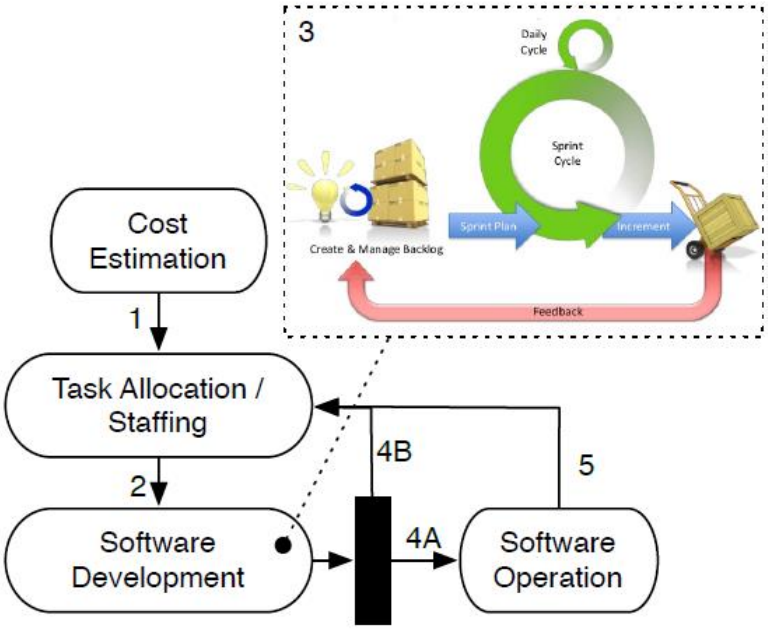

Figura 3.1:Un tipico ciclo di vita DevOps [2].

Seguendo queste assunzioni, fornire un metodo DevOps significa supportare le attività descritte precedentemente. Seguendo la ***Figura 3.1***, EDEN immagina:

1. Modelli di supporto dove gli interessi dei team operativi e di sviluppo sono valutati insieme, cioè bisogna considerare entrambi i reparti per pianificare lo sviluppo del software.

2. Supportare l'allocazione dei task e l'associazione con i dipendenti in maniera dinamica, la fase 2 nella ***Figura 3.1***, utilizzando ad esempio le reti di Petri e le stime ottenute nella fase iniziale del processo. Utilizzando i costi iniziali e le previsioni sulle risorse da impiegare, EDEN può offrire delle basi sullo stato del processo da monitorare e il tempo richiesto prima che possa considerarsi concluso. Le reti di Petri mostrano lo stato dei task e delle funzioni (l'ordine di esecuzione dei task nel processo è stabilito dal responsabile del progetto) insieme al tempo previsto richiesto per concludere il processo e la percentuale di processi già completata. In più, EDEN potrebbe sfruttare le informazioni in

tempo reale ricevute da strumenti DevOps per fare delle raccomandazioni o correzioni.

3. Supportare lo sviluppo, test e funzionamento del software, rappresentati dalle fasi 3, 4 e 4B in ***Figura 3.1***, che, attraverso l'uso di raccomandazioni, permetterebbe di reagire immediatamente alle richieste del mercato o dei clienti. Ad esempio, un sistema di raccomandazione basato su EDEN potrebbe essere costruito per progettare quali sono le funzionalità più utili da fornire agli utenti attraverso uno strumento che sfrutta un algoritmo che trasforma le "user stories" in diagrammi di sequenza durante una fase del processo di Scrum [13], i quali possono essere facilmente usati per generare casi di test o trasformati in altri modelli per condurre analisi più approfondite.

4. Supportare il principio di progettazione continua di DevOps, fase 5 in ***Figura 3.1***, attraverso la gestione di informazioni aggiuntive utili per apportare modifiche al sistema, dando la precedenza alle caratteristiche da cui si trae maggior beneficio, in termini di costi e tempi, secondo le raccomandazioni fornite da EDEN in maniera automatica.

Di conseguenza, EDEN è un approccio che permette di utilizzare gli strumenti di modellazione posseduti da un'azienda per guidare il ciclo di vita del software secondo la metodologia Devops, tramite un'analisi continua e basata sulle evidenze. I primi benefici offerti da EDEN per sfruttare il principio di analisi continua di DevOps sono il monitoraggio sullo stato del processo e il sistema di raccomandazione.

Per monitorare lo stato del processo, EDEN mostra lo stato dei task e delle funzioni, considerando l'ordine in cui devono essere svolte, insieme ad una stima del tempo previsto a concludere il processo e la percentuale dei processi già completati. Questa caratteristica usa sia le informazioni sui costi iniziali e sugli sforzi previsti, sia quelle in tempo reale: usando le previsioni iniziali si crea la base dello stato del processo,

la stima del tempo previsto per eseguirlo e le raccomandazioni, che saranno ulteriormente raffinate durante il processo DevOps che ottiene le informazioni sullo stato.

Come sistema di raccomandazione, che usa le funzioni del sistema non ancora utilizzate, i task di sviluppo o operativi ad esse connessi e i modelli che li rappresentano, EDEN usa queste informazioni per proporre un ordine di programmazione dei task su scala temporale e l'assegnazione di essi agli sviluppatori, tenendo anche conto delle abilità migliori possedute da ogni dipendente o della diversa collocazione geografica al fine di evitare ritardi. Ciò può essere fatto in maniera automatica integrando ad esempio in EDEN strumenti che gestiscono i task e il ciclo di vita di un'applicazione come Mylyn[3], che monitora le attività degli utenti e cerca di identificare delle informazioni rilevanti per svolgere uno specifico task.

3.3 MODELLI EDEN

Con EDEN vi è la creazione di modelli interni che rappresentano le informazioni sui processi software disponibili. Questi modelli sono poi analizzati, ad esempio per stimare le performance di un processo, quando ne è prevista la conclusione o per riallocare o modificare la priorità di un task. Il ciclo di vita di un task inizia quando il task è identificato, ovvero quando è inserito in EDEN o estratto da un archivio di gestione del ciclo di vita del software come JIRA[4] o BugZilla[5], ad esempio quando sono riscontrati dei problemi. Quando un task è creato, potrebbe essere in uno stato "scheduled", se un dipendente è stato assegnato per il suo completamento. Uno sviluppatore può lavorare sia ai task che gli sono stati assegnati sia a quelli etichettati come "unscheduled", se non hanno uno sviluppatore associato ad essi. Quando uno sviluppatore inizia a lavorare sul task, per completarlo passa dalla fase di sviluppo a quella operativa, come mostrato in ***Figura 3.1***. I modelli interni di EDEN perfezionano ulteriormente l'attività di sviluppo del software in attività di implementazione e test.

[3]https://eclipse.org/mylyn/
[4]https://www.atlassian.com/software/jira
[5]https://www.bugzilla.org/

Dopo il test, il task procede verso il reparto operativo se il test ha avuto esito positivo, oppure dovrà essere implementato nuovamente in caso di esito negativo. Questo flusso delle attività di uno sviluppatore comporta che ogni task passi attraverso le fasi di sviluppo, test, operativa, e solo dopo di ciò il task può considerarsi effettivamente concluso.

Il diagramma delle classi UML in ***Figura 3.2*** riassume ciò che necessita EDEN, che ha la capacità di considerare le abilità degli sviluppatori e, in futuro, avere la capacità di fare delle elaborazioni in base anche al grado ricoperto nell'azienda. Ogni sviluppatore infatti possiede un valore in base alle sue capacità di implementare, testare e gestirne il funzionamento di una determinata risorsa.

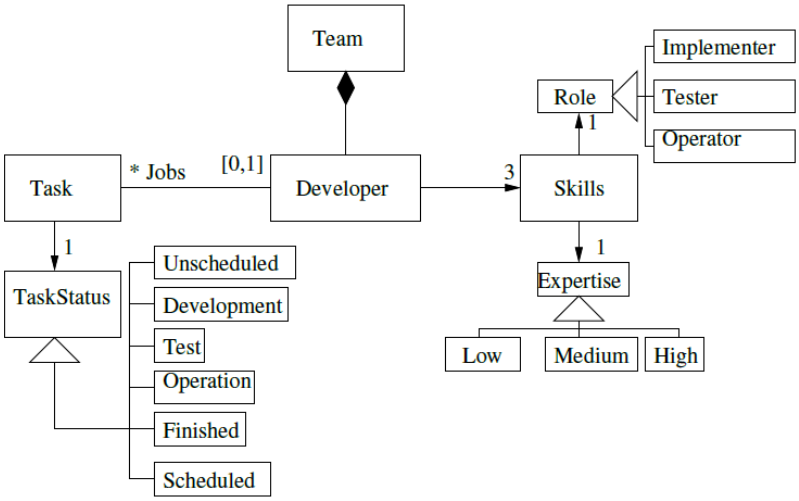

Figura 3.2: Diagramma delle classi di un processo software [2].

Per rappresentare l'evoluzione del ciclo di vita di un task si può utilizzare ad esempio un diagramma di sequenza UML, che analizzeremo nel dettaglio nel prossimo paragrafo.

I modelli UML generati da EDEN sono usati come input per strumenti che trasformano in maniera automatica modelli, trasformandoli in modelli dove possono essere condotte delle analisi più accurate, possibilmente fatte già dallo strumento.

Nell'analisi che condurremo nel Capitolo ***4***, utilizzeremo i diagrammi di sequenza UML per identificare il comportamento di un sistema e mostrare le comunicazioni tra i diversi partecipanti, e le reti di Petri, in quanto sono modelli formali basati su precise teorie matematiche e appropriati per modellizzare e analizzare sistemi. Inoltre, molti metodi di analisi e verifica sono stati sviluppati per le reti di Petri e, tra questi, considereremo quelli che trasformano i diagrammi di sequenza in reti di Petri.

3.3.1 DIAGRAMMI DI SEQUENZA

Un diagramma di sequenza, o "Sequence Diagram" in inglese, è un diagramma appartenente alla famiglia UML, acronimo di "Unified Modelling Language", che viene utilizzato principalmente per rappresentare le interazioni tra gli oggetti, rispettando l'ordine sequenziale in cui avvengono questi scambi di messaggi. Sono utili non solo agli sviluppatori per modellare il comportamento di un'applicazione, ma anche ai dirigenti di un'azienda, in quanto possono riprodurre il comportamento dei vari elementi del sistema che costituiscono il business, mostrando come interagiscono tra loro. Lo staff tecnico di un'organizzazione potrebbe avvalersi di questi diagrammi per documentare un comportamento. Ad esempio, durante la fase di progettazione, architetti e sviluppatori possono utilizzare questo schema per aggiungere o eliminare le interazioni tra le componenti del sistema.

È possibile utilizzare i diagrammi di sequenza a diversi livelli durante il processo di sviluppo:

- Nella fase di analisi, possono aiutare ad identificare le classi di cui un sistema ha bisogno e ciò che gli oggetti di classe fanno nelle interazioni.
- Nella fase di progettazione, spiegano come il sistema funziona per compiere le interazioni.

- Durante la costruzione di un'architettura di un sistema, è possibile utilizzarli per mostrare il funzionamento dei modelli di progettazione ed i meccanismi che il sistema utilizza.

Uno dei principali usi del diagramma è quello di raffinare, in uno o più diagrammi, i requisiti espressi nei diagrammi dei casi d'uso UML, che vengono usati per la descrizione delle funzioni o servizi offerti da un sistema, così come sono percepiti e utilizzati dagli attori che interagiscono col sistema stesso. Ad esempio in EDEN potrebbe essere integrato uno strumento che sfrutti l'algoritmo proposto in [14] per trasformare i due diagrammi in modo da poter condurre analisi più accurate sostenendo il minimo sforzo.

Lo scopo principale è quello di definire sequenze di eventi che portano a qualche risultato desiderato. L'attenzione si concentra su l'ordine in cui si verificano i messaggi piuttosto che sul loro contenuto. Il diagramma trasmette queste informazioni lungo le dimensioni orizzontali e verticali:

- In verticale, dall'alto verso il basso, è evidenziata la sequenza temporale dei messaggi secondo l'ordine in cui si verificano.
- In orizzontale, da sinistra a desta, le istanze degli oggetti a cui sono inviati i messaggi.

Gli elementi che costituiscono un diagramma di sequenza sono descritti nel dettaglio in [15].

3.3.2 RETI DI PETRI

Per modellare ed analizzare le attività di cui è composto un sistema, vi sono diversi strumenti a disposizione, come quello basato sulla teoria delle reti di Petri [16], introdotte nel 1962 da Carl Adam Petri, che presenta notevoli vantaggi. Si tratta, infatti, di una tecnica grafica e, in quanto tale, semplice da utilizzare. Nonostante ciò, si basa su un buon rigore formale, facendo sì che ogni processo sia definito in maniera

chiara ed inequivocabile. Quindi le reti di Petri costringono ad utilizzare definizioni precise prevenendo ambiguità, incertezze e contraddizioni e, grazie alle forti basi matematiche, forniscono un formalismo che permette l'utilizzo di tecniche analitiche.

Le reti di Petri sono spesso utilizzate per modellare e gestire i flussi di lavoro che compongono un sistema [17] e le sue proprietà comportamentali, come conflitti e concorrenza. Seguendo la ***Figura 3.3***, una rete di Petri è un grafo orientato bipartito con due tipi di nodi, che prendono il nome di "posti" e "transizioni", connessi da archi diretti.

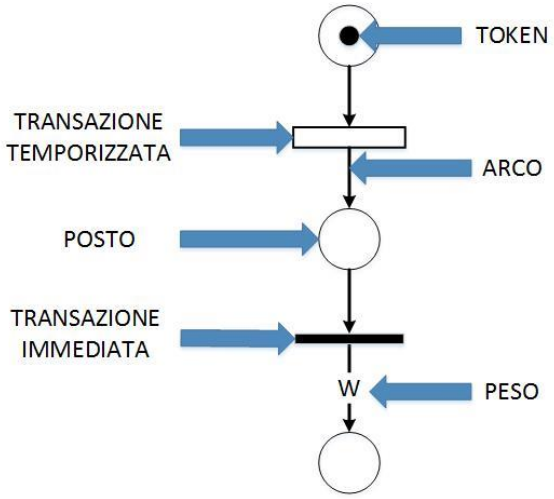

Figura 3.3: Elementi di una rete di Petri.

Un arco può unire solamente nodi di tipo diverso, quindi posti con transizioni e viceversa. I posti sono rappresentati graficamente da cerchi e identificano delle condizioni, mentre le transizioni sono rappresentate da rettangoli e identificano eventi. La rete evolve passando attraverso una serie di stati, e lo stato è rappresentato graficamente collocando dei gettoni nei posti, chiamati "token", rappresentati tramite dei punti di colore nero. Questi gettoni indicano lo stato di avanzamento delle operazioni descritte nei passi del processo. Ad ogni arco può essere associato un peso "w", rappresentato da un numero intero positivo. Qualora la scrittura sia omessa, il peso è considerato unitario.

Nella rete di Petri, un concetto fondamentale è quello di “marcatura”, ovvero una funzione che assegna ad ogni posto della rete un numero intero di token. La configurazione iniziale della rete prende il nome di “marcatura iniziale”.

Quando una transazione è abilitata:

- Consuma i token dal posto di input e li produce nel posto di output, secondo il peso dei rispettivi archi.
- In tutti i posti in uscita dalla transizione si generano un numero di token pari al peso dell’arco che collega la transizione al posto.
- L’istante in cui una transazione è abilitata prende il nome di “firing time”.

Come mostrato in ***Figura 3.3***, esistono due tipologie di transizioni:

1. *Transizioni immediate*, rappresentate con un rettangolino di colore nero, caratterizzate da un “firing time” nullo. Con tale tipologia di transizioni si continuano a modellare gli eventi istantanei che caratterizzano il sistema rappresentato dalla rete di Petri.
2. *Transizioni temporizzate*, rappresentate da rettangolini vuoti, caratterizzate da un “firing time” positivo. Con tale tipologia di transizioni si modellano i processi che caratterizzano il sistema rappresentato dalla rete di Petri

La ***Figura 3.4*** presenta il metamodello della rete di Petri che abbiamo utilizzato nel nostro lavoro. Tale metamodello differisce da quello adottato in [18] in quanto le transazioni temporali saranno trattate senza specificare l’intervallo in cui l’evento si verifica.

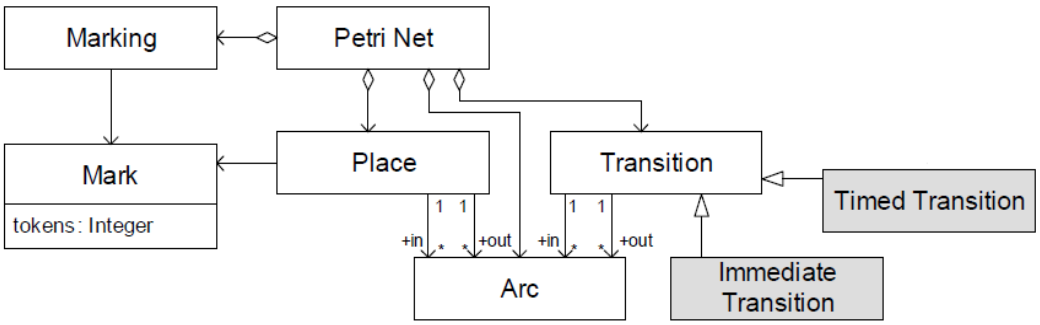

Figura 3.4: Metamodello della rete di Petri [18].

