Attenzione

Il file non ha alcuna pretesa di correttezza. Per facilitare il processo di scrittura delle risposte, si è usato anche ChatGPT, parte di risposte presenti, slide e vari fonti di appunti.

Si ringrazi comunque per quanto presente, direi, ogni tanto.

Scritto da Gabriel

Domanda (punti 8/30)

Lo sviluppo incrementale e quello agile entrambe suddividono il tempo (lungo) di progetto in periodi (più corti) di avanzamento. Questo quesito vi chiede di discutere le differenze (o le somiglianze) che individuate nel modo in cui quei due modelli interpretano l'articolazione di tali periodi, per determinazione della loro durata, l'individuazione dei loro specifici obiettivi, le modalità di gestione (cioè pianificazione, esecuzione, e verifica). Fornirete la vostra risposta in modo strutturato, descrivendo esplicitamente:

- A quale parte di domanda state rispondendo.
- Quale sia la vostra risposta a essa.
- Sulla base di cosa lo avete pensato.

Risposta

Determinazione della durata del periodo:

- Sviluppo incrementale: I periodi, o incrementi, sono tipicamente determinati dalla dimensione e dalla complessità dei compiti da completare al loro interno. La durata di ciascun incremento può variare, ma in genere è più breve rispetto allo sviluppo agile. Sono articolati in periodi di durata fissa.

- Sviluppo agile: I periodi, o sprint, sono tipicamente impostati su una durata variabile, impostata tra 1 e 4 settimane. Ciò consente una cadenza costante dei progressi e permette al team di pianificare e impegnarsi a realizzare una serie di risultati entro quel lasso di tempo.

Identificazione di obiettivi specifici:

- Sviluppo incrementale: Gli obiettivi specifici per ogni incremento sono determinati in base agli obiettivi generali del progetto e alle priorità delle parti interessate. Questi obiettivi devono essere specifici, misurabili, raggiungibili, pertinenti e limitati nel tempo (SMART). Essi sono definiti in anticipo, dato che non ammette devianze dall'obiettivo; produrrà un risultato certo

- Sviluppo agile: Gli obiettivi specifici per ogni sprint sono determinati dal team in base agli obiettivi e alle priorità generali del progetto. Questi obiettivi sono catturati nello sprint goal, che è una dichiarazione di alto livello di ciò che il team spera di raggiungere nello sprint. In particolare, ogni obiettivo è dinamico, cambiando in base ad esigenze ed opportunità di progetto.

Gestione:

- Sviluppo incrementale: La pianificazione comporta la determinazione degli obiettivi e dei compiti specifici per ogni incremento in modo rigido, nonché l'identificazione di eventuali dipendenze o rischi. L'esecuzione comporta l'effettivo completamento dei compiti all'interno dell'incremento, che può comportare la collaborazione con altri team o stakeholder. La verifica comporta l'esame dei progressi e dei risultati dell'incremento per assicurarsi che stia raggiungendo gli obiettivi desiderati.

- Sviluppo agile: Anche la pianificazione, l'esecuzione e la verifica sono aspetti importanti dello sviluppo agile. La pianificazione comporta la determinazione degli obiettivi e dei compiti specifici per ogni sprint, nonché l'identificazione di eventuali dipendenze o rischi. L'esecuzione comporta il completamento dei compiti all'interno dello sprint, che può comportare la collaborazione con altri membri del team o con le parti interessate. La verifica comporta l'esame dei progressi e dei risultati dello sprint specifico. Una differenza fondamentale tra lo sviluppo incrementale e lo sviluppo agile

Scritto da Gabriel

in termini di gestione è l'uso di cerimonie agili, come stand-up giornalieri, pianificazione dello sprint e revisione dello sprint, per facilitare la comunicazione e la collaborazione tra i membri del team.

Domanda (punti 8/30)

Vogliamo ragionare sulla relazione che intercorre tra i concetti di “processo” e di “progetto”, facendo riferimento a situazioni concrete, direttamente sperimentate. Nello specifico, vogliamo capire se (1) un progetto, ovvero l’insieme di attività da esso mobilizzate, possa essere descritto, spiegato, compreso, in termini di una qualche articolazione organizzata di processi, e se (2), conseguentemente, la definizione dei secondi debba o meno preesistere all’organizzazione del primo. Per rispondere alle due domande, scegliete uno tra due approcci: (a) analizzare un dato segmento del vostro progetto didattico, individuando le attività previste in esso, e attribuendo ciascuna di esse a un processo di appartenenza, oppure (b) “dispiegare” alcuni processi per voi essenziali e “mapparli” su un tratto del vostro progetto, scendendo in entrambi i casi a livello di singole attività elementari della dimensione dei vostri normali “ticket”. La risposta dovrà anche spiegare come (3) le relazioni di precedenza intercorrenti tra specifiche attività di progetto si riflettano all’interno dei processi di appartenenza e nelle relazioni tra di essi.

Risposta

Per rispondere a queste domande, adotterò l'approccio (a) e analizzerò un determinato segmento di un progetto didattico, identificando le attività in esso previste e attribuendo ciascuna di esse a un processo a cui appartiene. Sappiamo infatti che i processi raggruppano e codificano una serie di attività, trasformando ingressi in uscite.

