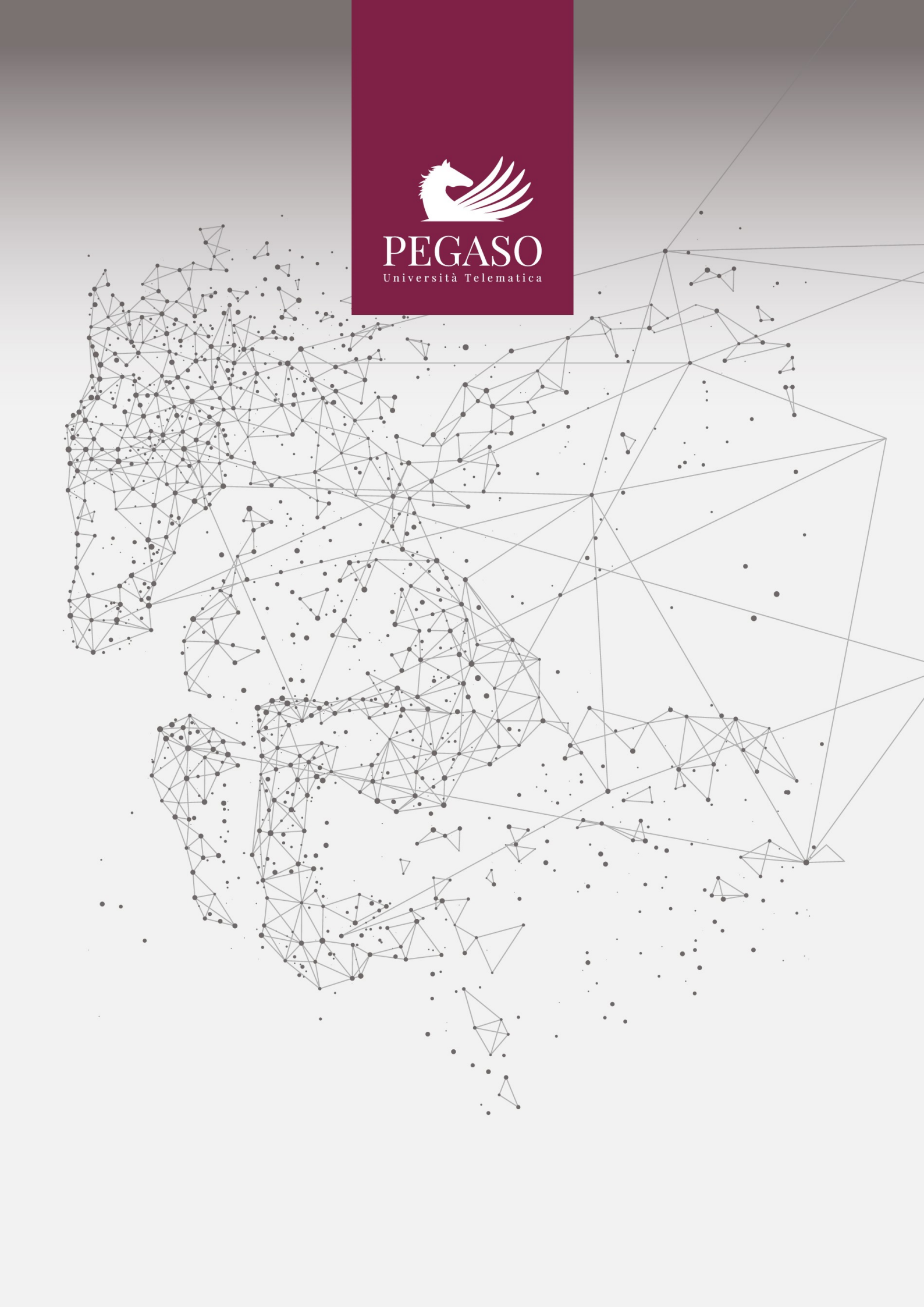
PEGASO
Università Telematica

Indice

1. Premessa

Il **Software Project Management** rappresenta un ambito essenziale per garantire il successo dei progetti software all'interno di contesti professionali sempre più complessi e dinamici. Non si tratta semplicemente di gestire la scrittura del codice, ma di orchestrare un insieme eterogeneo di elementi: risorse umane, strumenti tecnologici, vincoli economici, scadenze temporali e aspettative degli stakeholder. Questa disciplina richiede l'adozione di un punto di vista manageriale, in cui il focus si sposta dal codice al contesto organizzativo e strategico nel quale il software viene ideato, progettato e realizzato.