Nella nostra analisi, i posti generalmente rappresentano:

- Risorse, cioè macchine o impiegati; la presenza di token in tali posti indica la disponibilità della risorsa rappresentata.
- Stato di un'operazione; la presenza di token in tali posti indica che l'operazione è in corso.

Le transizioni generalmente rappresentano:

- Attività, quando si utilizzano transizioni temporizzate.
- Inizio o fine di attività, quando si utilizzano transizioni immediate.

3.4 TRASFORMAZIONE DEI MODELLI

Questa sezione descrive la trasformazione dei modelli generati da EDEN in modelli su cui condurre analisi più specifiche e misurabili. Questo passaggio è effettuato in maniera trasparente, ad esempio usando strumenti e tecniche descritti in [19] per trasformare i diagrammi di sequenza in reti di Petri, e internamente da EDEN. Quindi i professionisti aziendali che usano EDEN non devono preoccuparsi di possedere una conoscenza aggiuntiva su tecniche di analisi complesse o sui modelli. Un'ampia famiglia di questi modelli analizzabili può essere considerata per il tipo di informazioni e raccomandazioni immaginate da EDEN. In [2] sono stati adottati dei modelli

stocastici, come le “Extended Queueing Networks” [20] e le “Generalized and Stochastic Petri Nets” (GSPN) [21], poiché rappresentano l’incertezza che uno sviluppatore finisca un task con delle determinate tempistiche, usando le funzioni di distribuzione della probabilità che potrebbero rappresentare uno scenario DevOps in generale. In particolare con le GSPN, le decisioni d’instradamento sono stabilite in base alla probabilità associata ad ogni transizione. Nuovi task possono essere creati dopo la fase di sviluppo o operativa. Durante la fase di sviluppo, questi modelli possono essere aggiornati. I token possono essere impostati con il valore attuale della fase. In questo modo, l’analisi continua dei modelli lungo il processo software garantirà risultati migliori allineati con la realtà futura.

Per la nostra analisi considereremo i diagrammi di sequenza e le reti di Petri descritte precedentemente. Per fornire supporto per la progettazione e l’analisi, un approccio comune è quello di progettare in UML e poi trasformare il modello in una rappresentazione formale per l’analisi. La natura matematica delle reti di Petri crea una base solida per vari tipi di analisi. Murata [16] mette in evidenza un numero di metodi di analisi che si riferiscono ai problemi nel progettare un sistema aziendale. Tra questi ad esempio, l’analisi di raggiungibilità è usata per studiare le proprietà dinamiche di un sistema, cioè come un’azione può influenzare la probabilità che un evento accada in futuro, l’analisi di limitatezza è usata per controllare se i buffer e i registri del sistema memorizzano informazioni intermedie, mentre l’analisi di vivezza controlla che non accadano blocchi nel sistema. Tutte queste analisi e altre possono essere eseguite sulle reti di Petri e sono supportate da strumenti come PIPE [22], GreatSPN [23] e altri.

Il divario tra la fase di progettazione e di analisi può essere colmato usando la “Model Driven Development” [24], che fornisce tecniche per creare strumenti e infrastrutture software per la trasformazione automatica dei modelli. Il modello di trasformazione proposto, che supponiamo essere integrato in EDEN, affronta questo problema combinando i punti di forza tra due linguaggi: la specifica del comportamento del

sistema è formulato con i diagrammi di sequenza UML, mentre l'analisi è eseguita sulle reti di Petri. Il passaggio da UML a rete di Petri è realizzato attraverso un processo di trasformazione, il quale prende in input un diagramma di sequenza e automaticamente genera la rete di Petri equivalente. Il modello potrà poi essere successivamente analizzato con strumenti specifici che supportano le reti di Petri. La ***Figura 3.5*** fornisce una descrizione di alto livello di questo processo.

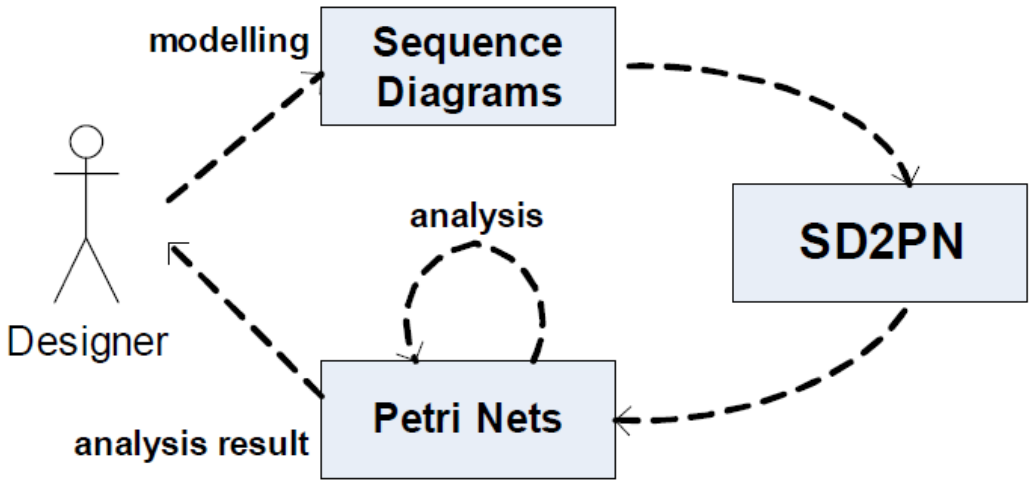

Figura 3.5: Rappresentazione del processo di trasformazione di un diagramma di sequenza in rete di Petri [18].

Per trasformare un diagramma di sequenza in rete di Petri abbiamo utilizzato le regole presenti in [25] e in [18], mostrate in ***Figura 3.6*** e ***Figura 3.7***.

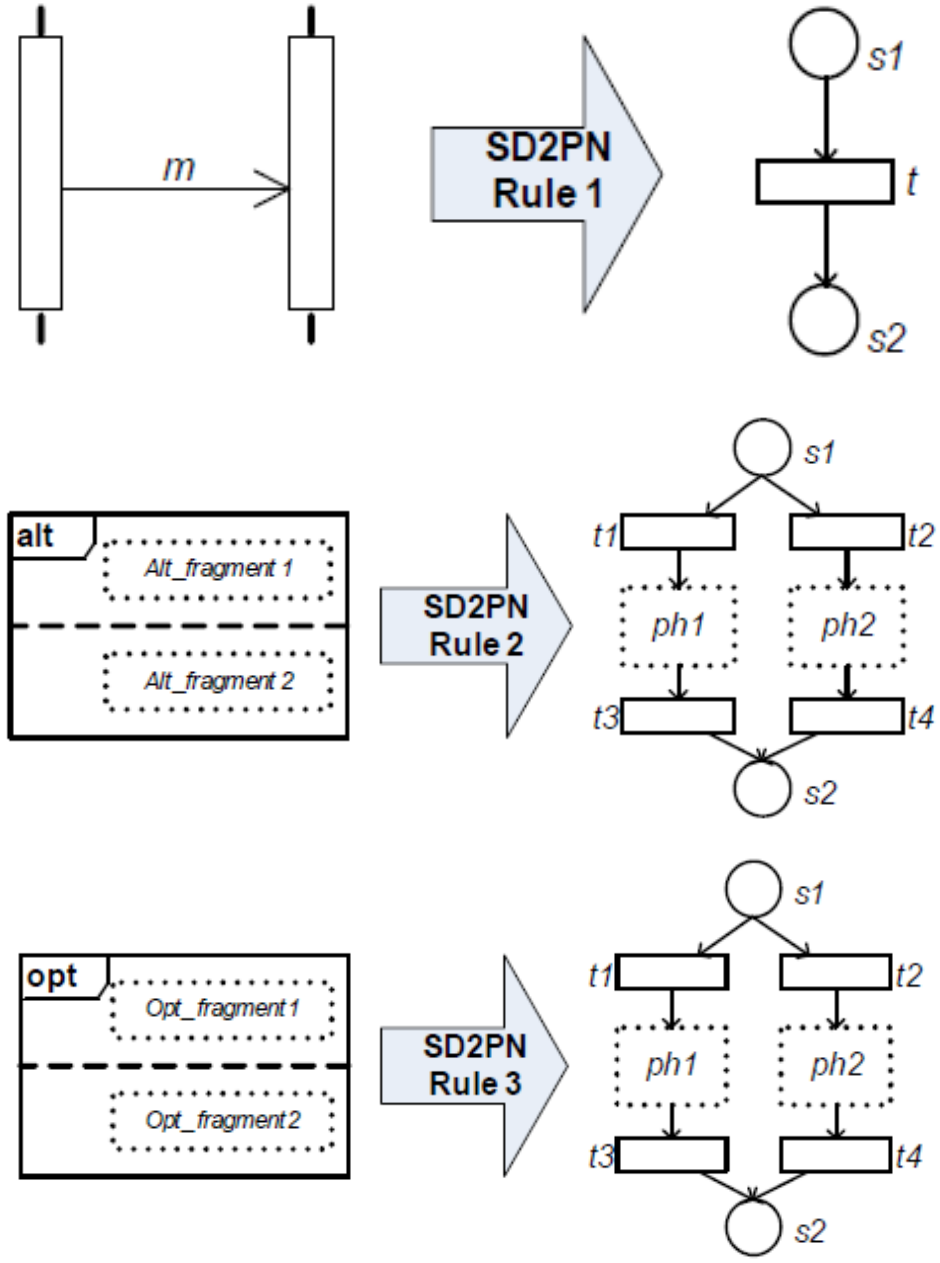

Figura 3.6: Regole 1,2 e 3 adottate nel processo di Trasformazione di un diagramma di sequenza in Rete di Petri [18], [25].

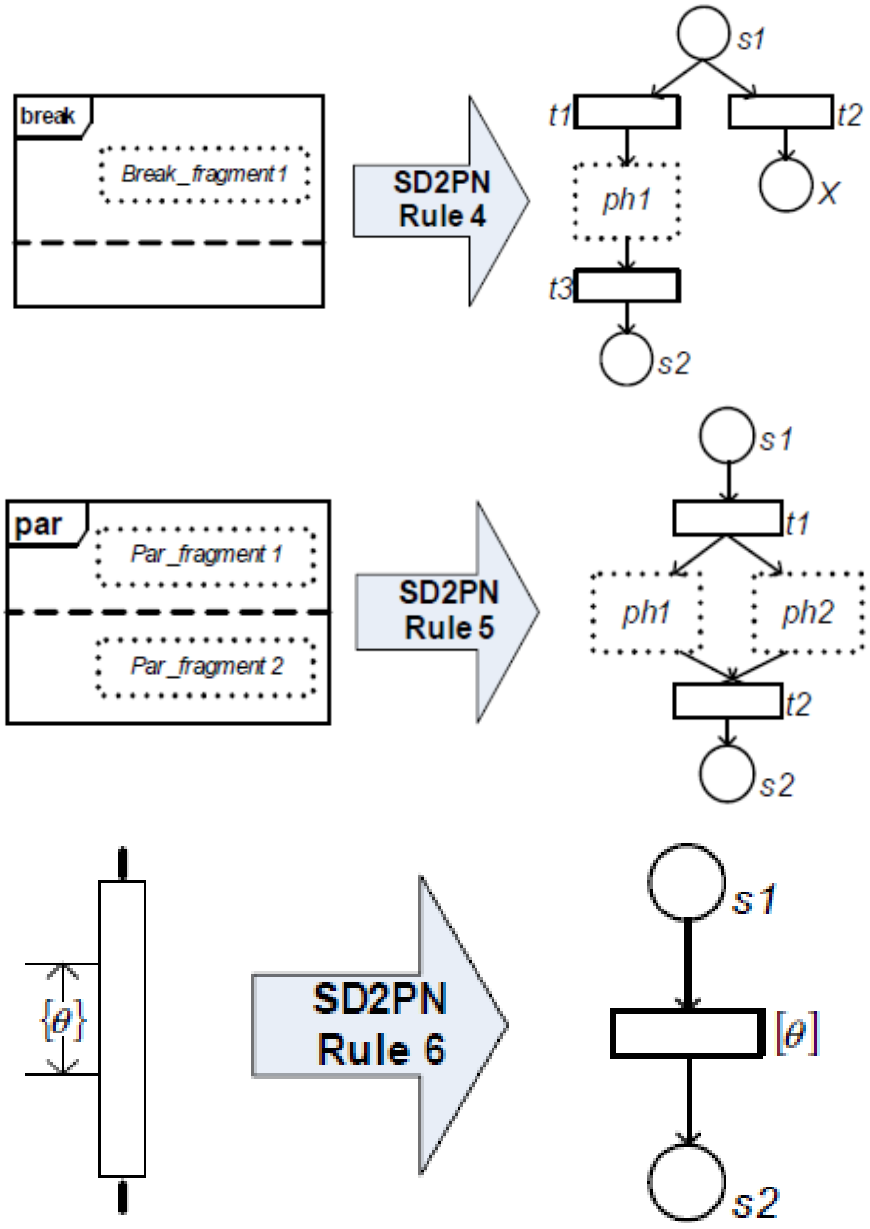

Figura 3.7: Regole 4,5 e 6 adottate nel processo di Trasformazione di un diagramma di sequenza in Rete di Petri [18], [25].

3.5 RIASSUNTO

In questo capitolo abbiamo parlato di EDEN [2], acronimo di “End-to-end Devops Engeneering”, cioè un approccio che riusa i processi di modellazione che un’azienda possiede, con l’obiettivo di pianificare e gestire gli sforzi sostenuti per adottare DevOps sin dall’inizio. Dopo aver introdotto EDEN nel paragrafo 3.1 ed esaminato le caratteriste nel paragrafo 3.2, abbiamo visto nel paragrafo 3.3. che con EDEN vi è la creazione di modelli interni, che rappresentano le informazioni sui processi software disponibili. Questi modelli sono poi analizzati, ad esempio per stimare le performance di un processo, quando ne è prevista la conclusione o per riallocare o modificare la priorità di un task. Nello specifico abbiamo analizzato i diagrammi di sequenza UML, nel paragrafo 3.3.1, e le reti di Petri, nel paragrafo 3.3.2.

Infine il paragrafo 3.4 descrive la trasformazione dei modelli generati da EDEN in modelli su cui condurre analisi più specifiche e misurabili. In particolare abbiamo concentrato la nostra analisi su strumenti e tecniche per trasformare i diagrammi di sequenza in reti di Petri.

Nel prossimo capitolo utilizzeremo i diagrammi di sequenza e le reti di Petri sfruttando le regole di trasformazione rappresentate in ***Figura 3.6*** e in ***Figura 3.7***. Esamineremo un caso di studio reale tratto dal libro “The Phoenix Project” [3], che racconta la storia di “Parts Unlimited”, un’azienda americana che, nonostante i problemi iniziali, sposando una metodologia DevOps gli ha sbarrato la strada verso il successo.

4 UN CASO DI STUDIO DEVOPS: "THE PHOENIX PROJECT"

Il capitolo presenta un caso di studio reale DevOps tratto dal libro "The Phoenix Project" [3]. Il paragrafo ***4.1*** riassume l'opera con particolare enfasi sulla trasformazione subita dai processi dell'azienda adottando una cultura DevOps. Gli scenari esaminati saranno modellati tramite diagrammi di sequenza nel paragrafo ***4.2*** e trasformati poi nel paragrafo ***4.3*** in reti di Petri, seguendo le regole di trasformazione discusse nel paragrafo ***3.4***.

4.1 DESCRIZIONE DELL'OPERA

L'opera racconta la storia di Bill Palmer, neo promosso a vice presidente dell'IT Operations dell'azienda "Parts Unlimited". L'azienda sta sviluppando un nuovo progetto, chiamato "Phoenix", fondamentale per il futuro dell'azienda, purtroppo fuori budget e in ritardo sulla tabella di marcia. Il CEO incarica Bill di risolvere questi problemi in novanta giorni, altrimenti l'intero dipartimento IT verrà esternalizzato. Superate le problematiche iniziali, Bill incomincia a notare come il lavoro IT ha molte cose in comune con quello svolto da un'azienda manifatturiera. Questo grazie all'aiuto di un membro del consiglio aziendale e la sua filosofia misteriosa delle "three ways", ovvero:

1) Creare flussi di lavoro veloci tra i team operativi e di sviluppo, in modo da reagire immediatamente ai cambiamenti del mercato e alle esigenze dei clienti.
2) Creare scambi di feedback veloci e costanti tra i vari team.
3) Creare una cultura che gestisca i rischi e impari dai fallimenti.