All’interno del progetto didattico, possiamo individuare una serie di attività:

- Analisi dei Requisiti: Ricerca dell'applicazione software e raccolta di informazioni sulle sue caratteristiche e funzionalità. Questa attività rientra nel processo di pianificazione e preparazione, cercando di comprendere informazioni sul pubblico target e sui requisiti del corso. Questa è un processo primario, in cui si discute dell’acquisizione dei requisiti e si struttura la fase di fornitura
- Documentazione: Processo di supporto, il quale pone le basi per ogni cosa nel sistema (configurazione delle parti, implementazione ad alto livello di calendario, parti, test, design)
- Design: progettazione del contenuto, una volta compresi a fondo i requisiti e come mettere insieme le singole parti
- Test e Verifica: un processo che accompagna lo sviluppo software, portando alla creazione di contenuti e al creare cose che sbagliano per aiutare l’apprendimento progressivo. In questa fase, si cerca sempre di fare in modo che tutto funzioni, intervenendo a vario livello sulle parti del sistema in modo opportuno
- Validazione, verificando la correttezza di quanto fatto fino ad ora ed effettuando in una fase successiva il collaudo e la distribuzione

Le relazioni di precedenza tra queste attività specifiche del progetto si riflettono nei processi a cui appartengono. Di fatto è cosa fondamentale individuare la priorità di processi e le singole attività, sapendo quando progettazione, implementazione, raccolta dei requisiti ed altre attività possano avere luogo ed implementazione.

In conclusione, un progetto può essere descritto, spiegato e compreso in termini di un'articolazione organizzata di processi. La definizione dei processi deve preesistere all'organizzazione del progetto per guidare le attività e garantire che siano completate nell'ordine corretto. Le relazioni di precedenza tra le attività specifiche del progetto si riflettono all'interno e nelle relazioni tra i processi a cui appartengono. I processi sono singole attività svolte all’interno del progetto che trasformano ingressi (bisogni) in uscite (prodotti) secondo regole date, consumando risorse nel farlo, mentre il progetto è definito uniformemente come insieme di processi, secondo degli ingressi predeterminati e uscite determinate dall’implementazione (design).

Scritto da Gabriel

Domanda (punti 8/30)

Per alcuni, la tupla <decisione, azione, verifica> costituisce l'unità base, tassello atomico (unitario e inscindibile), dei metodi di pianificazione, gestione, ed esecuzione di progetto.

Riflettendo sulla vostra esperienza pratica sin qui e su quanto ritenete di avere appreso dallo studio teorico:

- definite singolarmente ogni elemento di tale tupla, in termini pertinenti ai metodi di gestione di progetto che conoscete;
- prendete posizione sull'asserzione di cui sopra, motivandola in modo argomentato, anche attraverso esempi concreti, che la confermino o la confutino.

Risposta

Gli elementi della tupla <decisione, azione, verifica> nel contesto della gestione del progetto possono essere definiti come segue:

- Decisione: Una decisione è una scelta fatta tra un insieme di alternative sulla base delle informazioni e dei vincoli disponibili. Nella gestione dei progetti, le decisioni sono tipicamente prese dal project manager o da un'autorità decisionale designata (ad esempio il responsabile) e possono riguardare vari aspetti del progetto, come l'ambito, la tempistica, il budget, la gestione del rischio e il coinvolgimento degli stakeholder.
- Azione: L'azione si riferisce all'attuazione di una decisione. Nella gestione dei progetti, le azioni sono tipicamente intraprese dal team di progetto o dai singoli membri del team per eseguire i compiti necessari a raggiungere gli obiettivi del progetto.
- Verifica: La verifica è il processo di controllo della coerenza delle azioni intraprese con la decisione presa. Nella gestione dei progetti, la verifica può comportare l'esame del lavoro completato, il confronto con il piano di progetto e la garanzia che sia conforme agli standard di qualità specificati.

Per quanto riguarda l'affermazione che la tupla <decisione, azione, verifica> è l'unità di base dei metodi di pianificazione, gestione ed esecuzione dei progetti, direi che è generalmente vera, ma con alcune riserve.

Da un lato, il ciclo <decisione, azione, verifica> è un aspetto fondamentale di molti approcci alla gestione dei progetti, comprese le metodologie tradizionali come Waterfall/a cascata e quelle agili come Scrum. In questi approcci, si prendono decisioni sul lavoro da svolgere, si intraprendono azioni per completare il lavoro e si verificano i risultati per garantire che il progetto sia in linea con i suoi obiettivi; tutto questo su una base piccola e continuativa, possibilmente migliorante.

D'altra parte, è importante notare che il ciclo <decisione, azione, verifica> non è l'unico elemento della gestione del progetto. Ci sono molti altri fattori che contribuiscono al successo o al fallimento di un progetto, come il coinvolgimento degli stakeholder, la gestione del rischio, la comunicazione e l'allocazione delle risorse. Questi fattori sono spesso interdipendenti con il ciclo di decisione, azione e verifica e devono essere presi in considerazione per pianificare, gestire ed eseguire efficacemente un progetto.

A titolo di esempio, si consideri un progetto per lo sviluppo di una nuova applicazione software. Il project manager potrebbe decidere di utilizzare un certo linguaggio di programmazione e un certo framework di sviluppo, e il team potrebbe agire per implementare l'applicazione utilizzando questi strumenti. Tuttavia, se il project manager non si è impegnato adeguatamente con le parti interessate e non ha raccolto i loro input sui requisiti dell'applicazione, il prodotto finale potrebbe non soddisfare le loro esigenze e il processo di verifica lo rivelerà. In questo caso, il ciclo decisione, azione, verifica non è sufficiente da solo a garantire il successo del progetto; per essere efficace, deve essere integrato con altri fattori, come il coinvolgimento degli stakeholder.