Un progetto software ha, per sua natura, una forte componente di **incertezza** e **cambiamento**: requisiti che evolvono, tecnologie emergenti, dinamiche di mercato che impongono continui aggiustamenti. In tale contesto, la figura del project manager diventa centrale. Il suo ruolo consiste nel pianificare, organizzare, monitorare e chiudere progetti software, assicurandosi che ogni fase venga svolta nel rispetto di **tempi**, **costi** e **qualità** attesi. Le sue decisioni non influenzano solo l'esito del singolo progetto, ma anche la reputazione dell'organizzazione, la soddisfazione del cliente e la crescita professionale del team.

Attraverso l'analisi di modelli teorici e l'utilizzo di strumenti specifici, il Software Project Management consente di affrontare la **complessità tecnica** e **organizzativa** di questi progetti, fornendo un framework strutturato per la loro realizzazione. Non meno importante è l'attenzione rivolta agli **aspetti umani**: il coordinamento dei team, la gestione delle aspettative, la comunicazione efficace e la leadership sono componenti imprescindibili del lavoro manageriale.

Questa dispensa si propone di esplorare i concetti chiave del Software Project Management, offrendo una visione integrata delle sue fasi fondamentali, delle conoscenze necessarie, dei modelli organizzativi e degli strumenti operativi. Al termine del percorso, sarà possibile comprendere non solo cosa significhi gestire un progetto software, ma anche come farlo in modo efficace e professionale, contribuendo attivamente alla **realizzazione di sistemi di qualità**.

2. Introduzione e Panoramica

Nel contesto della gestione dei progetti software, la figura del **Project Manager** emerge come un elemento cruciale. Il suo compito principale è garantire che il prodotto finale venga consegnato nel rispetto dei **vincoli temporali**, **budgetari** e **qualitativi**. A differenza dello sviluppatore, che interviene in modo operativo sul codice e sulla documentazione, il project manager coordina le attività attraverso l'impiego di **modelli gestionali**, strumenti di pianificazione e capacità decisionali.

Uno degli aspetti cardine della sua attività è la capacità di gestire simultaneamente una molteplicità di **stakeholder**: clienti, fornitori, membri del team, e figure dirigenziali. Ciascuno di questi attori ha esigenze, aspettative e competenze differenti, il che rende la comunicazione e la negoziazione strumenti indispensabili. Il project manager deve saper bilanciare queste esigenze e prendere decisioni che mantengano il progetto allineato agli obiettivi stabiliti.

I progetti software sono composti da quattro elementi interdipendenti:

- **Outcome**: il risultato atteso, ovvero il deliverable, che può assumere la forma di un prodotto finito, di un servizio o di un insieme di funzionalità specificate.
- **Work**: tutte le attività e i task necessari per ottenere l'outcome. Questa componente include l'analisi, lo sviluppo, i test, la documentazione e tutte le operazioni correlate.
- **Schedule**: la pianificazione temporale delle attività, con l'identificazione di date di inizio e fine, milestones e dipendenze.
- **Resources**: le risorse umane, tecnologiche e finanziarie necessarie per portare avanti le attività previste.

La modifica di uno solo di questi elementi implica inevitabilmente un impatto sugli altri. È dunque fondamentale disporre di strumenti che consentano di modellare, monitorare e reagire a tali variazioni.

Due sfide ricorrenti nella gestione dei progetti software sono la **complessità** e il **cambiamento**. La complessità deriva dalla presenza di componenti tecnologiche eterogenee, dalla difficoltà di prevedere l'evoluzione del progetto e dall'interazione tra attori diversi. Il cambiamento, invece, è spesso legato all'emergere di nuovi requisiti, alla mutazione delle priorità aziendali, o alla necessità di adattarsi a vincoli esterni.

Per affrontare queste problematiche, si ricorre a modelli concettuali come la **Work Breakdown Structure (WBS)**, che permette di scomporre il lavoro in task gestibili, o i **task model**, che rappresentano le dipendenze temporali tra le attività. Inoltre, strumenti come gli **organization chart** definiscono ruoli, responsabilità e interazioni tra i membri del team.