Quindi, non avendo molto tempo a disposizione, egli deve organizzare i flussi di lavoro, ottimizzare le comunicazioni interdipartimentali ed efficacemente servire le altre funzioni aziendali.

Di seguito riassumiamo il contenuto del libro, riportando gli avvenimenti verificatisi e ogni tipo di inefficienza riscontrata nell'azienda, con l'evoluzione delle soluzioni proposte dal protagonista dell'opera.

- Il progetto Phoenix è in fase di sviluppo. Interruzioni e guasti nel sistema IT sono all'ordine del giorno a causa della scarsa comunicazione tra i vari reparti. Qualsiasi cambiamento viene effettuato senza rispettare processi e procedure. Inoltre non esiste nessun processo in grado di tenere traccia di qualsiasi attività effettuata.

Soluzione: Quando si verifica un incidente, si indaga sui dipendenti che possono aver causato l'interruzione, inserendo in una lista ordinata tutti i cambiamenti o modifiche degli ultimi giorni, riportando causa ed effetto di ognuno.

- Dovendo rispettare una data di scadenza, un'implementazione frettolosa da parte dei membri del team di Sviluppo comporta la mancanza di Test, quindi i problemi sorgeranno in produzione, dove il team operativo svolgerà lavoro aggiuntivo.

Soluzione: Vengono creati due team, chiamati "Dev" e "Ops", formati dagli omonimi reparti, che devono lavorare a stretto contatto in modo da accelerare i tempi di sviluppo del progetto.

- Produzione gestita senza conoscere la domanda, priorità, stato del lavoro nel processo e le risorse disponibili.

Soluzione: Tramite indagini aggiuntive, viene generata una lista ottenuta esaminando tutte le risorse chiave, per conoscere su cosa stanno lavorando e da quanto tempo, con un occhio di riguardo sui progetti principali che sta svolgendo l'azienda.

- Su 150 dipendenti IT, è impegnata più di una persona per progetto. E se un progetto viene messo in pausa, le risorse chiave rimarranno comunque legate. In più bisogna considerare gli incidenti, che potrebbero essere evitati con un po' di manutenzione preventiva. Inoltre, i dipendenti IT non sono mai produttivi al 100% perché altri reparti, come business e marketing, chiedono sempre di eseguire lavoro per loro.

Soluzione: Si vuole cercare di creare un processo che riesca a prevenire incidenti. Per fare ciò, si coinvolgono tutti i membri del team Operativo per creare una lista di cambiamenti, autorizzati e schedulati, da apportare nei prossimi 30 giorni. Ogni cambiamento è classificato nel modo seguente, in base alla priorità e soprattutto al tipo di rischio:

- Alto: devono essere autorizzati prima di essere implementati e schedulati.
- Medio: bisogna consultare e farli approvare dalle persone potenzialmente coinvolte.
- Basso: sono cambiamenti standard, non necessitano di approvazione.

Per cambiamento si intende qualsiasi attività che è fisica, logica o virtuale per applicazioni, database, sistemi operativi, reti o hardware che potrebbe avere un impatto sui servizi disponibili.

- La domanda per il lavoro IT è superiore alla loro abilità di consegna. I progetti portati avanti dall'azienda, incidenti e fallimenti giornalieri richiedono risorse chiave, che non riescono a gestire in modo efficiente nonostante l'azienda sostenga costi maggiori proprio nell'IT rispetto ai competitors. Inoltre quando avviene un'interruzione non si riesce mai a capire la causa dell'incidente in tempi brevi.

Soluzione: L'azienda si dovrà abituare a documentare ogni attività svolta, in modo da avere il totale controllo su ogni processo e, se si verifica un problema, sapere subito cosa è successo e dove intervenire. Per fare ciò, è necessario creare una cultura per avere più controllo sui dipendenti, in modo che dichiarino ogni operazione svolta.

- La risorsa principale dell'azienda, il capo ingegnere, gestisce troppi lavori. Ogni volta che sorge un problema viene sempre chiamato in causa.

Soluzione: Per proteggerlo, viene creata un "resource pool of LEVEL 3", composta da tre dipendenti che si occuperanno di gestire i problemi. Solo loro possono comunicare con il capo ingegnere, posto al "LEVEL 3S", salvo previa approvazione da parte dei superiori. Inoltre la risorsa chiave dovrà risolvere un problema solo una volta. Cioè, se lo stesso problema accadrà un'altra volta, qualcun altro deve essere in grado di risolverlo. Quindi è importante documentare tutte le procedure che sono state adottate.

- Primo Deployment di Phoenix. C'è molta confusione tra i reparti e anche il problema più banale rallenta il lavoro a causa della scarsa comunicazione. Quando gli Sviluppatori risolvono un problema, se ne presentano subito degli altri. Il risultato è un totale fallimento, poiché è stato rilasciato un prodotto software poco affidabile.

Soluzione: Ogni release che è stata fatta e si farà dovrà essere schedulata e documentata. Ai negozi dovranno essere inviate istruzioni ben precise su come gestire i malfunzionamenti del sistema. In questa situazione di emergenza: alcuni stanno monitorando la fragilità dei sistemi e servizi, altri stanno gestendo le telefonate dei manager degli store, altri stanno aiutando nei test, altri gli Sviluppatori nel riprodurre i problemi. In pratica si sta applicando il concetto di "collaborazione" della metodologia DevOps.

- I cambiamenti schedulati in precedenza non sono stati eseguiti a causa di Phoenix, che ha impiegato tutte le risorse, generando quello che nel testo è chiamato "Unplanned Work", in altre parole lavoro non pianificato, che avviene quando sorgono dei problemi.

Soluzione: Si cerca di creare un processo che renda visibile il "work in process", definito nel testo come il "killer silenzioso" perché, se non si controlla, è la causa principale di ritardi e qualità. Si seguono i seguenti step:

1) Identificare il vincolo, nel testo chiamato “constraint”, ovvero la “bottlenck”, ed essere sicuro che sia quello, altrimenti qualsiasi miglioramento fatto sarà solo un’illusione.

2) Sfruttare il vincolo, non consentirgli di sprecare il tempo e farlo lavorare sulle alte priorità dell’azienda.

3) Questo step è subordinato al vincolo, poiché è lui che detta i tempi di lavoro.

Quindi bisogna mappare i ritmi di lavoro dell’IT Operations con il vincolo. Inoltre, essere abili a lasciare fuori dal sistema lavoro inutile è più importante che aggiungerne.

- Concentrando tutto il tempo per fare lavoro di “recovery”, si riduce il tempo necessario per la pianificazione, che non è abbastanza per fare le dovute considerazioni e decidere se si può accettare o no un lavoro. In questo modo più progetti vengono messi nel sistema, si creano scorciatoie per cercare di eseguire quante più cose possibili, quindi si mette benzina sul fuoco.

Soluzione: I team di sviluppo e operativi non devono accettare nessun nuovo progetto per due settimane, e tutto il lavoro nell’ambiente di produzione verrà congelato, eccetto quello connesso a Phoenix. Nessun flusso di lavoro circolerà tra il dipartimento di sviluppo e operativo in modo da poter identificare le aree di “technical debt”, che il dipartimento di sviluppo userà per diminuire il lavoro non pianificato creato, o che si potrebbe creare, da problematiche applicazioni in produzione.

- Il capo ingegnere del software, che rappresenta il vincolo o bottleneck dell’azienda, supporta più centri di lavoro ma, dato che può essere ogni volta in uno solo di essi, il lavoro non è mai eseguito per tempo.

Soluzione: Bisogna standardizzare il suo lavoro, così che anche altre persone possono eseguirlo, e i centri di lavoro dove è richiesto diminuiranno. Tra i progetti messi in pausa, riprendono quelli che non richiedono il suo intervento.

Inoltre parte il “Monitoring Project”, che serve a controllare ogni attività, al fine di prevenire fallimenti e intervenire in maniera tempestiva qualora si verificasse un incidente. Si sta applicando il principio del “controllo continuo” di DevOps discusso nel paragrafo ***2.4.1***.

- Quello che ha fatto finora l’azienda è accumulare, e svolgere, una montagna di lavoro e, quando sorge un problema, non si sa dove risolverlo. In aggiunta, le attività svolte dal team di che gestisce la sicurezza rovina il lavoro schedulato dagli altri. Il suo obiettivo non è quello di mettere lavoro senza utilità nel sistema IT, ma di toglierlo.

Soluzione: Si sperimenta una “Kamban board” intorno al vincolo in modo da rendere lo stato del lavoro in corso, ovvero il “work in process” discusso precedentemente, visibile a tutti. Viene fatta un’analisi accurata delle risorse chiave in modo che, a fronte di una richiesta, si potranno conoscere le risorse che non sono disponibili e i tempi di attesa senza interromper nessun lavoro.
Documentando tutti i task ricorrenti, una volta standardizzati, riusciranno a gestire e migliorare il flusso di lavoro sino ad ottenere una configurazione di produzione uniforme.
Per favorire scambi di lavoro veloci tra i centri di lavoro, viene creato un nuovo ruolo, che eseguirà il “minute-by-minute control”. Se necessario, questa persona dovrà aspettare al “work center” fino a quando l’attività non sarà stata completata, e poi consegnare l’output al centro successivo.

- La concorrenza ogni giorno sottrae quote di mercato all’azienda e stanno perdendo clienti perché Phoenix non ha soddisfatto le attese.

Soluzione: Bisogna garantire che l’IT conosca ciò che deve sostenere e proteggere e, scoprire dove il business confida nell’IT per raggiungere i suoi obiettivi. In questa fase vengono intervistati i manager responsabili dei “business goal”, cercando di capire il valore di ogni step che serve a raggiungere ogni goal, inclusi quelli che non sono

visibili, come nell'IT. Mostrando come i rischi IT mettono in pericolo le "business performance measures", si possono prendere decisioni di business migliori. Emerge il concetto di "pianificazione continua" di DevOps descritto nel capitolo precedente.

- Fragilità delle applicazioni e infrastrutture che supportano i vari reparti dell'azienda, che richiedono di essere sostituite. I dati che acquisisce il sistema spesso non sono attendibili.

Soluzione: Vengono individuati alcuni punti che ridurranno notevolmente il lavoro svolto dalla Sicurezza:

- fare in modo che le vulnerabilità che si presentano in produzione non avvengano più modificando il processo di deploy;
- cercare di evitare cambiamenti che possano mettere in pericolo l'audit, ovvero una valutazione per stabilire in quale misura gli obiettivi prefissati siano stati soddisfatti o meno. e creare la documentazione che gli auditor richiedono "in corso";
- esternalizzare i dati emessi da applicazioni secondarie inutili ai fini del business aziendale.

Realizzando i punti precedenti, si ripagheranno i "Technical Debt" di Phoenix.
Si vuole quindi creare quella che nel testo viene chiamata "Deployment Pipeline", analoga alla "Delivery Pipeline" descritta nel capitolo precedente, che ha l'obiettivo di costruire e avere sotto controllo un processo che abbia un ambiente comune, facilitando in questo modo il lavoro dei team di sviluppo, operativi e controllo qualità. Attraverso "build procedure" comuni, gli sviluppatori scriverebbero codice in un ambiente che assomiglia a quello di produzione. Per fare ciò, vengono schierati i vincoli di Dev e Ops al fronte, in quanto saranno loro a disegnare la linea di produzione in modo che non venga generato "unplanned work".

- Con l'Hardware in loro possesso, il tutto viene eseguito troppo lentamente.

Soluzione: Dopo un'accurata analisi su rischi e conformità, si investe nel CLOUD, ottenendo risultati eccezionali.
Con l'obiettivo di guadagnare "agilità" e sfruttare i principi DevOps finora appresi grazie ai quali si può reagire immediatamente alle esigenze del mercato e dei clienti, l'azienda investe con successo in un progetto chiamato "Unicorn", che ha lo scopo di correggere i difetti di Phoenix. Orami, il flusso del planned work è più veloce che mai. I team di sviluppo e operativi lavorano insieme per rendere il codice e l'infrastruttura più resistente ai fallimenti, rendendo i servizi IT più robusti.
Si lavora anche ad un progetto che sfrutta falle di sicurezza per stressare il sistema con ogni sorta di attacco malvagio, con lo scopo di rafforzare la cultura di gestire i rischi e imparare dai fallimenti.
Insomma, è stata sviluppata una metodologia DevOps, in cui ogni reparto lavoro l'uno con il supporto dell'altro.

4.1.1 EDEN IN AZIONE

Prendendo come riferimento gli scenari descritti nel paragrafo precedente, abbiamo simulato il comportamento di EDEN [2]. Abbiamo supposto che l'azienda utilizzi EDEN, e che negli scenari descritti, siano stati inseriti e acquisiti i dati nelle determinate circostanze tramite gli strumenti integrati, i quali sono elaborati da EDEN generando internamente dei modelli UML. Tra questi abbiamo scelto il diagramma di sequenza, poiché modella perfettamente le informazioni presenti nel caso di studio, che esprimono l'evoluzione dei processi adottati e il flusso di lavoro interno all'azienda.

UML è privo di una semantica formale e quindi non è possibile applicare direttamente tecniche matematiche sui modelli per la validazione del sistema. Per questo motivo, abbiamo utilizzato i diagrammi di sequenza generati come input per strumenti integrati in EDEN che li trasformano in maniera automatica in reti di Petri, che sono uno dei più affermati formalismi per modellare comportamenti complessi, ampiamente utilizzate per l'analisi e la sintesi di sistemi. In questo modo possono essere condotte

analisi più accurate, come la gestione della concorrenza e la valutazione delle performance del sistema, ad esempio integrando EDEN con strumenti come [22] e [23], che acquisiscono le reti di Petri generate e permettono di esaminarle. Abbiamo simulato questo processo nel paragrafo ***4.3***, considerando le regole di trasformazione tra i due diagrammi descritte nel capitolo precedente nel paragrafo ***3.4***.

4.2 FASE DI ELABORAZIONE

In questo paragrafo mostreremo i diagrammi di sequenza, dedotti dagli scenari precedenti, che abbiamo utilizzato per modellare i processi aziendali che hanno subito uno spostamento verso DevOps.

Abbiamo sfruttando anche le notazioni grafiche del “Boundary-Control-Entity” pattern [26], in quanto si adatta perfettamente con gli scenari descritti.

I processi che modelleremo coinvolgeranno i seguenti attori:

- DIRECTION: formata dalla direzione aziendale.
- DEPARTMENT: si riferisce ai responsabili di ogni dipartimento. Nel contesto viene trattata come una “Boundary”, in quanto comunica con la direzione, riceve le direttive da essa, le elabora e le assegna ai dipendenti.
- TEAM MEMBER: identifica un dipendente.
- TASK: identifica l’attività da svolgere.
- SYSTEM: il “Control”, identifica il sistema aziendale, sempre in funzione e tramite il quale si monitora ogni attività svolta. È colui che controlla i flussi esecuzione negli scenari.
- LOG: l’ “Entity”, dove ogni informazione riguardante l’avanzamento e lo stato del lavoro di ogni processo viene salvata.

4.2.1 INIZIALIZZAZIONE

Questo scenario ha inizio con l’arrivo di un nuovo progetto, che può essere un progetto di business oppure interno. Il primo passo lo compiono i manager aziendali, i quali

devono valutare se l'organizzazione ne trae beneficio nel svilupparlo. Per questo motivo, all'inizio viene svolta un'analisi più accurata, mostrata in ***Figura 4.4*** nel frame "ONGOING PROJECTS", esaminando i progetti a cui l'azienda sta lavorando.

Con riferimento al frame "RESOURCES MAPPING" della ***Figura 4.1***, nello specifico bisogna conoscere, in ogni dipartimento, a quali task ogni dipendente sta lavorando, da quanto tempo e quelli per lui schedulati. Tali caratteristiche sono espresse tramite la condizione di guardia all'interno del frame "TASK ASSOCIATION". I risultati sono ordinati in una lista accurata per poi essere consegnati ed elaborati dalla direzione aziendale.