Scritto da Gabriel

Domanda (punti 8/30)

Il termine *coverage* ricorre sovente, con diverse accezioni, sfumature e ramificazioni, nel dominio dell'ingegneria del software. Questa prova d'esame vi chiede di elencare tutte le accezioni di quel termine che avete effettivamente incontrato nella vostra esperienza sin qui, come concetto solo udito oppure anche praticato. Di ognuna di esse spiegherete precisamente il significato, illustrando anche le modalità di determinazione del valore quantitativo corrispondente.

Risposta

Il termine coverage indica letteralmente "copertura" e possiamo usarlo in vari modi:

- Code coverage: La copertura del codice si riferisce alla misura in cui una determinata suite di test esercita le varie linee di codice di un'applicazione software. In genere viene misurata in percentuale e può essere determinata eseguendo la suite di test e registrando quali linee di codice sono state eseguite.
- Test coverage: La copertura dei test si riferisce alla misura in cui una determinata suite di test esercita le varie caratteristiche o funzionalità di un'applicazione software. Ciò può includere il test di diversi valori di input, il test delle condizioni di errore, il test delle prestazioni, ecc. La copertura dei test è tipicamente misurata in termini di percentuale di caratteristiche o funzionalità testate.
- Statement/Condition coverage: La copertura delle dichiarazioni è un tipo di copertura del codice che misura la percentuale di dichiarazioni in un'applicazione software che sono state eseguite durante il test. È una misura semplice di copertura che non tiene conto della complessità o dell'importanza delle istruzioni eseguite.
- Branch coverage: La copertura delle diramazioni è un tipo di copertura del codice che misura la percentuale di diramazioni in un'applicazione software che sono state esercitate durante i test. Un ramo è un punto del codice in cui il flusso di esecuzione può andare in più di una direzione, in base all'esito di un'istruzione condizionale.
- Function coverage: La copertura delle funzioni è un tipo di copertura del codice che misura la percentuale di funzioni di un'applicazione software che sono state eseguite durante i test. Una funzione è un blocco di codice autonomo che esegue un compito specifico.
- Path coverage: La copertura del percorso è un tipo di copertura del codice che misura la percentuale di possibili percorsi attraverso un'applicazione software che sono stati esercitati durante il test. Un percorso è una sequenza di istruzioni e rami che possono essere eseguiti in un determinato ordine.
- Integrated test coverage: La copertura del test integrato si riferisce alla misura in cui una suite di test esercita i vari componenti o moduli di un'applicazione software quando sono integrati e funzionano insieme. Si può misurare in termini di copertura del codice e di copertura dei test ottenuti tra i vari componenti o moduli.
- Copertura del test di sistema: La copertura del test di sistema si riferisce alla misura in cui una suite di test esercita la funzionalità e le prestazioni complessive di un'applicazione software quando viene distribuita in un ambiente simile alla produzione. Ciò può essere misurato in termini di copertura del codice e di copertura dei test raggiunti tra i vari componenti o moduli, nonché di prestazioni e affidabilità dell'applicazione nel suo complesso.

Domanda (punti 8/30)

Il cosiddetto “ciclo di Deming” descrive una collaudata pratica a supporto del miglioramento continuo, pensata per essere applicata in qualunque ambito di attività. Per testare tale applicabilità, vogliamo studiare come calarla in uno specifico e limitato insieme di attività di lavoro di uno studente di informatica. La risposta dovrà avere quattro parti: (1) una breve descrizione generale del ciclo di Deming che ne colga l’essenza e non sia un copia-incolla di fonte esterna; (2) la specifica dello scenario di interesse: quali attività di lavoro e quale scenario vogliate considerare; (3) la corrispondenza da voi proposta tra la descrizione al punto precedente e lo specifico scenario di interesse; (4) una concreta procedura di applicazione, redatta normativamente, come parte di un ideale *way of working*.

Risposta

(1) Il ciclo di Deming, noto anche come ciclo PDCA (Plan-Do-Check-Act), è una metodologia di miglioramento continuo ideata da Deming e Shewhart (per questo ha anche come nome Deming and Shewhart Cycle) basato sul continuo problem-solving, quasi come un metodo scientifico. Esso prevede quattro fasi:

- Pianificare: Identificare un problema o un'opportunità di miglioramento e sviluppare un piano per affrontarlo, identificando relazioni causa-effetto, relazioni temporali e dati temporali
- Fare: Attuare il piano e raccogliere dati sulla sua efficacia.
- Controllare: Analizzare i dati e valutare i risultati dell'implementazione, comparando quanto si distanza dalle aspettative, imparando cose nuove e agire dopo la revisione
- Agire: Apportare modifiche in base ai risultati della valutazione, cercando di pensare a cosa fare dopo e continuare il ciclo aggiustando e migliorando.

(2) Lo scenario di interesse è quello di uno studente di informatica che lavora a un progetto di programmazione per un corso. Le attività specifiche di questo scenario comprendono la scrittura di codice, il debug e il test.

(3) Nel contesto del progetto di programmazione, il ciclo di Deming può essere applicato come segue:

- Pianificazione: Identificare le aree del codice che necessitano di miglioramenti o le aree in cui si verificano bug. Sviluppate un piano per affrontare questi problemi, ad esempio rifattorizzando il codice o aggiungendo ulteriori test.
- Eseguire: Attuare il piano scrivendo o modificando il codice ed eseguendo i test.
- Controllare: Analizzare i risultati dei test e valutare l'efficacia delle modifiche apportate al codice.
- Agire: Apportare modifiche in base alla valutazione, come la correzione di eventuali bug residui o l'ulteriore ottimizzazione del codice.