Il processo di gestione di un progetto può essere suddiviso in quattro macro-attività:

1. **Planning**: definizione degli obiettivi, delle attività, delle risorse e dei tempi.
2. **Organizing**: strutturazione del team e assegnazione dei compiti.
3. **Controlling**: monitoraggio dell'avanzamento e gestione delle deviazioni.
4. **Terminating**: chiusura del progetto, consegna del prodotto e analisi dei risultati.

Oltre a queste attività gestionali, ogni progetto attraversa diverse **fasi dinamiche**, che includono:

- **Conception**: definizione preliminare dell'idea e valutazione di costi e benefici.
- **Definition**: formalizzazione di obiettivi, vincoli, architettura e piano iniziale.
- **Start**: creazione dell'infrastruttura operativa e avvio delle attività.
- **Steady State**: fase operativa con produzione, monitoraggio e gestione del rischio.
- **Termination**: consegna finale, verifica dell'aderenza ai requisiti e analisi post-mortem.

Questa struttura consente di affrontare la gestione dei progetti software in modo **sistematico**, favorendo la previsione, il controllo e la capacità di adattamento.

3. Le Fasi Dinamiche di un Progetto Software

La gestione di un progetto software prevede una suddivisione in fasi ben definite, ognuna delle quali contribuisce a strutturare e organizzare le attività necessarie per il successo del progetto. Le fasi dinamiche rappresentano la sequenza temporale attraverso cui un progetto evolve, dalla sua ideazione fino alla chiusura. Esse forniscono una **cornice metodologica** utile per delineare obiettivi, deliverable e responsabilità in ciascun momento del ciclo di vita.

La fase di **Conception** è dedicata all'emergere dell'idea progettuale e alla sua prima valutazione. In questa fase iniziale, gli stakeholder avviano discussioni informali e valutano la **fattibilità economica e tecnica** del progetto. Si analizzano i potenziali ritorni sugli investimenti, i benefici attesi e i rischi preliminari. La **decisione di proseguire** viene presa sulla base di uno studio di fattibilità che considera le risorse disponibili, le tecnologie richieste e l'impatto previsto sull'organizzazione.

Segue la fase di **Definition**, nella quale il progetto assume una struttura più formale. Il project manager, in collaborazione con il cliente e con l'architetto software, definisce gli **obiettivi concreti**, le **scadenze**, l'**architettura iniziale del sistema** e il primo **Software Project Management Plan (SPMP)**. Questo documento costituisce il riferimento operativo per le fasi successive e dettaglia le attività previste, le responsabilità assegnate, i rischi identificati e i criteri di accettazione. Un **accordo contrattuale** tra cliente e fornitore formalizza le aspettative reciproche.

La fase di **Start** segna l'inizio operativo del progetto. In questo momento vengono predisposti gli ambienti di lavoro, selezionati gli strumenti per il versionamento, la comunicazione e il testing, e si procede alla **formazione dei team di sviluppo**. Un'attività importante in questa fase è la **compilazione dello skill matrix**, utile per mappare le competenze dei membri del team e assegnare i ruoli in maniera efficace. Il **kickoff meeting**, infine, serve a condividere gli obiettivi comuni, chiarire le modalità di lavoro e rafforzare la coesione tra i partecipanti.

Durante la fase di **Steady State**, si entra nel cuore produttivo del progetto. I team lavorano ai deliverable pianificati, mentre il project manager si occupa di **monitorare costantemente lo stato di avanzamento**, gestire i rischi, affrontare gli imprevisti e mantenere attiva la comunicazione con gli stakeholder. Attività cruciali includono la **gestione del rischio**, il **replanning** delle attività in caso di deviazioni significative e la verifica della qualità del lavoro svolto. È in questa fase che emergono eventuali criticità organizzative, tecniche o relazionali, che devono essere affrontate in modo tempestivo ed efficace.

Infine, la fase di **Termination** conclude il ciclo di vita del progetto. Essa prevede la **consegna formale del prodotto** al cliente, l'esecuzione dei test di accettazione secondo i criteri stabiliti, l'installazione

del sistema e la migrazione dei dati. Un'attività fondamentale è il **postmortem**, che consiste nell'analisi retrospettiva dell'intero progetto. L'obiettivo è identificare le lezioni apprese, riconoscere i successi e gli errori, e produrre raccomandazioni utili per migliorare i processi futuri.