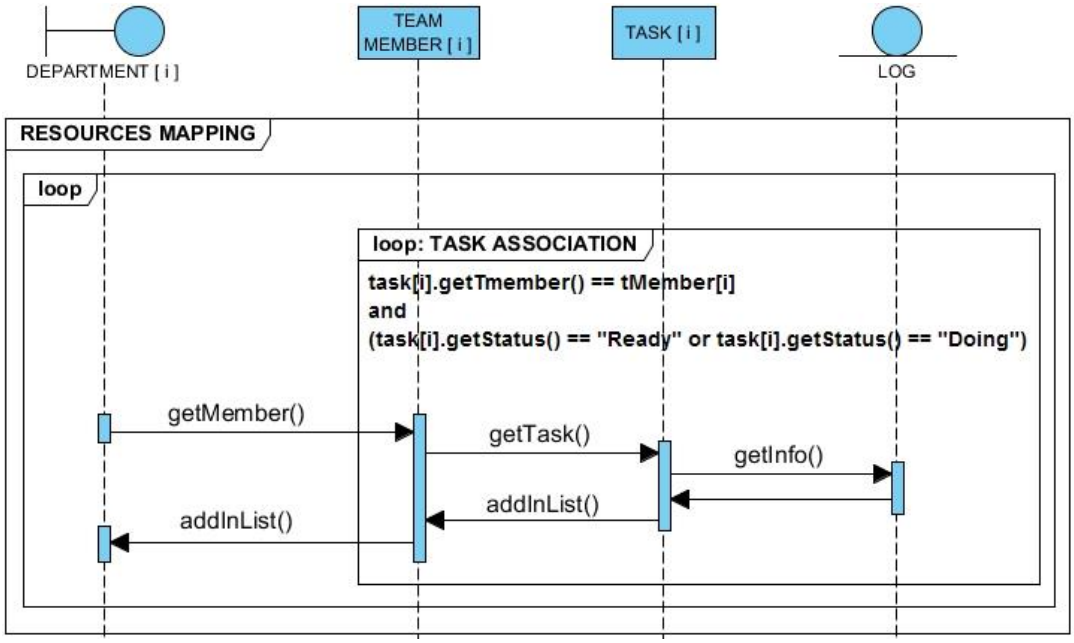

Figura 4.1: Diagramma di sequenza che rappresenta il recupero delle risorse aziendali.

Come mostrato in ***Figura 4.4***, se dopo le dovute analisi l'azienda decide di sposare il progetto, il flusso di esecuzione si sposta nel frame "PROJECT INITIALIZATION". La direzione invia le "skills" del progetto ai vari dipartimenti, che a loro volta controlleranno il lavoro dei propri dipendenti, identificheranno la risorsa migliore per sviluppare il nuovo progetto e pianificheranno la linea di produzione in base alle "bottlenecks". Questo passaggio è fondamentale in quanto bisogna evitare carichi di

lavoro eccessivi per determinate risorse. Infatti ciò potrebbe compromettere lo sviluppo del nuovo progetto e di quelli in corso d’opera. Dopo aver richiesto l’approvazione della Direzione, vengono creati tutti i task necessari per sviluppare il progetto, come mostrato in ***Figura 4.2*** nel frame “CREATE TASK”.

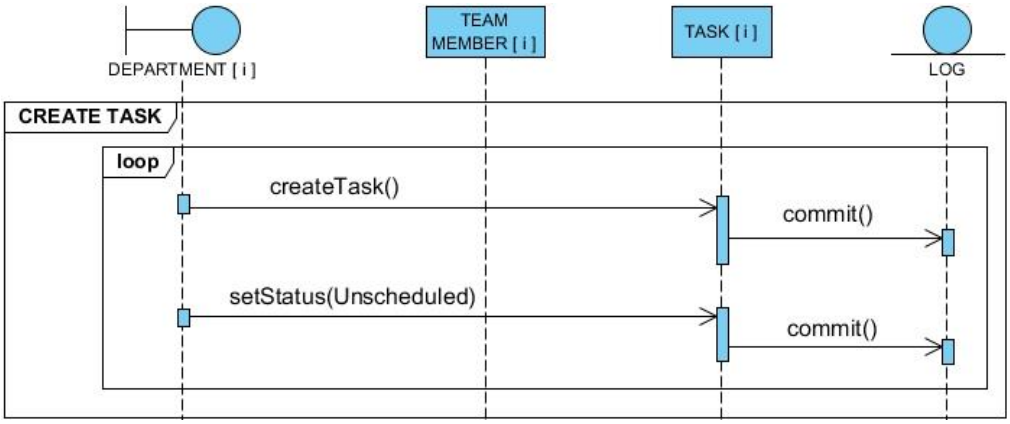

Figura 4.2: Diagramma di sequenza che rappresenta la creazione dei task.

Dopo che i task sono stati creati, bisogna associarli ad uno o più dipendenti. La ***Figura 4.3*** rappresenta il comportamento di questo scenario nel frame “TASK ASSIGNMENT”, dove il dipartimento recupera le informazioni del task, controlla le competenze che richiede e il lavoro svolto da tutti i dipendenti.

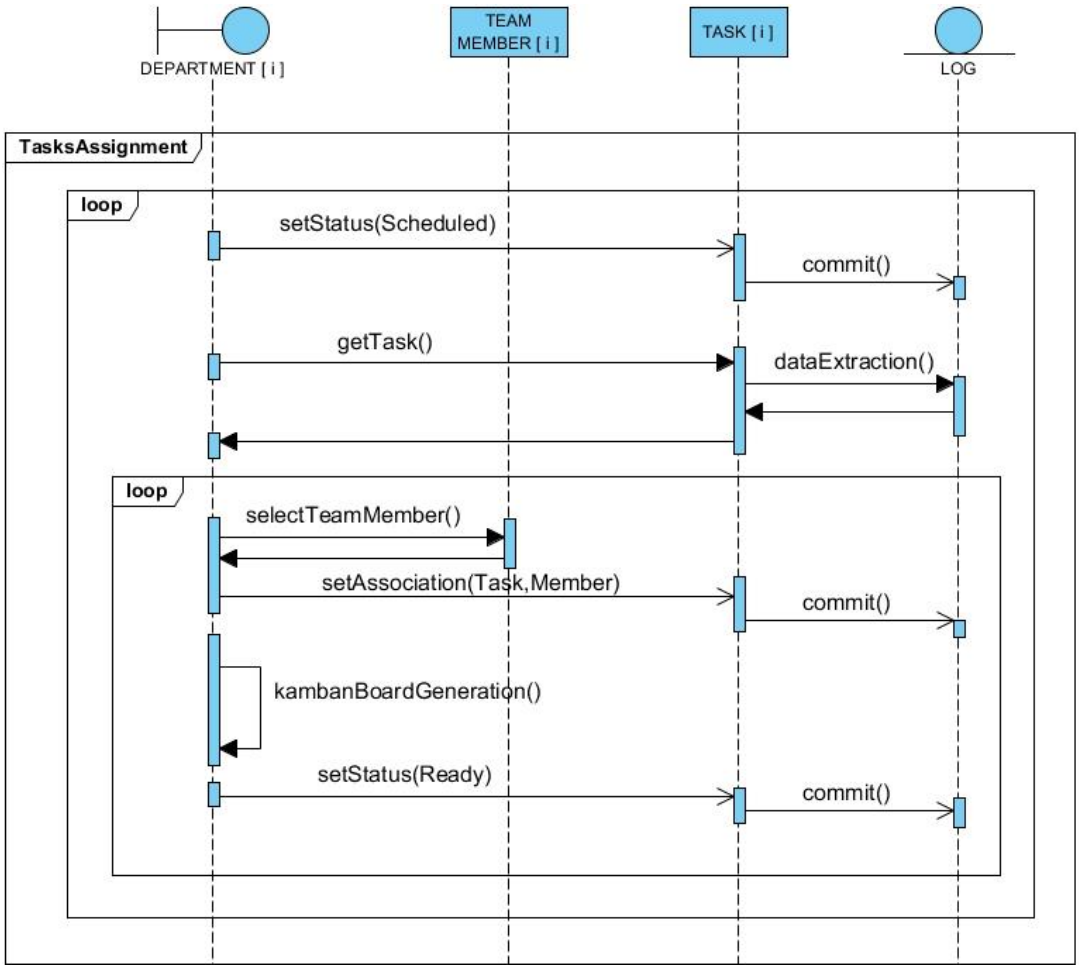

Figura 4.3: Diagramma di sequenza che rappresenta l'assegnamento dei task.

Alla fine di questo processo vi è poi l'associazione. Ad ogni dipendente è associato un task con relativa "kanban board", utile per rendere visibile il "work in process".

La ***Figura 4.4*** mostra lo scenario appena discusso al completo.

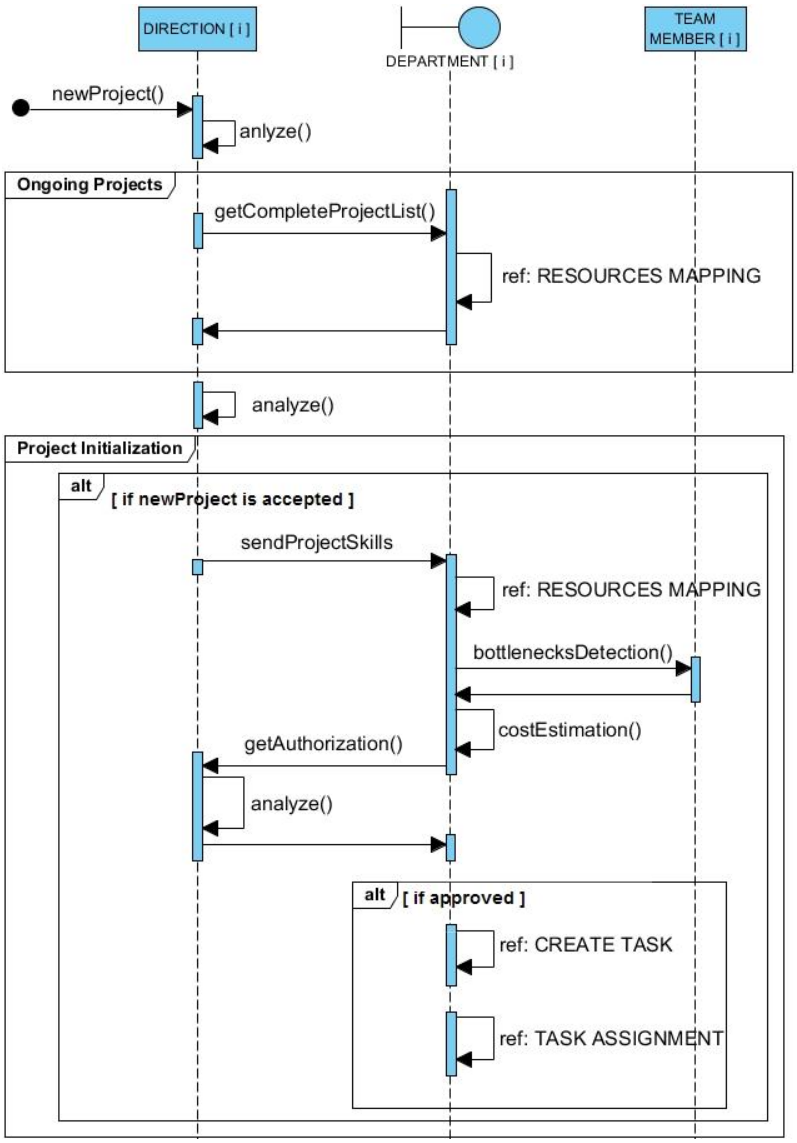

Figura 4.4: Diagramma di sequenza che rappresenta l’intero scenario esposto.

4.2.2 ESECUZIONE

Prima di entrare nel cuore dell’esecuzione, descriviamo i due frame “WARNING” e “NEW TASK”, mostrati in ***Figura 4.5***, che possono venire in aiuto durante l’esecuzione del processo.

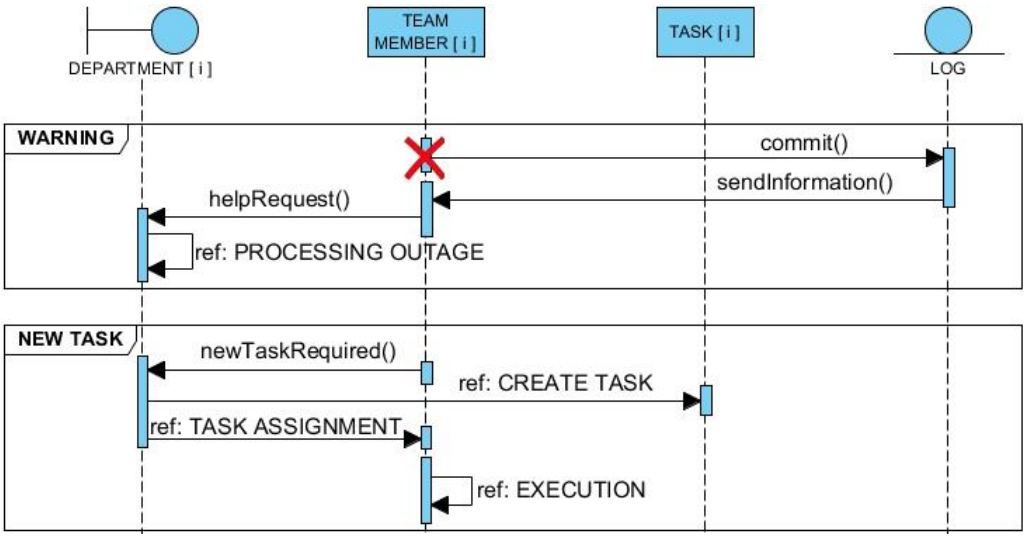

Figura 4.5: Diagramma di sequenza che rappresenta le procedure che gestiscono il tipo di interruzione.

- WARNING, che reagisce immediatamente a fronte di un'interruzione, passando il controllo alle procedure di "PROCESSING OUTAGE", descritte in seguito e mostrate nella ***Figura 4.11***;
- NEW TASK, che entra in gioco quando è richiesto un nuovo task, richiamando i frame "CREATE TASK" e "TASK ASSIGNMENT" descritti e rappresentati in precedenza rispettivamente nella ***Figura 4.2*** e nella ***Figura 4.3***, e "EXECUTION", mostrato in ***Figura 4.6.***

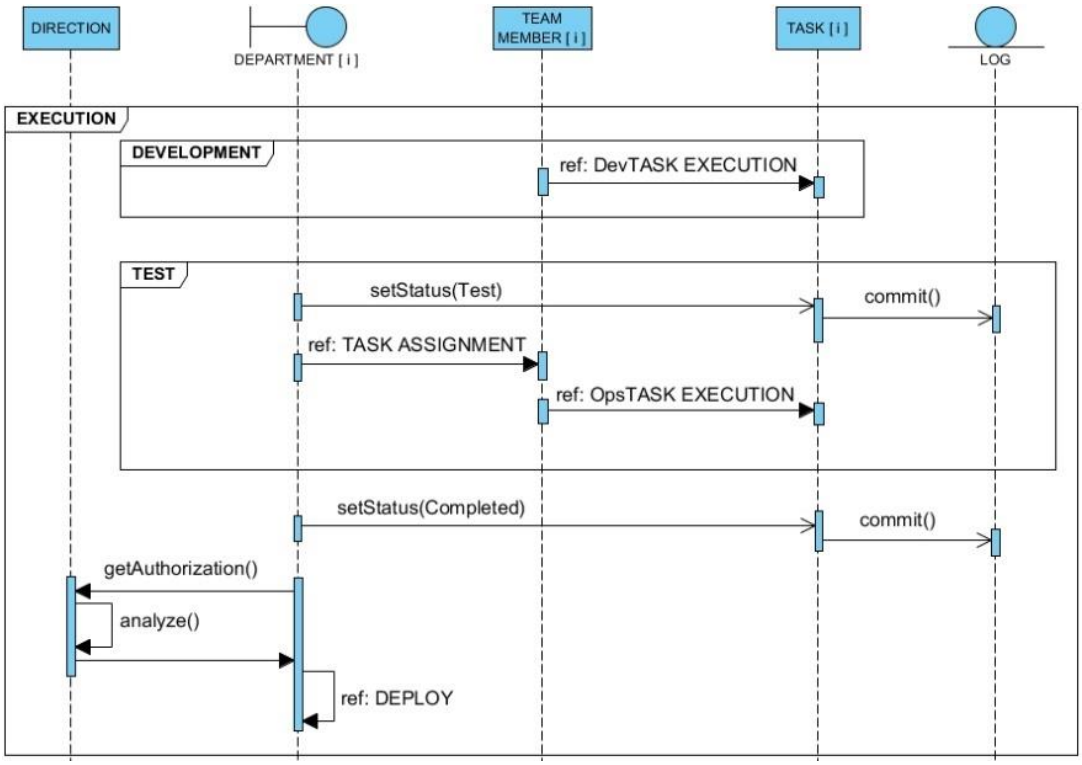

Figura 4.6: Diagramma di sequenza che rappresenta il processo di esecuzione di un’attività.

Il frame principale “EXECUTION”, coinvolge nel processo due attività fondamentali, espresse dai frame:

- DEVELOPMENT: al suo interno contiene il riferimento al frame “DevTASK EXECUTION”, mostrato in ***Figura 4.7***, che coinvolge nel processo i task contrassegnati come “Ready”. Cambiando il “flag” che identifica lo status del task in “Doing”, comincia la fase di sviluppo, condotta dai membri dell’omonimo reparto. Questa fase, insieme alla fase di pre-test, è caratterizzata da un continuo feedback sullo status di avanzamento del lavoro, in modo da monitorare costantemente ogni attività, al fine di prevenire fallimenti e intervenire in maniera tempestiva qualora si verificasse un problema.