(4) Una procedura concreta e normata per l'applicazione del ciclo di Deming come parte di un metodo di lavoro ideale potrebbe includere le seguenti fasi:

- Identificare un problema specifico o un'opportunità di miglioramento relativa al progetto di programmazione.
- Sviluppare un piano per affrontare il problema o l'opportunità, compresi obiettivi e traguardi specifici.
- Attuare il piano scrivendo o modificando il codice ed eseguendo dei test.
- Analizzare i risultati dei test e valutare l'efficacia delle modifiche apportate al codice.
- Apportate modifiche in base alla valutazione e continuate il ciclo fino a quando il problema o l'opportunità di miglioramento non sono stati affrontati in modo soddisfacente.

Scritto da Gabriel

- Documentate i risultati del ciclo, comprese le modifiche apportate al codice, i risultati dei test e le lezioni apprese.
- Ripetete il ciclo per qualsiasi altro problema o opportunità di miglioramento identificati.

Domanda (punti 8/30)

Sani principi di gestione di progetto (project management) consigliano l'uso del consuntivo di periodo come strumento essenziale per misurare il grado di avanzamento e valutare le corrispondenti prospettive di completamento. Per l'efficace attuazione di tale pratica serve dotarsi di una interpretazione "intelligente" (non superficiale) del concetto di consuntivo di periodo. Questa prova d'esame vi chiede di: (1) fornire una definizione di "consuntivo di periodo" sia come nozione in sé che in termini dei dati che esso racchiude e della specifica funzione informativa di ciascuno di essi; (2) spiegare quali indicazioni possano essere tratte da un consuntivo siffatto; (3) suggerire una metodica utile ad alimentare la raccolta dati alla base di un consuntivo di periodo, rendendola veloce e affidabile.

Risposta

1) Un consuntivo di periodo è uno strumento di project management che permette di valutare per certo lo stato di avanzamento del progetto e delle sue attività, consentendo di avere delle prospettive di completamento utili sia per pianificare le attività attuali che quelle nel medio/lungo periodo. Include informazioni sulle risorse, tempo, persone utilizzate e budget utilizzati. Ogni informazione viene utilizzata, a livello documentale e pratico, per adattare le attività del modello che si sta seguendo e decidere l'andamento delle singole fasi presenti e future, decidendo come pianificare e prende decisioni in futuro.

2) Le informazioni che si possono ricavare dai rendiconti di periodo includono:

- Prestazioni economiche/di risorse: I rendiconti di periodo possono aiutare i project manager a monitorare le prestazioni finanziarie di un progetto, compresi i ricavi, le spese e i profitti.
- Pianificazione temporale e lavorativa: come è stato impiegato il tempo di calendario per lo svolgimento delle attività utili, vedendo quali persone sono state utilizzate e come, vedendo come migliorare possibilmente queste statistiche

3) Una metodologia utile per raccogliere i dati alla base di un rendiconto di periodo comprende:

- Determinare il periodo di tempo per il rendiconto.
- Identificare le fonti di dati che verranno utilizzate per creare il rendiconto, come ad esempio i registri finanziari, le fatture e le ricevute.
- Raccogliere i dati dalle fonti identificate.
- Organizzate i dati nelle categorie appropriate per il rendiconto, come entrate, uscite, attività e passività.
- Utilizzare una checklist delle attività da controllare e riportare in modo aggiornato secondo uno strumento possibilmente automatizzabile costi, scadenze, responsabilità, etc.
- Esaminare il rendiconto periodico per verificarne l'accuratezza e la completezza.
- Aggiornare regolarmente il rendiconto periodico per garantire che rifletta le informazioni più aggiornate.

Tutto ciò può essere fatto con un software di gestione dei progetti che consente di registrare e monitorare l'avanzamento del progetto in tempo reale. In questo modo, i dati sono sempre aggiornati e disponibili per essere inclusi nel consuntivo di periodo. Inoltre, è importante che i membri del progetto siano istruiti su come registrare correttamente i dati e che ci sia un processo stabilito per la revisione e l'approvazione dei dati prima che vengano inclusi nel consuntivo di periodo.

Scritto da Gabriel

Domanda (punti 8/30)

Nel mondo SWE, vi è chi sostiene che il modello di sviluppo ispirato alla “continuous integration” sia in contrasto, e quindi incompatibile con la salvaguardia dei principi ispiratori del “modello a V”. Altri voci di quel mondo sostengono invece che i due modelli possano bene coesistere e contaminarsi a vicenda. Per rispondere a questo quesito: (1) fornite una definizione, concisa e ragionata di ciascuno dei due modelli, separatamente, ponendo l’accendo sui loro rispettivi obiettivi e le corrispondenti precondizioni di attuazione; (2) prendete posizione sul punto in questione, argomentando la vostra conclusione; (3) in caso affermiate compatibilità, illustrate un modello di sviluppo che ne combini i tratti; in caso contrario, suggerite la scelta a vostro avviso migliore (più conveniente, per rapporto costi/benefici) tra i due modelli.

Risposta

1) Il modello di integrazione continua (CI) è una pratica di sviluppo del software che prevede l'integrazione regolare delle modifiche al codice in un repository condiviso e la costruzione e il collaudo automatico della base di codice risultante per identificare e risolvere i problemi nelle prime fasi del processo di sviluppo. Questo aiuta ad integrare continuativamente piccoli pezzi che funzionano sempre. Gli obiettivi principali della CI sono la riduzione del tempo e dello sforzo necessario per integrare le modifiche al codice, il miglioramento della qualità della base di codice e l'aumento della velocità di sviluppo. I prerequisiti per l'implementazione del CI includono la disponibilità di un repository condiviso, processi di compilazione e test automatizzati e un team che si impegna a integrare e testare regolarmente le modifiche al codice.