L'adozione di una struttura a fasi consente non solo di mantenere il controllo sul progetto, ma anche di favorire un approccio iterativo e incrementale alla gestione. Ogni fase rappresenta un **momento di verifica**, in cui è possibile valutare i risultati ottenuti, ricalibrare le strategie e procedere con maggiore consapevolezza verso le tappe successive.

4. Software Engineering Body of Knowledge (SWEBOK)

Il **Software Engineering Body of Knowledge (SWEBOK)** rappresenta una delle principali iniziative a livello internazionale per la sistematizzazione delle conoscenze nell'ambito dell'ingegneria del software. Promosso dall'**IEEE Computer Society**, il progetto ha l'obiettivo di formalizzare la disciplina dell'ingegneria del software come una vera e propria branca dell'ingegneria, con contenuti, metodi e finalità coerenti con le esigenze del settore industriale e accademico.

SWEBOK funge da **quadro di riferimento** condiviso, utile per la formazione accademica, la certificazione professionale e la progettazione di curricula universitari. La necessità di definire un linguaggio comune tra professionisti, aziende, istituzioni e ambienti educativi ha spinto alla codificazione di un insieme di **aree di conoscenza** che coprono i molteplici aspetti della progettazione, sviluppo e manutenzione del software.

Una delle caratteristiche distintive di SWEBOK è l'organizzazione della conoscenza in **15 aree tematiche (Knowledge Areas, KAs)**. Ognuna di queste aree descrive concetti fondamentali, tecniche, strumenti e metodi riconosciuti come best practice nel settore. Questa suddivisione permette di affrontare la complessità dell'ingegneria del software in modo strutturato e completo, evidenziando le competenze necessarie per ogni ambito operativo.

Le aree di conoscenza includono, tra le altre:

1. **Software Requirements**: si occupa della raccolta, analisi, validazione, documentazione e gestione dei requisiti. Sono trattate tecniche come interviste, scenari, casi d'uso e modellazione dei requisiti.
2. **Software Design**: riguarda l'architettura del sistema, la progettazione dettagliata, i design patterns, e si focalizza sulla qualità strutturale e manutenibilità del software.
3. **Software Construction**: corrisponde alla fase di codifica. Includendo attività di programmazione, debugging, documentazione e uso degli ambienti di sviluppo, richiede una profonda conoscenza dei linguaggi e paradigmi.
4. **Software Testing**: tratta le metodologie di verifica e validazione, con focus su test unitari, d'integrazione, di sistema e d'accettazione. Vengono incluse strategie come black-box e white-box, e l'automazione dei test.
5. **Software Maintenance**: si riferisce alle attività post-rilascio, come correzione di bug, aggiornamenti evolutivi e adattamenti tecnologici. È una fase continua che necessita pianificazione e risorse dedicate.

Il contributo di SWEBOK non si limita agli aspetti tecnici. Altre aree fondamentali includono:

6. **Software Configuration Management**: comprende il versionamento, la gestione delle release, l'integrazione continua e il coordinamento tra team distribuiti.
7. **Software Engineering Management**: si focalizza sugli aspetti gestionali, come la pianificazione, il monitoraggio, la stima dei costi e la gestione dei rischi.
8. **Software Engineering Process**: analizza i modelli di processo come waterfall, agile, iterativo, e descrive attività di standardizzazione e miglioramento continuo.
9. **Software Engineering Models and Methods**: copre le tecniche di modellazione (es. UML, diagrammi di stato), metodi formali e semi-formali, e approcci strutturati allo sviluppo.
10. **Software Quality**: include la definizione delle metriche di qualità, l'assicurazione e il controllo qualità, e la verifica della conformità agli standard.

Oltre a queste, SWEBOK riconosce l'importanza degli aspetti **etici, economici e fondamentali**:

11. **Professional Practice**: affronta i comportamenti etici, le dinamiche di team, la comunicazione e la leadership nel contesto professionale.
12. **Software Engineering Economics**: analizza i costi del software, il ROI, le analisi costi-benefici e le decisioni di trade-off progettuale.
13. **Computing Foundations**: copre argomenti di base come strutture dati, algoritmi, sistemi operativi e reti.
14. **Mathematical Foundations**: include logica, teoria degli insiemi, teoria dei grafi e algebra booleana.
15. **Engineering Foundations**: tratta principi ingegneristici come affidabilità, sicurezza, valutazione delle performance e integrazione con sistemi fisici.