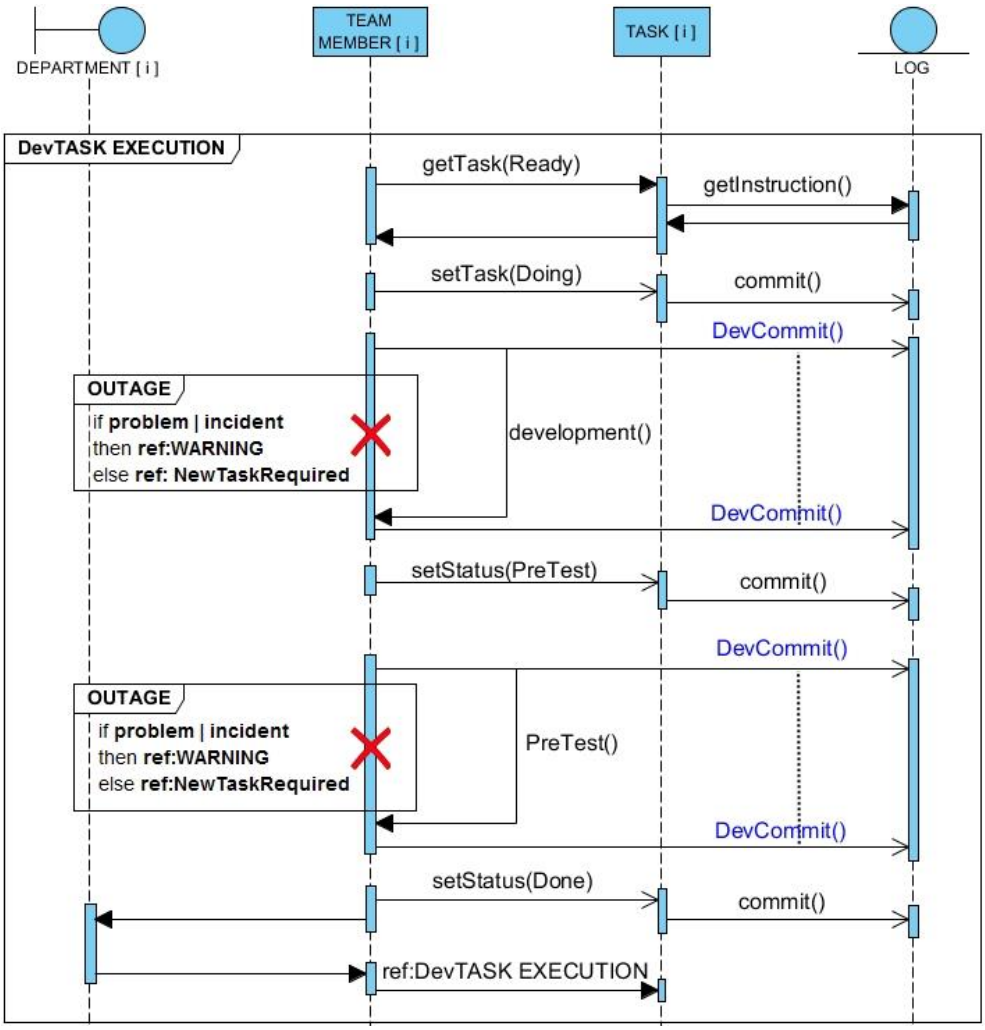

Figura 4.7: Diagramma di sequenza che rappresenta le operazioni svolte dai membri del team di sviluppo.

- TEST: al suo interno contiene il riferimento al frame "OpsTASK EXECUTION", mostrato in ***Figura 4.8***. Terminata la fase di sviluppo, inizia la fase di test, condotta dal reparto operativo. Dopo aver assegnato i task a ciascun dipendente, parte il processo di test caratterizzato da un continuo salvataggio di informazioni che descrivono l'avanzamento del processo. Se non si verifica nessun problema, l'esecuzione di ogni task si conclude settando il suo "flag" in "completed".

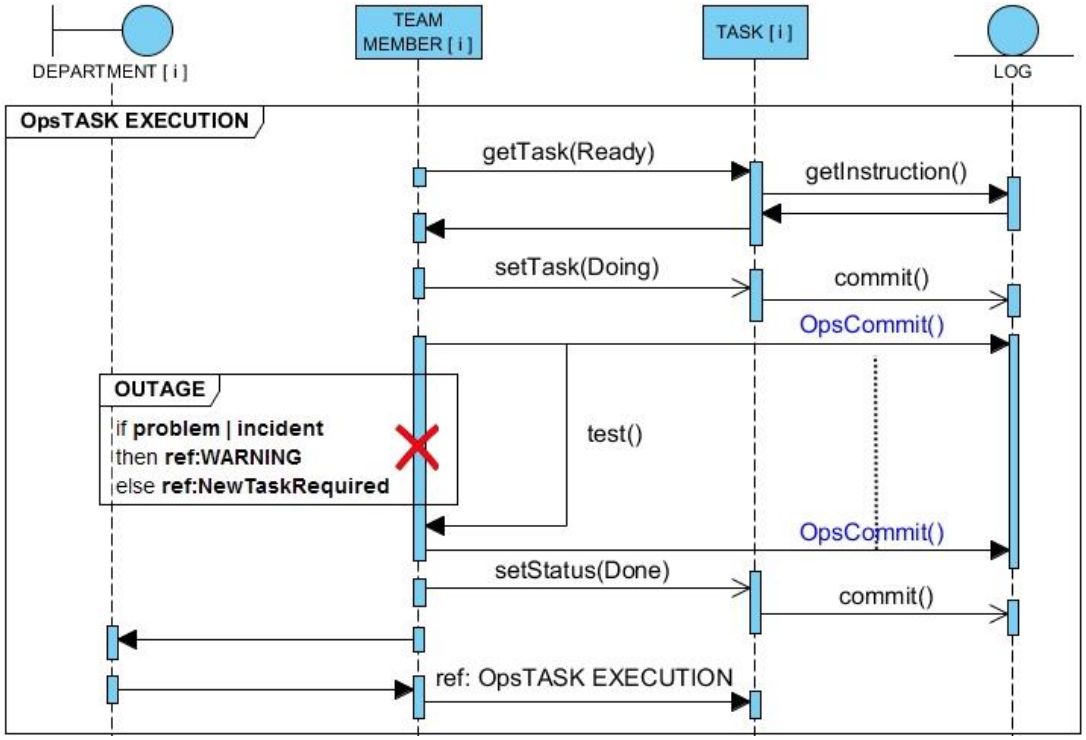

Figura 4.8: Diagramma di sequenza che rappresenta le operazioni svolte dai membri del team operativo.

Quando si concludono le fasi di sviluppo e test, viene richiesta l’autorizzazione alla Direzione per eseguire il rilascio del software nel sistema, discusso nel prossimo paragrafo.

4.2.3 RILASCIO

Prima di descrivere il nuovo scenario, bisogna introdurre il frame “SYSTEM WARNING”, mostrato ***Figura 4.9***, che descrive la sequenza di operazioni da attuare in caso venga segnalato un problema dal sistema. Nello specifico, il dipendente che riscontra il problema lo segnala al dipartimento di competenza, che tramite il frame “PROCESSING OUTAGE”, descritto nel prossimo paragrafo e rappresentato in ***Figura 4.11***, avvia le procedure di ripristino del sistema.

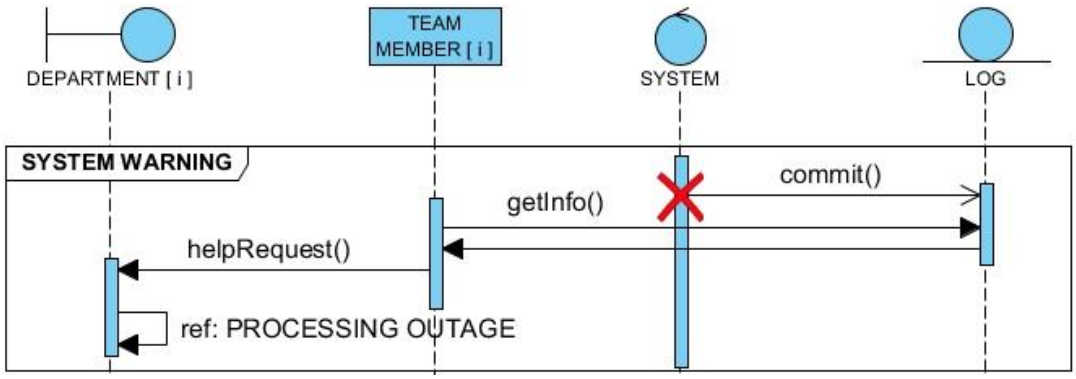

Figura 4.9: Diagramma di sequenza che rappresenta le procedure che gestiscono un'interruzione del sistema.

Il sistema viene monitorato costantemente al fine di prevenire fallimenti e intervenire in maniera tempestiva qualora si verificasse un incidente. All'interno del frame "CONTINUOUS MONITORING" mostrato in ***Figura 4.10***, sono presenti i seguenti frammenti combinati:

- DEPLOY: indica il processo di rilascio del software, che avviene tramite stretta collaborazione tra i dipartimenti di Sviluppo, Operativo e Controllo Qualità tramite la formazione di un team DevOps. Anche durante questa fase ogni attività svolta viene documentata al fine di prevenire incidenti. Finito il Deploy, si passa alla fase di test, rappresentata in precedenza nel frame "TEST" in ***Figura 4.6***.
- CONTINUOUS STRESSING: per rendere l'infrastruttura più resistente ai fallimenti, rendendo i servizi IT più robusti, il dipartimento di Controllo Qualità sfrutta falle di sicurezza per stressare il sistema con ogni sorta di attacco malvagio, con lo scopo di rafforzare la cultura tipica di DevOps di gestire i rischi e imparare dai fallimenti.

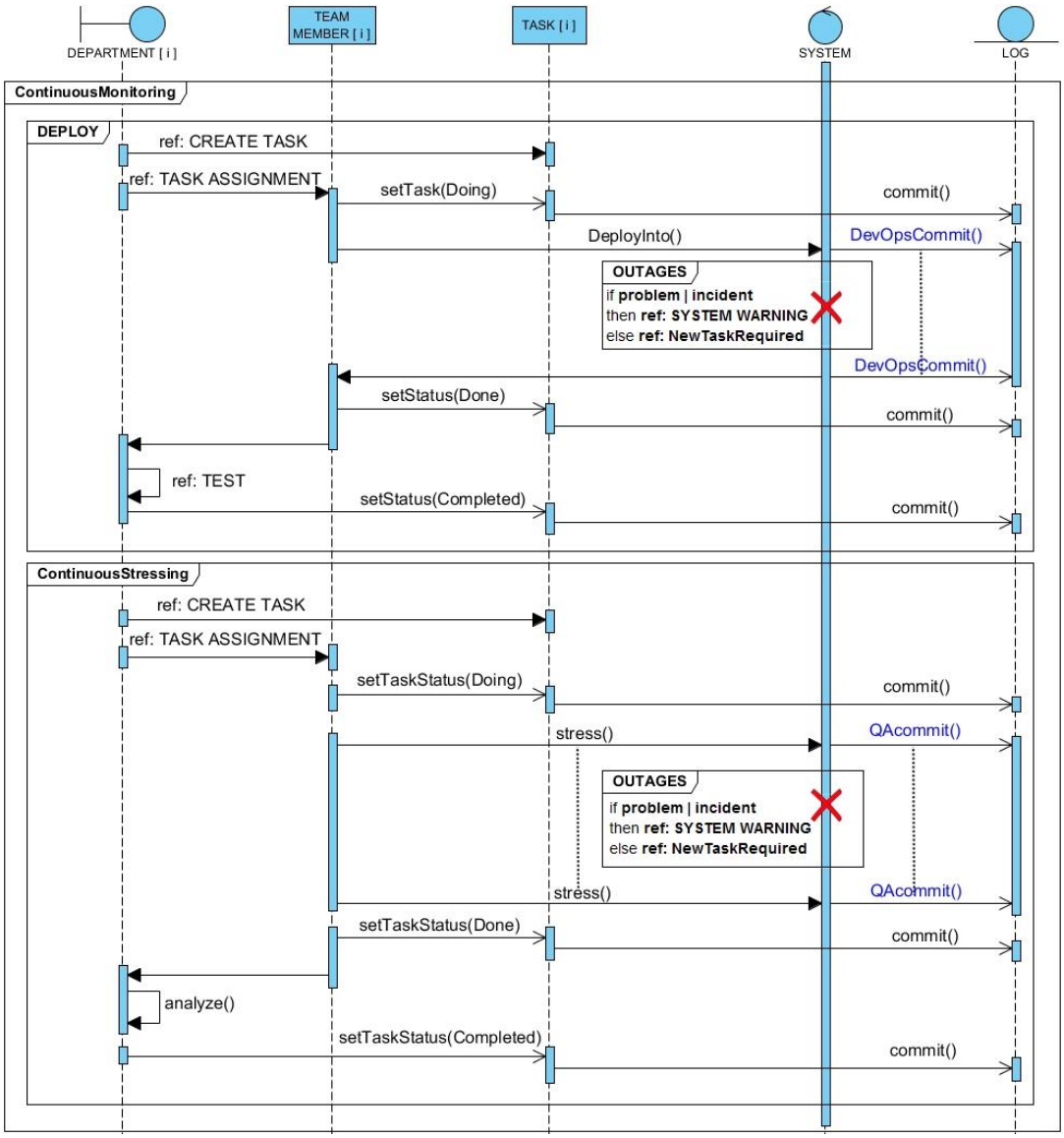

Figura 4.10: Diagramma di sequenza che rappresenta le attività svolte durante il deploy di un’applicazione e le procedure per irrobustire il sistema.

4.2.4 GESTIONE DELLE INTERRUZIONI

Questo scenario coinvolge due nuove entità:

- LEVEL 3: composto da 3 ingegneri che si occuperanno di risolvere i problemi già riscontrati in azienda.

- LEVEL 3S: dove risiede il capo ingegnere dell'azienda, che sarà chiamato in causa solo per le emergenze.

Il frame "PROCESSING OUTAGE", mostrato in ***Figura 4.11***, rappresenta le attività che si susseguono quando si verifica un incidente. Se il problema è conosciuto, la richiesta di aiuto è inoltrata al LEVEL 3. Qualora non fosse capace di risolverlo, la richiesta è inoltrata, previa approvazione, ad un livello superiore, il LEVEL 3S.

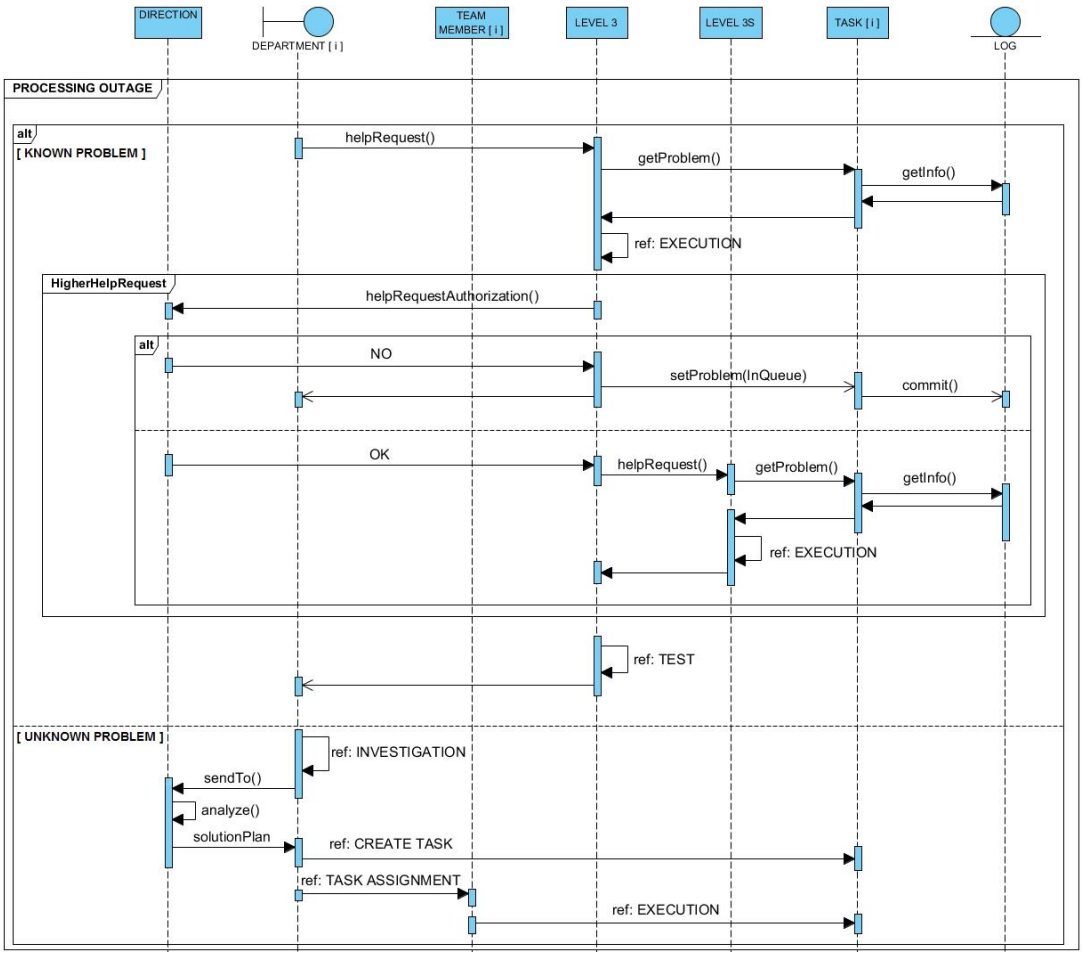

Figura 4.11: Diagramma di sequenza che rappresenta le procedure che gestiscono un'interruzione.