Il modello a V è un modello di sviluppo del software che prevede una sequenza lineare di attività, in cui ogni attività rappresenta una fase del processo di sviluppo. Il V Model è spesso utilizzato nel contesto dell'ingegneria dei sistemi e sottolinea l'importanza di verificare e validare il sistema in ogni fase dello sviluppo, garantendo un collegamento sicuro tra tutti i punti del processo di progetto, partendo dai requisiti ed arrivando alla validazione. Gli obiettivi principali del modello V sono garantire che il sistema soddisfi i requisiti specificati e identificare e correggere i difetti il più presto possibile nel processo di sviluppo. I prerequisiti per l'implementazione del modello V includono la presenza di un insieme chiaro di requisiti e di un processo di sviluppo strutturato che segue una sequenza lineare di attività.

2) Ritengo che il modello CI e il V model siano compatibili e possano coesistere e contaminarsi a vicenda. Mentre il V Model enfatizza l'importanza di verificare e validare il sistema in ogni fase dello sviluppo in modo bidirezionale, il modello CI sottolinea l'importanza di integrare e testare regolarmente le modifiche al codice per migliorare la qualità della base di codice. Combinando i principi di entrambi i modelli, è possibile creare un processo di sviluppo che combini i vantaggi di entrambi gli approcci. In questo modo, in qualsiasi processo di sviluppo, ogni fase ha certezza di essere costruita con pezzi validi e funzionanti.

3) Un modello di sviluppo che combina i tratti del modello di sviluppo "continuous integration" e del modello di sviluppo "a V" potrebbe essere un modello di sviluppo "agile". Il modello agile combina i principi dell'integrazione continua con quelli della pianificazione e della verifica delle specifiche del progetto. In questo modello, i team di sviluppo lavorano in modo continuo per integrare il codice e eseguire test automatizzati, ma allo stesso tempo, si concentrano sulla pianificazione e sulla verifica delle specifiche del progetto per garantire che il progetto soddisfi i requisiti del cliente e a ciò che desidera.

Il modello agile è una scelta conveniente per il rapporto costi/benefici poiché combina i vantaggi dei due modelli, permettendo di rilevare e correggere i problemi di integrazione il più presto possibile e di garantire che il progetto soddisfi i requisiti del cliente. Il modello agile prevede una stretta collaborazione tra i membri del team, una maggiore flessibilità nell'adattamento ai cambiamenti e una maggiore efficienza nell'utilizzo delle risorse.

Scritto da Gabriel

Domanda (punti 8/30)

Nel colloquio di assunzione presso l'organizzazione dei vostri sogni, il vostro interlocutore, che ha un curriculum professionale importante, vi chiede di fornire linee guida operative per l'utilizzo delle nozioni di baseline e di milestone nella pianificazione di progetto, spiegando anche se e come tali nozioni siano legate tra loro. Il vostro interlocutore desidera risposte chiare, nella loro formulazione e nei concetti che essa utilizza, e concrete, non limitate alle definizioni "da libro", ma pronte all'uso in contesti reali, possibilmente anche personalmente sperimentate.

Risposta

Le linee guida operative per l'utilizzo delle nozioni di baseline e di milestone nella pianificazione di progetto sono le seguenti:

- Baseline: una baseline è un punto di riferimento stabilito per un progetto, che rappresenta lo stato attuale del progetto o gli obiettivi stabiliti per il progetto. È utilizzato come punto di partenza per la valutazione del progresso del progetto e per la determinazione dei cambiamenti necessari per raggiungere gli obiettivi del progetto. Nella pianificazione di un progetto, è importante stabilire una baseline per gli obiettivi, il budget, il programma e le risorse del progetto. Concretamente, un insieme di milestone viene sostanziata da una baseline, che si pone come base di avanzamento per obiettivi raggiunti
- Milestone: una milestone è un evento o un traguardo significativo che indica il completamento di una fase o di un'attività del progetto. Le milestone sono utilizzate per segnalare l'avanzamento del progetto e per determinare se il progetto è in linea con il programma stabilito. Nella pianificazione di un progetto, è importante identificare le milestone chiave e includerle nel programma del progetto per monitorare l'avanzamento del progetto.

Le baseline e le milestone sono collegate tra loro, in quanto sono entrambe utilizzate per misurare i progressi e garantire che un progetto rimanga in linea con i tempi. La baseline serve come punto di riferimento per l'intero progetto, mentre le milestone segnano punti specifici di avanzamento all'interno del progetto. Infatti, normalmente, una baseline serve a sostanziare le milestone individuate. Il loro numero varia in base agli obiettivi preposti, agli accordi presi con gli stakeholder e quanto si vuole realizzare nel periodo.

Per utilizzare efficacemente le baseline e le milestone nella pianificazione del progetto, è importante:

- Definire chiaramente la linea di base del progetto all'inizio, includendo tutti i criteri rilevanti come l'ambito, il calendario, il budget e la qualità.
- Identificare le tappe fondamentali che segnano il completamento di fasi o compiti importanti del progetto.
- Misurare regolarmente i progressi rispetto alla linea di base e alle tappe fondamentali per garantire che il progetto sia in linea con i tempi.
- Apportare le modifiche necessarie per rimanere in linea con i tempi, come ad esempio l'adeguamento del calendario o del budget.
- Comunicare i progressi e gli eventuali aggiustamenti necessari alle parti interessate, tra cui il team di progetto, le parti interessate e la direzione.