La funzione principale del SWEBOK è fornire una **base comune di conoscenze** che faciliti il dialogo e la cooperazione tra i diversi attori coinvolti nella produzione di software. Inoltre, rappresenta un **strumento di orientamento** per chi desidera intraprendere un percorso di formazione avanzata o ottenere certificazioni professionali riconosciute a livello internazionale. La sua adozione consente di sviluppare progetti software più coerenti, affidabili e allineati agli standard di qualità globali.

5. Concetti Generali: Scomposizione, Task e Work Package

Alla base della gestione di progetti software risiede il principio metodologico del **divide et impera**, ovvero la scomposizione sistematica del lavoro complesso in unità più semplici e gestibili. Tale approccio consente di affrontare l'**elevata complessità** che caratterizza i moderni sistemi software, agevolando la pianificazione, l'allocazione delle risorse e il monitoraggio dell'avanzamento del progetto. Il presupposto di fondo è che non sia possibile affrontare un progetto di grandi dimensioni come un monolite: occorre invece suddividerlo in attività elementari, comprensibili e monitorabili.

Nel contesto del Software Project Management, tale suddivisione si concretizza attraverso l'identificazione di **task** e **attività**. Un **task** rappresenta la più piccola unità di lavoro gestibile. Esso è caratterizzato da una durata specifica, da input e output ben definiti, da un responsabile dell'esecuzione e dalla produzione di uno o più **work products**. Le **attività**, invece, sono aggregazioni di task correlati e spesso coincidono con le fasi di un progetto (come l'analisi dei requisiti, la progettazione o il collaudo). Le attività hanno un orizzonte temporale più ampio e comprendono al loro interno diverse unità operative.

Ogni attività produce **work products**, i quali si distinguono in:

- **Deliverables**: prodotti finali destinati al cliente, come manuali d'uso, applicazioni software o report tecnici.
- **Internal work products**: artefatti intermedi utilizzati internamente dal team, come diagrammi UML, prototipi, piani di test.

Tali output vengono formalizzati attraverso **work packages**, documenti che definiscono l'insieme di compiti assegnati, con indicazione di obiettivi, risorse impiegate, durata stimata, input necessari e responsabilità coinvolte. I work package possono variare in complessità: da una semplice azione assegnata ad un singolo collaboratore, fino a intere sezioni di documentazione tecnica.

Uno strumento cardine nella pianificazione è la **Work Breakdown Structure (WBS)**, che rappresenta la gerarchia del lavoro da svolgere. Essa suddivide il progetto in livelli successivi: dalle fasi principali, si scende fino a singole attività e task, in modo da ottenere una mappa chiara e completa del lavoro necessario. La WBS permette di stimare costi e tempi, di assegnare responsabilità e di individuare eventuali criticità. Tuttavia, non fornisce informazioni sulla sequenza temporale: per questo è necessario introdurre il **task model**.

Il **task model** o **network diagram** rappresenta le **dipendenze temporali** tra i task. Ogni nodo corrisponde a un task, mentre le frecce descrivono i vincoli di precedenza. Uno degli elementi centrali del task model è il **critical path**, ovvero la sequenza di attività che determina la durata minima del progetto. La

gestione del critical path è cruciale per evitare ritardi, poiché qualsiasi slittamento su di esso incide direttamente sulla consegna complessiva. Il modello consente inoltre di calcolare lo **slack time**, cioè il margine di tempo entro cui un task può essere posticipato senza influenzare l'intero progetto.

Oltre alle dipendenze logiche, alcuni task sono soggetti a **vincoli temporali esterni**, derivanti da accordi con il cliente o da limitazioni operative. Il **SWEBOK** propone una classificazione standard di tali vincoli:

- **ASAP (As Soon As Possible)**: l'attività deve iniziare appena possibile.
- **ALAP (As Late As Possible)**: l'attività deve terminare il più tardi possibile senza ritardare il progetto.
- **MSO (Must Start On)**: inizio vincolato a una data specifica.
- **MFO (Must Finish On)**: termine vincolato a una data precisa.