Se l'incidente è sconosciuto, dovrà essere condotta un'investigazione i cui risultati saranno analizzati dalla direzione aziendale, che stabilirà un "solution plan".

Tale indagine, mostrata nel dettaglio nel frame" INVESTIGATION" in ***Figura 4.12***, descrive le procedure che servono a recuperare le informazioni che riguardano modifiche e cambiamenti avvenuti prima del verificarsi dell'interruzione, come specificato dalla condizione di guardia presente all'interno del frame.

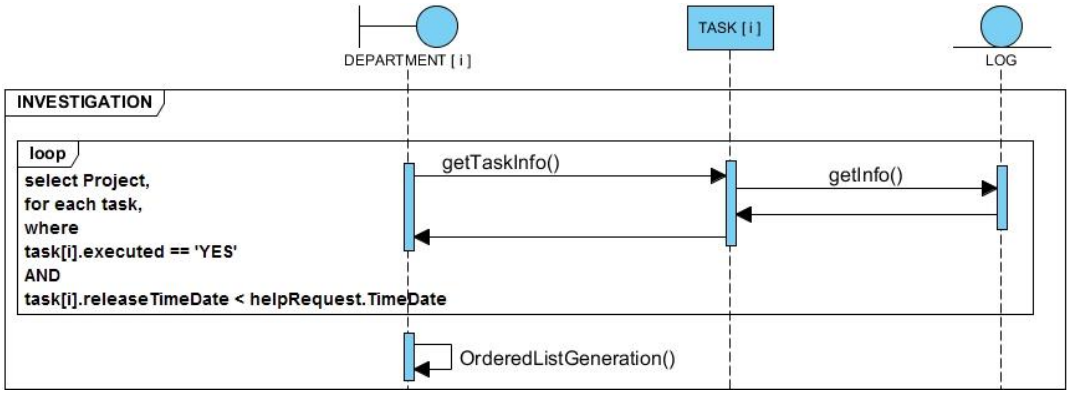

Figura 4.12: Diagramma di sequenza che rappresenta l'attività di investigazione.

4.2.5 GESTIONE DEI CAMBIAMENTI

Processo di aggiornamento utile al fine di prevenire incidenti. Come mostrato in ***Figura 4.13***, ogni dipartimento coinvolge i suoi dipendenti per creare una lista di cambiamenti, autorizzati e schedulati, da apportare nell'azienda.

I cambiamenti vengono classificati in base alla priorità e al rischio nel seguente modo:

- *High-risk:* devono essere autorizzati prima di essere implementati e schedulati.
- *Medium-risk:* bisogna consultare e farli approvare dalle persone potenzialmente coinvolte.
- *Low-risk:* sono cambiamenti standard, non necessitano di approvazione.

Fatto ciò, si creano i task e si assegnano ai dipendenti, richiamando i frame descritti in precedenza.

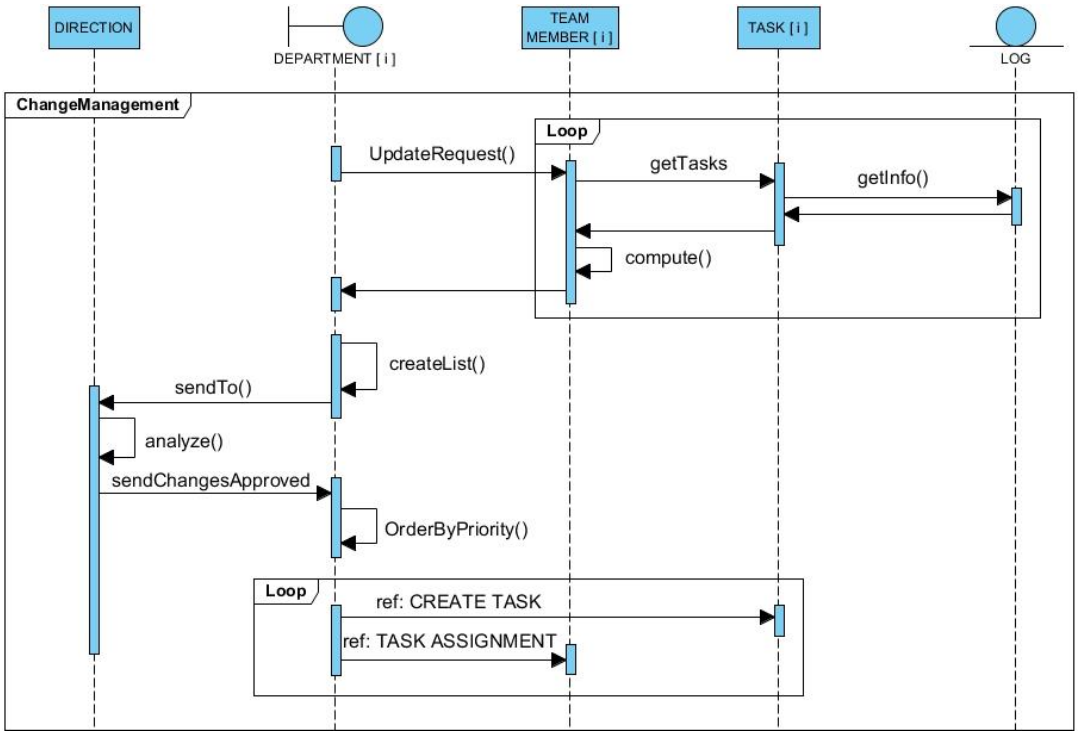

Figura 4.13: Diagramma di sequenza che rappresenta il processo di gestione del cambiamento.

4.3 FASE DI ANALISI

In questo paragrafo mostreremo le reti di Petri generate considerando i diagrammi di sequenza precedenti. Abbiamo simulato il comportamento di EDEN che, tramite gli strumenti in esso integrati, prende in input i diagrammi di sequenza e li trasforma automaticamente, seguendo le regole di trasformazione descritte nel paragrafo ***3.4***, in reti di Petri. In questo modo gli scenari DevOps possono essere esaminati tramite analisi più accurate grazie agli strumenti posseduti da EDEN, che acquisiscono le reti di Petri generate e permettono nello specifico di valutare le proprietà e il comportamento del sistema aziendale.

Le reti di Petri generate con questo approccio coprono le specifiche strutturali e comportamentali del sistema descritto negli scenari precedenti, rispettando l'ordine degli eventi così come si presentano nei diagrammi di sequenza.

4.3.1 FASE DI INIZIALIZZAZIONE DEL PROGETTO

La rete di Petri in ***Figura 4.14*** è stata generata considerando il diagramma di sequenza rappresentato in ***Figura 4.4***. Con l’arrivo di un nuovo progetto, l’azienda deve verificare di avere risorse necessarie per poterlo sviluppare. Se il progetto viene accettato, viene suddiviso in “Task” che saranno assegnati ai suoi dipendenti, selezionati in base alle competenze richieste per sviluppare uno specifico Task.

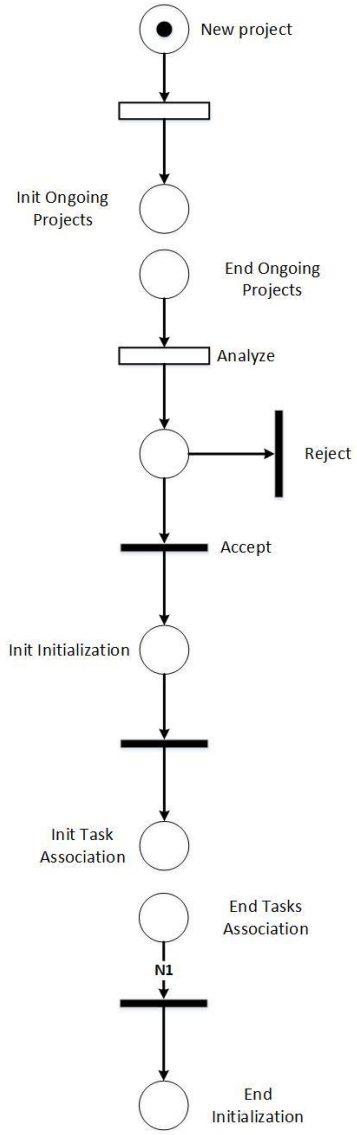

Figura 4.14: rete di Petri ottenuta trasformando il diagramma di sequenza rappresentato in Figura 4.4.

4.3.2 RECUPERO RISORSE

La rete di Petri in ***Figura 4.15*** è stata generata considerando il frame "ONGOING PROJECTS" del diagramma di sequenza rappresentato in ***Figura 4.4***. Il flusso di esecuzione della rete consente di recuperare le informazioni sugli N progetti a cui l'azienda sta lavorando. Ogni progetto consta di K Task. Per ogni task vengono recuperate le informazioni sullo stato di avanzamento del lavoro, in particolare sul dipendente che lo sviluppa e sulla risorsa impiegata. Tutto ciò verrà inserito in una lista accurata che verrà esaminata dall'alta direzione aziendale, a cui spetterà la decisione di accettare o rifiutare il nuovo progetto.

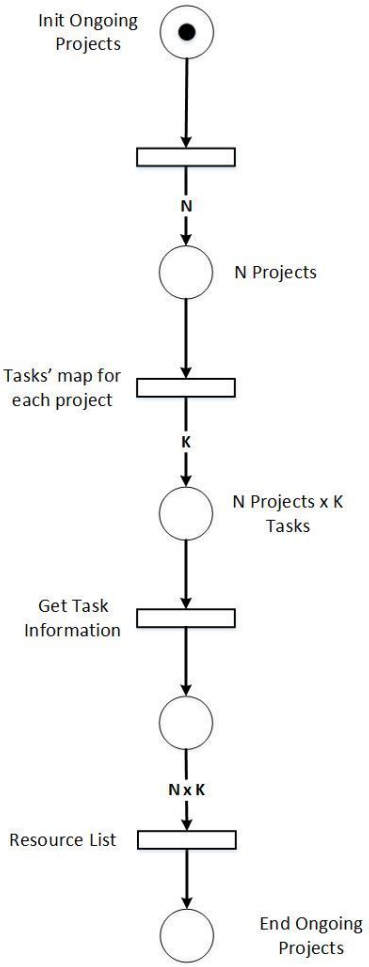

Figura 4.15: rete di Petri che rappresenta il frame "ONGOING PROJECTS" del diagramma di sequenza rappresentato in Figura 4.4

4.3.3 ASSOCIAZIONE DEI TASK

La rete di Petri in ***Figura 4.16*** è stata generata considerando i diagrammi di sequenza rappresentati in ***Figura 4.3*** e in ***Figura 4.4***. La rete prevede due flussi di esecuzione paralleli. Nella parte sinistra vengono creati i Task necessari per sviluppare il progetto con le relative specifiche. Nella parte destra invece inizialmente sono considerati tutti gli N dipendenti dell’azienda, ma ne vengono selezionati N1 perché non è detto che tutti gli N dipendenti siano disponibili o abbiano le competenze necessarie per eseguire uno specifico task. Fatto ciò, a ciascun dipendente selezionato vengono assegnati K Task, con $K \geq 1$, e viene generata la “Kanban Board” che renderà visibile il flusso di esecuzione di ogni Task per ciascun dipendente. Il flusso di esecuzione si blocca quando a tutti gli N1 dipendenti selezionati saranno associati uno o più Task.

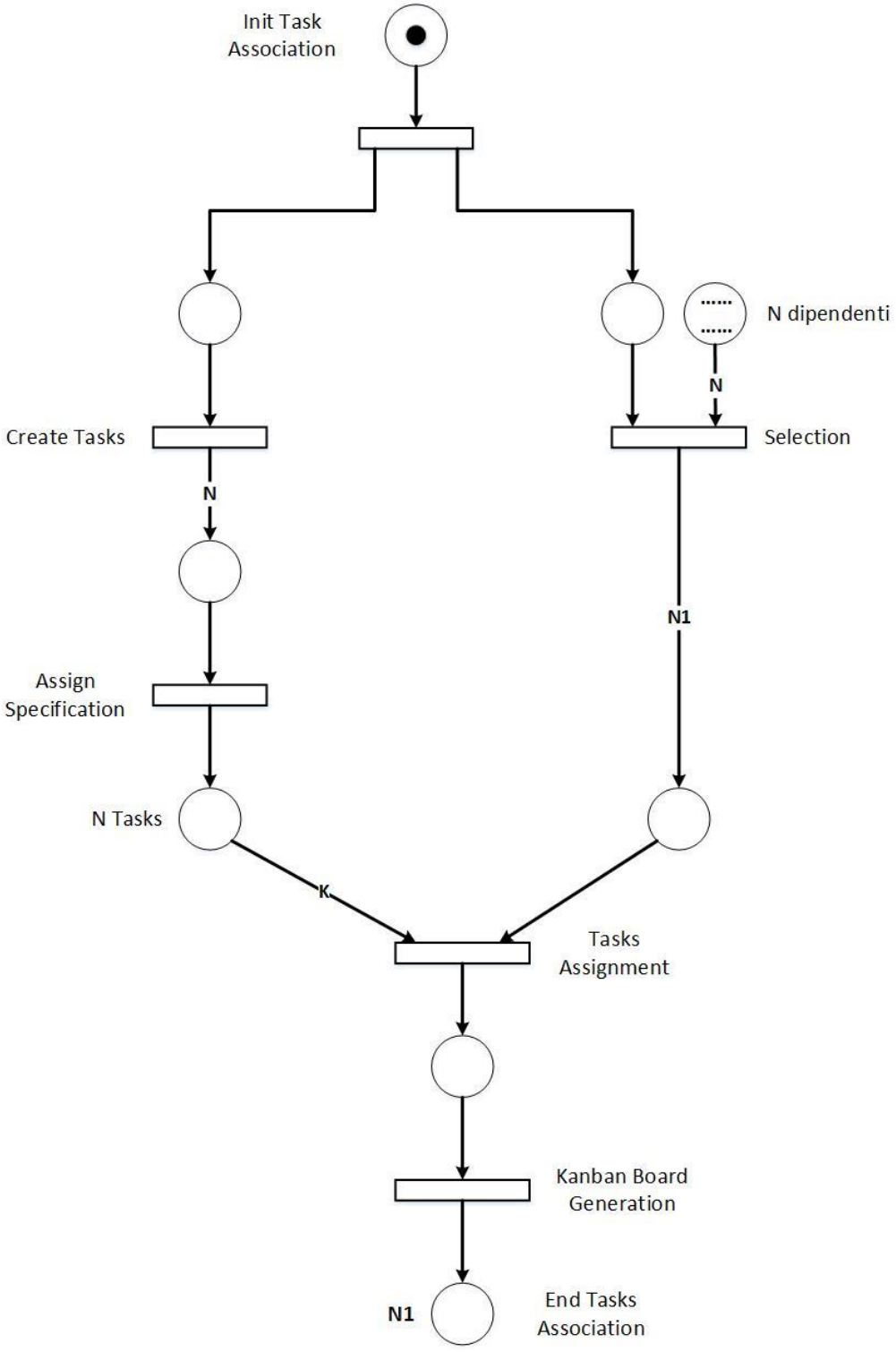

Figura 4.16: rete di Petri ottenuta trasformando i diagrammi di sequenza rappresentati in Figura 4.3 e in Figura 4.4.

4.3.4 ESECUZIONE

La rete di Petri in ***Figura 4.17*** è stata generata considerando il diagramma di sequenza rappresentato in ***Figura 4.6***.

Il flusso di esecuzione della rete si divide in tre fasi eseguite da tre attori differenti:

- N Developers, che implementano le specifiche dei task assegnati in precedenza.
- IT Operations, composto da N1 dipendenti, che testano il lavoro dei Developers.
- DevOps team, composto da N1 dipendenti, a cui spetta il compito di eseguire il deploy e quindi di apportare modifiche al sistema aziendale.