Risposta data e presente su MEGA:

Per garantire uno svolgimento efficace ed efficiente del progetto è necessaria una pianificazione precisa e realistica, che tenga conto delle richieste del committente ma anche delle risorse a disposizione del team e delle capacità e disponibilità dei singoli elementi. A tale scopo, vanno innanzitutto concordate le date di inizio e fine dell'attività di progetto.

Tuttavia, ponendo come unico limite temporale la fine del progetto, il rischio di non riuscire a monitorare opportunamente il consumo delle risorse è alto; questo causerebbe ritardi rispetto al piano concordato col committente e, nel peggiore dei casi, potrebbe compromettere l'intera riuscita del progetto. È quindi opportuno fissare, sin dall'inizio, delle tappe intermedie, le milestone, e associarvi obiettivi di sviluppo misurabili, ben definiti ed equamente distribuiti nel tempo.

Raggiunte tali date, di volta in volta il confronto tra gli obiettivi pianificati e quelli effettivamente realizzati darà al team la consapevolezza dello stato di avanzamento del progetto; ciò permetterà di effettuare ripianificazioni o riassegnazioni dei compiti secondo necessità. Il numero di milestone da fissare dipende dall'esperienza dei membri del gruppo. In caso di team giovani è preferibile stabilirne molte, a frequenza alta (circa una settimana di distanza l'una dall'altra), per mitigare il rischio di grandi deviazioni da quanto pianificato.

Gruppi senior possono porsi milestone più diradate nel tempo, poiché la loro capacità di gestione delle risorse e di autovalutazione dell'operato, dovute all'esperienza, scongiura i rischi di eccessivi scostamenti dalla pianificazione iniziale. In generale, il numero di tappe intermedie va sempre bilanciato con gli obiettivi del progetto: accumularne troppi su poche milestone lontane nel tempo risulta sempre azzardato. Se la milestone è un punto nel tempo, infatti, ad essa è sempre associato un prodotto (software) ad un preciso stadio di avanzamento, determinato dagli obiettivi di sviluppo associati alla milestone.

L'insieme di funzionalità implementate nel prodotto al raggiungimento di una milestone è una baseline. Per tracciare lo storico di una baseline va adottato un sistema di versionamento: ciò permette di risalire alle versioni intermedie e monitorare lo sviluppo nel tempo. Adottando un approccio incrementale allo sviluppo, il concetto di baseline assume tratti più specifici: è il punto di arrivo di un incremento e la base di partenza per il successivo (si presuppone che il periodo associato a ciascun incremento sia delimitato da milestone).

Durante un incremento, ogni baseline si costruisce dalla precedente per aggiunte successive, e il suo sviluppo concorre incrementalmente alla maturazione del prodotto finale. Si intende la baseline come "base di avanzamento" dello sviluppo, prototipo funzionante da cui non dovrebbe più essere necessario retrocedere, poiché tutte le funzionalità implementate sono state verificate, hanno superato i controlli di qualità e possono considerarsi stabili, ma a cui si può tornare in caso di sviluppi futuri non funzionanti. Nella realtà, un'eccessiva rigidità nella concezione di baseline potrebbe essere dannosa, perché potrebbe essere necessario rivedere alcune parti del prodotto precedentemente assemblate, ad esempio per errori nella verifica o incomprensioni col committente; queste condizioni sono indesiderabili e quindi da evitare, ma vanno tenute in considerazione come possibili.

Il numero di incrementi (quindi di milestone) andrebbe concordato col committente, come anche i relativi obiettivi di sviluppo, in modo da implementare per prime le funzionalità fondamentali. Così facendo, il committente avrà presto a disposizione un prototipo funzionante che offra le funzionalità che più gli premono. Inoltre, essendo implementate per prime, esse saranno le più testate e verificate, e di conseguenza le più stabili.

Nell'attività di progetto svolta in ambito universitario abbiamo sperimentato i vantaggi operativi di questo approccio alla pianificazione e allo sviluppo software. In particolare, essi sono risultati evidenti se

Scritto da Gabriel

confrontati coi problemi derivati da una pianificazione scandita da milestone troppo distanziate nel tempo e associate ad obiettivi di sviluppo imprecisi, o addirittura tanto ampi da non dare alcun obiettivo intermedio ma sancire solo il termine dello sviluppo.

Simili pianificazioni hanno portato a scarsa cognizione dello stato di avanzamento del prodotto, eccesso di ore di lavoro dedicate ad un'attività, conseguente carenza di tempo per gli altri compiti, ritardi nello sviluppo. Ciò si è riflesso anche nelle valutazioni di consuntivo delle singole fasi: erano rilevati scostamenti considerevoli dai preventivi, con necessità di grandi ridistribuzioni delle risorse dovute a tarda presa di consapevolezza dei ritardi accumulati. Questi problemi sono stati risolti aumentando il numero delle milestone, distanziandole di circa 7-10 giorni, assegnando ad ognuna specifici obiettivi di sviluppo e monitorando l'uso delle risorse alla conclusione di ognuno dei periodi (corrispondenti ad incrementi di sviluppo) individuati. Per rendere ancora meno ambigua la pianificazione, ogni obiettivo è stato tracciato coi requisiti a cui corrispondeva: in questo modo ad ogni baseline era associato un avanzamento misurabile (in numero di requisiti)

Domanda (punti 8/30)

Lo standard ISO/IEC 12207 specifica l'attività ***configuration control*** (parte del processo di configurazione) come segue:

"This activity consists of the following (traced) tasks:
-identification and recording of change requests;
- analysis and evaluation of the changes;
- approval or disapproval of the request;
- (in case of approval) implementation, verification, and release of the modified software item."