Questi vincoli risultano particolarmente rilevanti nelle fasi di **rilascio del software**, quando la disponibilità delle risorse o le scadenze di mercato impongono rigidi limiti temporali.

Un ulteriore strumento strategico è la **Skill Matrix**, una tabella che mappa le competenze, le conoscenze e gli interessi dei membri del team in relazione ai task previsti. Essa aiuta a migliorare l'assegnazione dei ruoli, rende evidenti le **lacune di competenza** da colmare attraverso formazione mirata, e stimola la **motivazione individuale**. Le qualifiche possono essere:

- **Primary skill**: competenza principale, con possibilità di assumere ruoli guida.
- **Secondary skill**: partecipazione come supporto tecnico.
- **Interest**: interesse espresso per un'area senza competenza ancora acquisita.

La combinazione di questi strumenti consente di rendere il progetto **prevedibile, monitorabile e adattabile**. La scomposizione delle attività favorisce la **chiarezza operativa**, la modellazione delle dipendenze consente un **controllo temporale accurato**, e la mappatura delle competenze garantisce la costruzione di **team efficaci**. Inoltre, la documentazione tramite work packages assicura **tracciabilità e responsabilità**. In definitiva, l'obiettivo è disporre di un progetto in cui il management sia in grado di pianificare con precisione, rispondere ai cambiamenti e guidare il team verso una consegna di successo.

6. Il Software Project Management Plan - SPMP

Il **Software Project Management Plan (SPMP)** è il documento centrale della gestione di progetto. Segue le linee guida dello **standard IEEE 1058** e rappresenta una guida condivisa tra il cliente, il project manager e il team di sviluppo. Il suo valore è duplice: da un lato offre una **visione d'insieme sistematica** delle attività previste, dall'altro costituisce una **base contrattuale e metodologica** per il governo dell'intero ciclo di vita del progetto.

Il suo scopo è:

- **Formalizzare gli obiettivi e le modalità operative** del progetto, definendo con precisione cosa si vuole ottenere e come si intende procedere.
- **Definire l'organizzazione e i ruoli**, chiarendo la responsabilità di ciascun partecipante e assicurando un'adeguata distribuzione dei compiti.
- **Descrivere i processi gestionali e tecnici** che saranno adottati, garantendo coerenza, standardizzazione e tracciabilità.
- **Fornire una base di riferimento per il monitoraggio**, utile per verificare lo stato di avanzamento e supportare la presa di decisioni.
- **Supportare la comunicazione e la trasparenza**, sia all'interno del team sia nei confronti degli stakeholder esterni.

La struttura del documento comprende:

1. **Overview**: sintesi del progetto, dei suoi obiettivi principali, dei vincoli noti (es. budget, tempo, qualità), e delle versioni precedenti dello SPMP. È il punto d'ingresso per chiunque debba consultare il piano.
2. **References**: raccolta delle fonti normative, linee guida aziendali, documenti tecnici e contratti rilevanti. Questo facilita la coerenza tra il piano e il contesto normativo e tecnico in cui si opera.
3. **Definitions**: glossario preciso di termini, sigle e concetti utilizzati nel documento, per ridurre ambiguità e garantire una comprensione condivisa.
4. **Project Organization**: descrizione dettagliata dell'organizzazione interna del team di progetto. Include organigrammi, ruoli e responsabilità, interfacce tra i team, nonché canali ufficiali di comunicazione. In questa sezione si possono anche esplicitare i meccanismi di escalation e le figure responsabili del coordinamento.