La transizione immediata dopo "Implementation & pre-test" fa il “firing” dopo che tutti i Developers hanno finito di svolgere i loro rispettivi task. Dopo di ciò, prima che il controllo del flusso di esecuzione passi ai membri dell’IT Operations, vi è la “Task Association”, che si riferisce alla rete rappresenta in ***Figura 4.16***, la quale si conclude con N1 token. La transazione immediata successiva quindi carica il posto “IT Operations” con N1 token, i quali avanzeranno uno alla volta. Il flusso di esecuzione poi prosegue in modo analogo come descritto precedentemente.

Se si verifica un problema, viene opportunamente gestito eseguendo il blocco di rete di Petri rappresentato in ***Figura 4.19***, e il Task che ha causato o è stato colpito dall’interruzione verrà rieseguito.

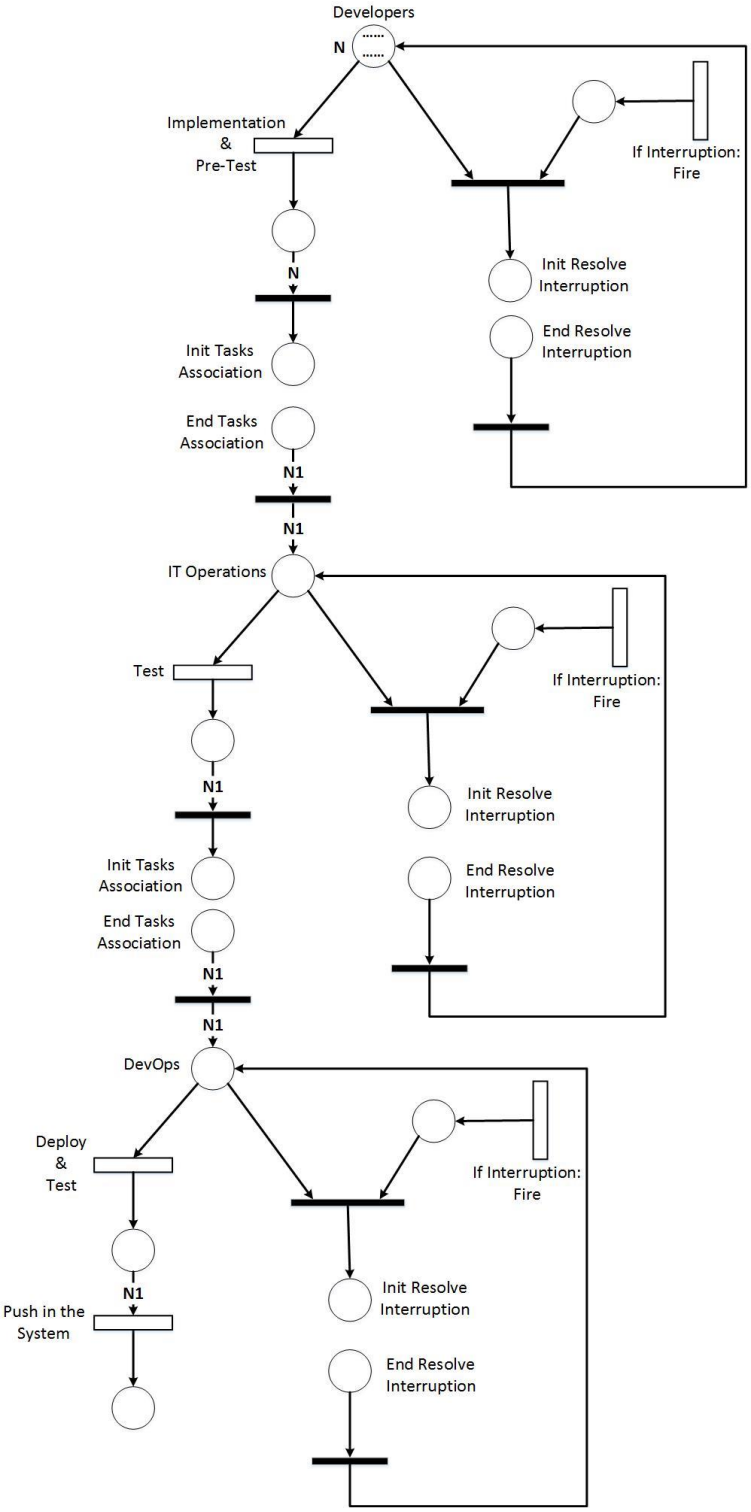

Figura 4.17: rete di Petri ottenuta trasformando il diagramma di sequenza rappresentato in Figura 4.6.

4.3.5 IRROBUSTIMENTO DEL SISTEMA AZIENDALE

La rete di Petri in ***Figura 4.18*** è stata generata considerando il diagramma di sequenza rappresentato in ***Figura 4.10***. Il sistema aziendale ha bisogno di essere “stressato” giornalmente, in modo da irrobustirsi e diminuire le vulnerabilità. Il flusso di esecuzione è analogo ai precedenti.

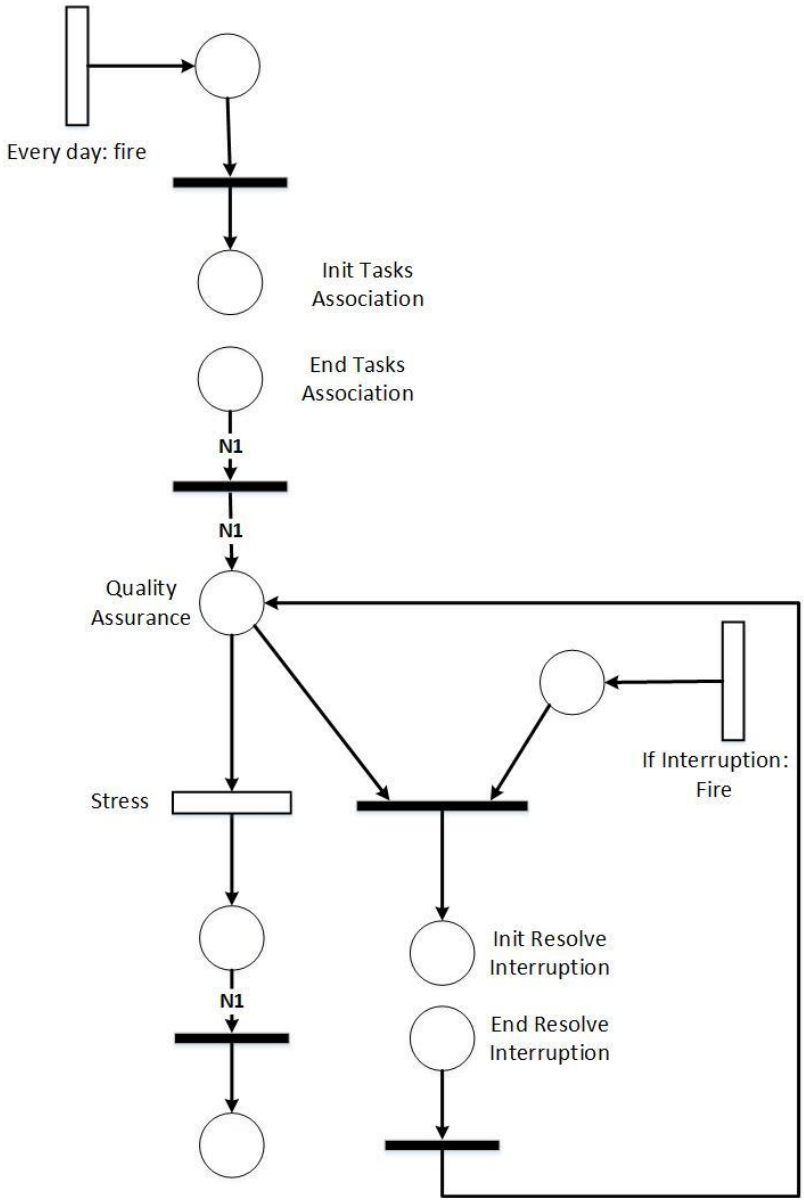

Figura 4.18: rete di Petri ottenuta trasformando il diagramma di sequenza rappresentato in Figura 4.10.

4.3.6 INTERRUZIONE

La rete di Petri in ***Figura 4.19*** è stata generata considerando il diagramma di sequenza rappresentato in ***Figura 4.9***. Un'interruzione si può verificare per due motivi:

- Vi è la necessità di aggiungere uno o più task.
- C'è un guasto nel sistema.

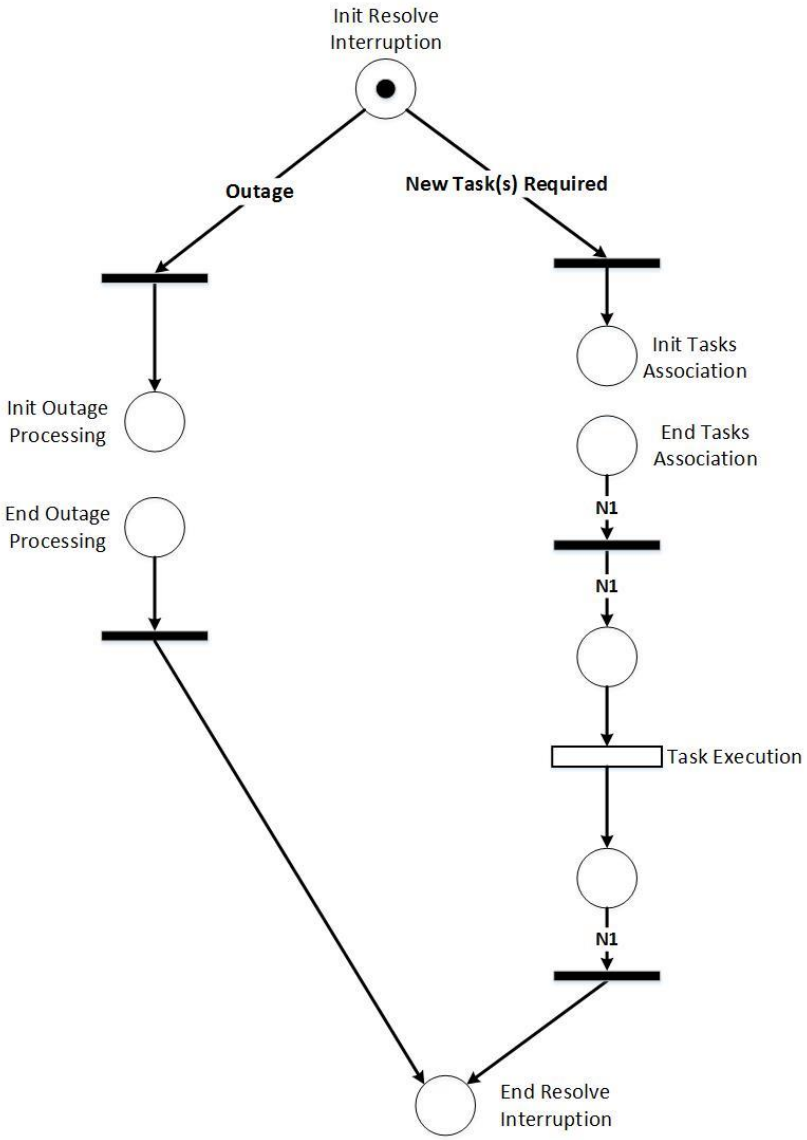

Figura 4.19: rete di Petri ottenuta trasformando il diagramma di sequenza rappresentato in Figura 4.9.

4.3.7 RIPRISTINO

La rete di Petri in ***Figura 4.20*** è stata generata considerando il diagramma di sequenza rappresentato in ***Figura 4.11***.

Il flusso di esecuzione dipende dal tipo di interruzione.

- Se un’interruzione è “conosciuta”, ovvero si è già verificata in passato, il controllo del flusso lo gestiscono i dipendenti del “Level 3”. Se hanno le competenze per farlo, risolvono l’interruzione, altrimenti passano il controllo del flusso al “Level 3S”, dove dipendenti più esperti, che sono risorse fondamentali per l’azienda, risolveranno l’interruzione subito o in un secondo momento, e il problema sarà aggiunto in una “coda” e posto in attesa di risoluzione.
- Se un’interruzione è sconosciuta, bisognerà investigare sulle cause che hanno provocato l’incidente e stabilire un “Solution Plan”.

Una volta stabilito il tipo di interruzione, il flusso di esecuzione prosegue come descritto nei casi precedenti. Se la fase di Test non sarà eseguita con successo, il Task verrà rieseguito.

L’arco con peso K che abilita la transazione immediata collegata al posto finale della rete "End Outage Processing" può assumere il valore 1 oppure N1, a seconda del flusso di esecuzione della rete. Se ad esempio viene eseguita la parte sinistra della rete, K vale N1 e l’ultima transizione immediata prima del "End Outage Processing" non potrà scattare finché non saranno stati eseguiti gli N1 tasks necessari per risolvere l’interruzione.

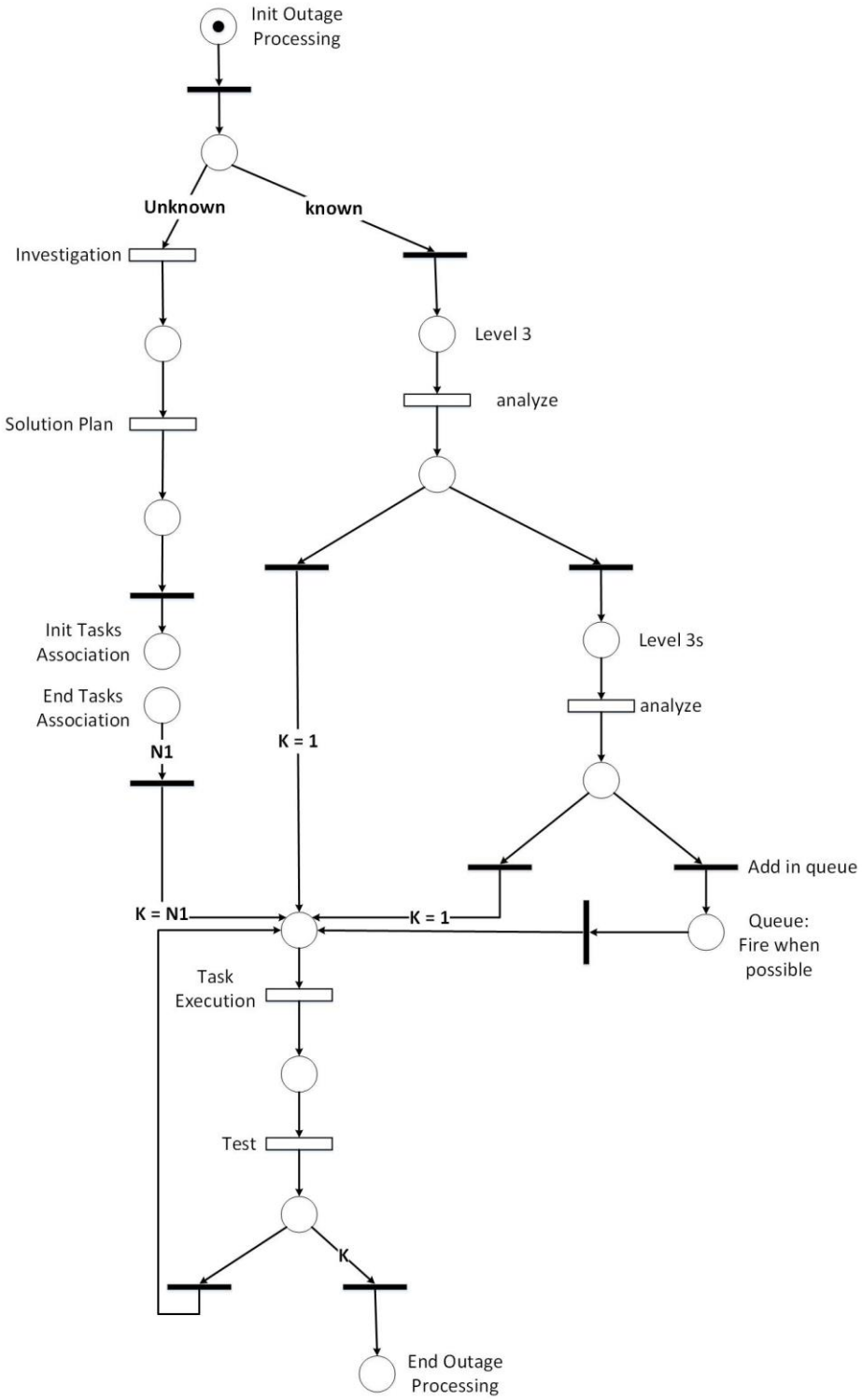

Figura 4.20: rete di Petri ottenuta trasformando il diagramma di sequenza rappresentato in Figura 4.11.