Vogliamo comprendere se e come la concreta attuazione di tale attività all'interno di un progetto dipenda dal modello di sviluppo adottato. Più precisamente, vogliamo rispondere, in modo argomentato, a queste due domande:

1. Tale specifica, nella formulata di cui sopra, implica uno specifico modello di sviluppo? In caso affermativo, come potremmo rendere neutra la sua formulazione?
2. Come potremmo istanziare, adattandola, tale attività nelle nostre norme di progetto, in modo che essa si integri bene con uno sviluppo veramente incrementale?

Risposta

1) Le specifiche dello standard ISO/IEC 12207 per il controllo della configurazione non implicano esplicitamente un modello di sviluppo specifico. Delinea un insieme di compiti relativi alla gestione delle modifiche agli elementi del software, indipendentemente dal modello di sviluppo utilizzato. In generale, questi sono passi che vengono definiti appositamente in modo astratto, tale da poter aderire a più modelli di sviluppo qualora necessario. Consideriamo che le attività sono svolte in modo netto, pertanto potremmo facilmente far corrispondere il modello sequenziale; a questo, si può far corrispondere il modello incrementale, tale che si possa pensare di implementare queste modifiche una volta prodotti completamente i requisiti e andare ad implementare valore aggiunto qualora necessario. Se invece si pensasse di adottare un modello Agile si ha convergenza all'interno di limitato periodi di tempo per l'identificazione dei cambiamenti, successiva analisi ed approvazione, sulla base di quello che può essere definito all'interno dei periodi di riferimento (sprint).

Per rendere la formulazione di questa specifica neutrale rispetto al modello di sviluppo, potrebbe essere modificata per includere un linguaggio più generale che non faccia riferimento a processi o tecniche specifiche. Per esempio, la specifica potrebbe essere modificata nel modo seguente: "Questa attività consiste nei seguenti compiti: identificazione e registrazione delle richieste di modifica; analisi e valutazione delle modifiche; approvazione o disapprovazione della richiesta; implementazione, verifica e rilascio dell'elemento software modificato, come appropriato per il modello di sviluppo scelto".

2) Per istanziare l'attività di controllo della configurazione nel contesto di uno sviluppo realmente incrementale, la si potrebbe adattare come segue:

- Identificare e registrare le richieste di modifica man mano che vengono identificate durante il processo di sviluppo.
- Analizzare e valutare le modifiche per determinarne l'impatto sull'intero progetto e sul programma di consegna incrementale.

Scritto da Gabriel

- Ottenere l'approvazione o la disapprovazione della richiesta dalle parti interessate, come il team di progetto e la direzione.
- Se la richiesta è approvata, implementate la modifica nella consegna incrementale successiva e verificate che funzioni come previsto.
- Rilasciare l'elemento software modificato come parte della consegna incrementale successiva.

Questo adattamento dell'attività di controllo della configurazione tiene conto della natura incrementale del processo di sviluppo, incorporando l'implementazione, la verifica e il rilascio delle modifiche nel programma di consegna incrementale. Inoltre, consente la flessibilità necessaria per adattarsi alle modifiche che vengono identificate e approvate durante il processo di sviluppo. Tutto questo deve essere opportunamente documentato come protocollo da noi seguito all'interno delle Norme di Progetto, affinché si abbia decisa tracciabilità di questa forma mentis nell'azione e poter così decomporre facilmente i requisiti che lavorano in questo modo qualora utile/necessario.

Domanda (punti 8/30)

Vi chiediamo di discutere quale rapporto vi sia tra il numero di versione assegnato a ogni singolo prodotto di progetto software e la storia delle modifiche (cioè operazioni di scrittura) effettuate sui contenuti del *repository* ove tale prodotto risieda. Per dare un fondamento alla vostra risposta, converrà che definiate preliminarmente: (1) cosa sia secondo voi un "prodotto" di progetto, e cosa lo caratterizzi rispetto a qualunque altro artefatto di sviluppo; (2) quanti "*repo*" sia desiderabile avere in circostanze analoghe al vostro progetto didattico, se uno o più, e a quali destinatari, sviluppatori o clienti/utenti, essi siano rivolti; (3) quando e perché debba avvenire un incremento nel numero di versione assegnato a qualsivoglia prodotto nell'accezione di cui sopra.

Risposta

1) Un prodotto di progetto è un deliverable, quindi un oggetto che risulta dal processo di sviluppo ed è destinato all'uso da parte di stakeholder esterni, come clienti o utenti. Un prodotto di progetto è tipicamente un prodotto finito e pronto per l'uso, a differenza degli artefatti di sviluppo intermedi come il codice sorgente o gli artefatti di compilazione. Le caratteristiche di un prodotto di progetto possono includere la presenza di un'interfaccia utente, l'essere autonomo e l'essere testato e documentato.

2) In un progetto simile a un progetto didattico, è generalmente auspicabile avere almeno un repository per conservare il codice del progetto e altri artefatti di sviluppo. Questo repository dovrebbe essere accessibile al team di sviluppo e potrebbe anche essere accessibile a stakeholder esterni, come clienti o utenti, a seconda delle esigenze specifiche del progetto. In particolare, è desiderabile avere delle repo per ogni artefatto rilevante (es. una per la documentazione, una per il codice, etc.), adoperando vari branch/diramazioni dal ramo principale qualora necessario per implementare alcune particolari caratteristiche o idee di progetto.

3) Il numero di versione del prodotto di un progetto software deve essere aumentato quando ci sono cambiamenti o aggiornamenti significativi al prodotto. Ciò potrebbe includere l'aggiunta di nuove funzionalità, la correzione di bug significativi o altre modifiche che influiscono sulla funzionalità o sulla qualità complessiva del prodotto. Il numero di versione deve essere incrementato in modo coerente e logico, ad esempio utilizzando un sistema di versioning semantico.