5. **Managerial Process Plans**:
   - **Start-up plan**: fasi iniziali, requisiti infrastrutturali, risorse da attivare e condizioni necessarie per il kickoff.
   - **Work plan**: calendario delle attività, milestone, dipendenze e allocazione temporale delle risorse.
   - **Control plan**: modalità con cui si misurerà l'avanzamento (es. KPI, burndown chart, milestone review), e frequenza dei controlli.
   - **Risk management plan**: processo di identificazione, valutazione e risposta ai rischi; comprende anche il registro dei rischi e le contromisure previste.
   - **Closeout plan**: procedure di chiusura del progetto, compresa la raccolta dei deliverables, il rilascio delle risorse e l'analisi post-progetto.
6. **Technical Process Plans**:
   - **Modello di processo**: struttura metodologica adottata (ad esempio agile con sprint, oppure a cascata con fasi sequenziali).
   - **Strumenti, tecniche e tecnologie**: dettaglio degli strumenti di sviluppo, testing, documentazione e deployment.
   - **Infrastruttura tecnica**: ambienti hardware e software utilizzati, pipeline CI/CD, sistemi di controllo versione, ambienti di staging.
   - **Criteri di accettazione**: condizioni da soddisfare affinché il prodotto venga considerato completato e approvato dal cliente.
7. **Supporting Process Plans**:
   - **Configuration management**: gestione delle versioni, branch strategy, tracciamento delle modifiche.
   - **Verification and validation**: approcci alla verifica (review, walkthrough) e alla validazione (testing, prototipi).
   - **Documentation**: piano per la produzione, revisione e distribuzione della documentazione tecnica e utente.
   - **Quality assurance**: tecniche e standard per garantire la qualità del prodotto e del processo.
   - **Review e audit**: frequenza e modalità delle revisioni periodiche e delle verifiche di conformità.
   - **Gestione dei fornitori**: se presenti, modalità di interazione, contratto, controllo qualità e milestone.

- **Miglioramento dei processi**: misurazione delle performance e piani di ottimizzazione continua.

Lo SPMP è un documento **vivo**: viene aggiornato man mano che il progetto evolve. Una buona pratica è gestirlo con strumenti di versionamento collaborativo (es. Git, SharePoint), rendendolo sempre accessibile e aggiornato. Ogni modifica significativa dovrebbe essere tracciata e comunicata a tutte le parti interessate. Il valore dello SPMP emerge in particolare nella **gestione del cambiamento**, dove rappresenta un punto di riferimento stabile, nel **confronto con il cliente**, dove assume valore legale e contrattuale, e nella **fase post-mortem**, in cui funge da base per la valutazione retrospettiva dell'intero progetto.

7. Conclusioni e sintesi

La **gestione dei progetti software** è un'attività articolata che richiede una combinazione di competenze tecniche, manageriali e relazionali. Non si tratta solo di rispettare tempi e budget, ma di creare le condizioni per cui le persone coinvolte possano esprimere il massimo potenziale all'interno di un sistema organizzato, collaborativo e orientato al risultato. È un processo che richiede lucidità analitica, empatia e una visione strategica. Saper interpretare correttamente le esigenze del cliente, definire obiettivi misurabili, scegliere le tecnologie adeguate, costruire team motivati e reagire con prontezza ai cambiamenti è parte integrante del mestiere.

In questo documento abbiamo esplorato:

- Il ruolo centrale del project manager come facilitatore e coordinatore, figura chiave per il successo dell'intero progetto.
- La struttura e l'interdipendenza tra outcome, lavoro, pianificazione e risorse, che richiedono una gestione olistica e interconnessa.
- Le sfide legate alla complessità e al cambiamento, e le strategie adottabili per affrontarle in modo efficace.
- Le fasi dinamiche di un progetto software e le attività chiave associate, che definiscono il ritmo e la direzione del lavoro.
- I concetti generali di task, attività, deliverable e work package, fondamentali per strutturare il lavoro in modo controllabile.
- Il contenuto e l'importanza strategica del SPMP, documento guida e pilastro della governance di progetto.

Un progetto ben gestito si riconosce non solo dal prodotto finale, ma dalla **qualità del processo** con cui è stato realizzato. Un processo ben impostato consente di minimizzare gli sprechi, prevenire i conflitti, massimizzare il valore prodotto e promuovere la crescita professionale del team. L'utilizzo di strumenti formali, la comunicazione trasparente, la documentazione accurata, il coinvolgimento continuo degli stakeholder e la gestione proattiva dei rischi costituiscono i pilastri di un approccio professionale al Software Project Management.

Gestire un progetto software significa orchestrare persone, competenze, strumenti e obiettivi per costruire un valore condiviso. È un esercizio di equilibrio tra pianificazione e flessibilità, tra visione e operatività, tra metodo e innovazione. È anche, e soprattutto, un processo umano, in cui la capacità di

ascoltare, adattarsi e guidare fa la differenza tra un semplice progetto completato e un risultato realmente trasformativo per l'organizzazione.

Bibliografia

- Bruegge, B., & Dutoit, A. H. (2010). Object-oriented software engineering. Using UML.