4.3.8 GESTIONE DEI CAMBIAMENTI

La rete di Petri in ***Figura 4.21*** è stata generata considerando il diagramma di sequenza rappresentato in ***Figura 4.13***. Ad ognuno degli N dipendenti viene chiesto di proporre dei cambiamenti in modo da migliorare il sistema aziendale. In analogia con lo scenario esaminato, ogni dipendente propone K cambiamenti, i quali saranno riordinati in una lista e consegnati all’alta direzione aziendale quando gli N dipendenti concluderanno questo processo, che li esaminerà e ordinerà in base alla priorità. Fatto ciò, il flusso di esecuzione prosegue come descritto nei casi precedenti e si conclude con l’assegnamento dei Tasks a N1 dipendenti, che saranno eseguiti da quest’ultimi in base alle priorità dei Task e dell’azienda.

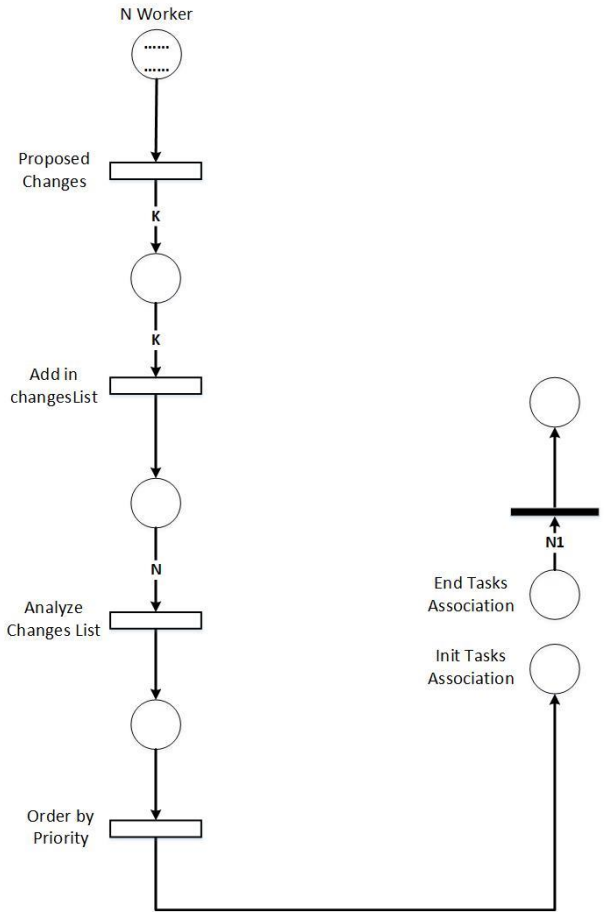

Figura 4.21: rete di Petri ottenuta trasformando il diagramma di sequenza rappresentato in Figura 4.13.

4.4 RIASSUNTO

In questo capitolo abbiamo esaminato un caso di studio reale DevOps. Nel paragrafo ***4.1***, abbiamo visto come applicando i principi di questa nuova metodologia, discussi nel paragrafo ***2.4***, l'azienda ha "snellito" i suoi processi, eliminando lavoro inutile dal sistema, e guadagnato "agilità", per far fronte alle richieste del mercato e alle esigenze dei clienti. Sposando tecniche di "collaborazione continua" e "monitoraggio continuo", gli ha sbarrato la strada verso il successo. Inoltre, l'uso del Cloud ha alleggerito il sistema aziendale, consentendo un ambiente sempre disponibile e configurabile automaticamente.

Nel paragrafo ***4.2***, abbiamo simulato il comportamento di EDEN supponendo che acquisisca i dati necessari a modellare le attività descritte negli scenari esposti tramite diagrammi di sequenza, che ci hanno consentito di osservare in maniera dettagliata l'ordine l'esecuzione dei processi che le costituiscono.

Nel paragrafo ***4.3*** abbiamo simulato il comportamento di EDEN supponendo di sfruttare uno strumento in esso integrato, ovvero quello che trasforma i diagrammi di sequenza in reti di Petri, che essendo un formalismo più potente, sono in grado di modellare le interazioni tra processi concorrenti e permettono di condurre analisi più dettagliate per valutare ad esempio l'affidabilità e le performance di un sistema.

Ad esempio possiamo analizzare i processi che compongono il sistema aziendale valutando le seguenti proprietà che caratterizzano una rete di Petri:

- Raggiungibilità, ovvero se è possibile ottenere un determinato stato della rete a partire da un altro. Utile ad esempio per verificare se lo stato corrente può raggiungere una condizione di "deadlock", che può causare interruzioni nel sistema.

- Reversibilità, ovvero se è possibile ripristinare un funzionamento a seguito di un guasto o problema. Tale proprietà consente di riportare la rete in uno stato funzionante.
- Limitatezza, ovvero controllare che i task assegnati ad un risorsa, rappresentati rispettivamente con token e posti, non superino una soglia stabilita a seguito dell’evoluzione della rete, al fine di evitare le bottleneck.
- Sicurezza, ovvero controllare che il flusso di esecuzione della rete proceda senza intoppi.
- Conservatività, dato che le reti di Petri modellizzano sistemi di allocazione di risorse, quando i token vengono usati per rappresentare le risorse è importante che si conservino.
- Vivezza, ovvero bisogna verificare se la rete può sempre continuare ad evolvere a partire da qualunque stato in cui si possa venire a trovare, oppure si blocca, tutta o in parte.

Il prossimo capitolo conclude il nostro lavoro. Discuteremo le nostre conclusioni in merito al lavoro svolto e le possibili direzioni future da poter intraprendere per proseguire e migliorare questo lavoro.

5 CONCLUSIONI

Il nostro lavoro si pone nell'ambito dell'Ingegneria del Software, in particolare nello studio dei modelli di sviluppo e del ciclo di vita del software, con particolare enfasi al modo in cui una metodologia di sviluppo o un modello di processo scompongono l'attività di realizzazione di prodotti software in sotto attività fra loro coordinate, il cui risultato finale è il prodotto stesso e tutta la documentazione ad esso associata. Abbiamo orientato la nostra ricerca su DevOps [7] [8], una metodologia di sviluppo del software "snella" e "agile", sviluppatasi nel corso degli ultimi anni. Essa mira alla comunicazione e collaborazione tra un'azienda e i dipartimenti da cui è composta, come sviluppo, produzione e controllo qualità, con lo scopo di aiutare un'organizzazione a sviluppare in modo continuo, rapido ed efficiente, prodotti e servizi software, garantendo alta qualità e reagendo immediatamente alle richieste del mercato e dei consumatori.

Nonostante all'inizio DevOps ha avuto un forte impatto dato i benefici che comporta, tuttavia risultano esserci ancora delle carenze su come adottare questa nuova metodologia sfruttando le risorse che un'azienda già possiede. Il nostro lavoro si è concentrato sullo studio di una possibile soluzione a questo problema in quanto non esistono ad oggi modelli e metodi che consentano di verificare effettivamente che i processi che costituiscono il ciclo di vita del software migliorino se si adotta un approccio DevOps.

Nel nostro lavoro abbiamo utilizzato EDEN[2], acronimo di "End-to-end Devops ENgineering", ovvero un approccio "model-based" in grado di pianificare e guidare i processi che costituiscono il ciclo di vita del software tramite un metodo DevOps,

sfruttando i modelli consolidati nel campo dell'ingegneria del software, come i diagrammi UML [15] e le reti di Petri [16].

Abbiamo esaminato un caso di studio DevOps tratto dal libro "The Phoenix Project" [3], che tratta degli sforzi compiuti da un'azienda nell'adottare questa nuova metodologia. Abbiamo assunto che l'azienda in questione usi EDEN, simulandone il comportamento. Abbiamo quindi modellato gli scenari esaminati tramite:

- Diagrammi di sequenza UML per identificare il comportamento di un sistema e mostrare le comunicazioni tra i diversi partecipanti.
- Reti di Petri, in quanto sono modelli formali basati su precise teorie matematiche e appropriati per modellizzare e analizzare sistemi.

Tramite il nostro studio, abbiamo individuato i possibili benefici che potrebbe ricavarne un'azienda utilizzando EDEN.

Primo, osserviamo che EDEN offre spunti che si adattano bene con i progetti, team e aziende che stanno adottando pratiche agili e non. EDEN aiuterebbe a trattare il ciclo di vita del software in maniera flessibile, cioè scegliendo la prossima fase, di sviluppo o operativo, in modo che sia la migliore scelta possibile in base ai risultati analizzati. Altre ricerche dovrebbero essere condotte per verificare questa osservazione e cosa implica, e anche per capire se e come EDEN potrebbe estendere strumenti agili, o altri strumenti tipici di approcci più tradizionali di ingegneria del software.

Secondo, EDEN potrebbe essere utile per decidere la migliore allocazione dei task considerando le varie figure aziendali tra le risorse di sviluppo del software. Ad esempio, si pensi ad uno scenario dove più aziende partner condividono lo sviluppo di un progetto software. EDEN potrebbe prendere questo scenario in considerazione, offrendo la possibilità di adottare schemi di pianificazione "ad hoc" per lo scenario in questione.

Da queste considerazioni concludiamo che EDEN:

a) Offre la possibilità di associare allo stesso tempo le notazioni consolidate in ambito dell'ingegneria del software, con l'obiettivo di guidare e governare il ciclo di vita del software in maniera DevOps.

b) Queste notazioni hanno un'interazione non banale e, se combinate, possono produrre delle informazioni importanti. Ad esempio si può passare da un modello ad un altro, scomporlo in sotto modelli, e avere quindi più modelli di processo che rappresentano gli sforzi nello sviluppare il prodotto sfruttando una metodologia DevOps.

c) EDEN potrebbe avere il potenziale per abbassare considerevolmente le barriere tecniche e organizzative [27] in DevOps.

Date queste premesse, ci sono diverse aree dove deve essere approfondita la nostra ricerca.

Primo, più ricerche devono essere investite sull'automazione di EDEN. Abbiamo osservato che molte informazioni di cui necessitano gli strumenti e i metodi di EDEN sono ampliamente utilizzati nelle aziende, come Jira, Trello, Mylyn. EDEN potrebbe essere attrezzato per recuperare le informazioni necessarie automaticamente. Tuttavia, esistono diverse applicazioni che gestiscono il ciclo di vita, che prendono il nome di "Application Lifecycle Management" [28], che potrebbero lavorare bene interfacciandosi con EDEN. Ad esempio esistono strumenti che lungo la "delivery pipeline" potrebbero alimentare EDEN e ricevere feedback sullo stato del lavoro svolto.

Secondo, bisogna investire nella ricerca nel campo "multi-view modeling", in modo passare da un modello ad un altro con il minimo sforzo. Bisogna indagare sull'uso di diversi modelli in base a: i requisiti dell'applicazione, la fase di sviluppo e i dati disponibili. Questi modelli potrebbero essere usati sia in maniera indipendente che in maniera combinata. Ad esempio, considerando le reti di Petri concentrandoci sulla concorrenza per utilizzare una risorsa, bisogna assegnare una priorità ai task.

Terzo, più ricerche devono essere investite nell'analisi di incertezza. Alcuni task nella fase iniziale del processo di sviluppo non possono essere precisamente definiti, per questo è importante includere tecniche di analisi e gestione dell'incertezza per prendere questi aspetti in considerazione. Specificamente, possiamo distinguere due principali direzioni di ricerca:

- *incompletezza dei modelli*: ulteriori ricerche si devono concentrare sulla definizione di approcci incrementali in grado di includere pezzi di informazione nei modelli non appena queste sono disponibili.
- *Analisi dei modelli*: ulteriori ricerche in EDEN si devono concentrare nel definire tecniche in grado di dedurre proprietà non definite su moduli o task basate sui requisiti generali di un sistema e dalla stima ottenuta dall'analisi dei modelli disponibili.

Per sviluppi futuri su EDEN, tutte le componenti utilizzate per la stesura di questo lavoro sono disponibili on-line [29].

BIBLIOGRAFIA

[1] Ian Sommerville. 2006. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[2] D. A. TAMBURRI, D. PEREZ-PALACIN and R. MIRANDOLA, *Internal Project,* 2016.

[3] Gene Kim, Kevin Behr, and George Spafford. 2013. *The Phoenix Project: A Novel about It, Devops, and Helping Your Business Win* (1st ed.). IT Revolution Press.

[4] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001). Manifesto for Agile Software Development *Manifesto for Agile Software Development.*

[5] Schwaber, K. and Sutherland, J. (2001). *The Scrum guide.*

[6] R. Kohavi, R. Longbotham, D. Sommerfield, and R. M. Henne, "Controlled experiments on the web: survey and practical guide," Data Mining and Knowledge Discovery, vol. 18, no. 1, pp. 140-181, 2009.

[7] *DevOps For Dummies*, IBM Limited Edition Published by John Wiley & Sons, Inc. 111 River St. Hoboken, NJ Copyright 2014 by John Wiley & Sons.

[8] Bass, L. J., Weber, I. M., Zhu, L. (2015). *DevOps - A Software Architect's Perspective*, Addison-Wesley. ISBN: 978-0-1340-4984-7.

[9] D. D. Scott Lowe,Building an Agile Infrastructure for DevOps, 2015, https://www.hpe.com/h20195/v2/getpdf.aspx/4AA6-2192EEW.pdf?ver=1.0.

[10] M. Kapadia, *Comparing DevOps to traditional IT: Eight Key differences,* https://devops.com/comparing-devops-traditional-eight-key-differences/, 2015.

[11] J. E. Matson, B. E. Barrett, and J. M. Mellichamp. 1994. Software Development Cost Estimation Using Function Points. *IEEE Trans. Softw. Eng.* 20, 4 (April 1994), 275-287.

[12] Ryan Polk. 2011. Agile and Kanban in Coordination. In *Proceedings of the 2011 Agile Conference*(AGILE '11). IEEE Computer Society, Washington, DC, USA, 263-268.

[13] M. Elallaoui, K. Nafil and R. Touahni, "Automatic generation of UML sequence diagrams from user stories in Scrum process," *2015 10th International Conference on Intelligent Systems: Theories and Applications (SITA)*, Rabat, 2015, pp. 1-6.

[14] Yue, T., Briand, L. C., Labiche, Y.: Automatically deriving UML sequence diagrams from use cases. Technical Report, Carleton University, Canada, TR-SCE-10-03 (2010).

[15] Object Management Group, 2013, OMG Unified Modeling Language (OMG UML), Version 2.5 Beta 2. OMG Document formal/13-09-05, September 2013. http://www.omg.org/spec/UML/2.5/Beta2.

[16] T. Murata, "*Petri Nets: Properties, Analysis and Applications*", an invited survey paper, Proceedings of the IEEE, Vol.77, No.4 pp.541-580, April, 1989.

[17] Van Der Aalst, W. M. P. (1998). The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66.

[18] Mohamed A. Ameedeen, Behzad Bordbar, and Rachid Anane. 2009. A Model Driven Approach to the Analysis of Timeliness Properties. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and*

Applications (ECMDA-FA '09), Richard F. Paige, Alan Hartman, and Arend Rensink (Eds.). Springer-Verlag, Berlin, Heidelberg, 221-236.

[19] Simona Bernardi, Susanna Donatelli, and José Merseguer. 2002. From UML sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the 3rd international workshop on Software and performance* (WOSP '02). ACM, New York, NY, USA, 35-45.

[20] *Kurose, J.F., MacNair, E.A., & Sauer, C.H. . (1982). The research queueing package: past, present, and future. AFIPS National Computer Conference.*

[21] Marco Ajmone Marsan, G. Balbo, Gianni Conte, S. Donatelli, and G. Franceschinis. 1994. *Modelling with Generalized Stochastic Petri Nets* (1st ed.). John Wiley & Sons, Inc., New York, NY, USA.

[22] P. Bonet, C. Lladó, R. Puigjaner, and W. Knottenbelt, "PIPE v2.5: A Petri net tool for performance modelling," in Proc. 23rd Latin American Conference on Informatics (CLEI'07), San Jose, Costa Rica, October 2007.

[23] *G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. 1995. GreatSPN 1.7: graphical editor and analyzer for timed and stochastic Petri nets. Perform. Eval. 24, 1-2 (November 1995), 47-68.*

[24] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. 2006. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons.

[25] Mohamed Ariff Ameedeen and Behzad Bordbar. 2008. A Model Driven Approach to Represent Sequence Diagrams as Free Choice Petri Nets. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference* (EDOC '08). IEEE Computer Society, Washington, DC, USA, 213-221.

[26] England, D.; Palanque, P. A.; Vanderdonckt, J. & Wild, P. J., ed. (2010), *Task Models and Diagrams for User Interface Design, 8th International Workshop, TAMODIA 2009, Brussels, Belgium, September 23-25, 2009, Revised Selected Papers* , Vol. 5963 , Springer .

[27] Damian A. Tamburri, Patricia Lago, and Hans van Vliet. 2013. Organizational social structures for software engineering. *ACM Comput. Surv.* 46, 1, Article 3 (July 2013), 35 pages.

[28] H. Lacheiner and R. Ramler, "Application Lifecycle Management as Infrastructure for Software Process Improvement and Evolution: Experience and Insights from Industry," *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, Oulu, 2011, pp. 286-293.

[29] https://github.com/beppev/master_thesis .