La relazione tra il numero di versione di un prodotto di progetto e la storia delle modifiche apportate al repository in cui risiede il prodotto è che il numero di versione è tipicamente usato per indicare il livello di modifiche apportate al prodotto. Quando la storia del repository riflette le modifiche apportate al prodotto nel corso del tempo, il numero di versione viene incrementato per riflettere tali modifiche e aiutare gli stakeholder a identificare lo stato attuale del prodotto.

Domanda (punti 8/30)

Riflettiamo sui concetti di efficienza ed efficacia, prima inquadrandone il significato astratto, e poi trasponendolo in situazioni concrete, di progetto, attraverso esempi – almeno uno per ciascun concetto – che illustrino: (1) come, in cosa, in che modo, si manifestano; (2) come si misurano, con quali tecniche, quali strumenti, e quando; (3) come si migliorano.

Risposta

L'efficienza si riferisce alla capacità di utilizzare le risorse in modo efficace per raggiungere un risultato desiderato. Si tratta di ridurre al minimo l'uso delle risorse ottenendo lo stesso livello di output.

L'efficacia si riferisce alla capacità di raggiungere gli esiti o i risultati desiderati. Si tratta di raggiungere gli obiettivi desiderati di un progetto.

In una situazione di progetto, l'efficienza e l'efficacia possono manifestarsi in modi diversi. Ad esempio:

Efficienza:

- Un team di progetto che è in grado di completare le attività in modo tempestivo utilizzando meno risorse, come tempo e denaro, sta dimostrando efficienza.
- L'uso di strumenti di automazione per ridurre il tempo e l'impegno necessari per completare le attività è un esempio di efficienza.

L'efficienza può essere misurata utilizzando tecniche come l'analisi costi-benefici o confrontando le risorse utilizzate con i risultati ottenuti. Per monitorare e misurare l'efficienza si possono utilizzare strumenti come i software di gestione dei progetti.

L'efficienza può essere migliorata identificando ed eliminando le fasi o le attività non necessarie, utilizzando strumenti di automazione o ottimizzando l'uso delle risorse.

Efficacia:

- Un progetto che raggiunge gli obiettivi dichiarati dimostra efficacia.
- Un progetto che fornisce un prodotto o un servizio che soddisfa le esigenze degli utenti previsti è un esempio di efficacia.

L'efficacia può essere misurata valutando gli esiti o i risultati ottenuti rispetto agli obiettivi del progetto. Questo può essere fatto attraverso metodi come il feedback dei clienti, i test di usabilità o le metriche di performance.

L'efficacia può essere migliorata stabilendo obiettivi chiari, rivedendo e regolando regolarmente il piano di progetto e raccogliendo il feedback degli stakeholder per assicurarsi che il progetto soddisfi le loro esigenze.

Domanda (punti 8/30)

Tra gli oneri determinati dalle regole di svolgimento del progetto didattico di IS stanno le attività di concezione, realizzazione, alimentazione e uso di un "cruscotto di valutazione della qualità". All'atto pratico, però, l'esecuzione di tali attività da parte dei gruppi di progetto risulta spesso insoddisfacente, per comprensione dell'intento, valore informativo dei contenuti presentati, esperienza utente. Questo compito d'esame chiede una vostra riflessione sincera, matura e articolata, sulle possibili ragioni di questo *deficit*, con il vincolo di toccare direttamente (almeno) i seguenti punti:

- Se, nella vostra percezione del mondo reale, la qualità sia una dimensione rilevante e pertinente dello sviluppo *software* o solo un "vezzo didattico".
- Se, nella vostra esperienza corrente e aspettativa futura, la qualità di cui sopra sia misurabile in modo significativo, accurato e con un rapporto accettabile tra costi (di misurazione) e benefici (di conoscenza).
- Se e come, a vostro giudizio, la disponibilità di indicatori di qualità possa contribuire alla determinazione delle attività di progetto.
- Quanta attenzione, nella vostra prassi corrente, sia personale che di progetto didattico, abitualmente dedicate al controllo e misurazione della qualità, e perché.

Risposta

A mio avviso, la qualità è una dimensione molto importante e pertinente dello sviluppo del software. Sebbene possa essere facile concentrarsi esclusivamente sull'immissione sul mercato di un prodotto il più rapidamente possibile, il successo a lungo termine di un prodotto dipende spesso dalla sua qualità. Un software di scarsa qualità può comportare una serie di problemi, tra cui vulnerabilità della sicurezza, bug e un'esperienza utente scadente, che in ultima analisi possono portare a una perdita di clienti e di fatturato.

In termini di misurabilità, ritengo che la qualità possa essere misurata in modo significativo e accurato, anche se i metodi specifici di misurazione dipenderanno dagli obiettivi e dai requisiti specifici del progetto. È importante considerare attentamente i costi e i benefici dei diversi metodi di misurazione e scegliere quelli più appropriati ed efficaci dal punto di vista dei costi.

La disponibilità di indicatori di qualità può essere molto utile per determinare le attività del progetto. Tracciando e rivedendo regolarmente i principali indicatori di qualità, i team possono identificare le aree in cui devono migliorare e apportare modifiche ai processi per garantire la fornitura costante di prodotti di alta qualità.

Nella mia pratica attuale, dedico una notevole attenzione al controllo e alla misurazione della qualità. Ritengo che ciò sia importante per garantire che i prodotti che partecipo a sviluppare siano della massima qualità possibile. Ciò include la revisione regolare del codice per individuare eventuali problemi, la conduzione di test e convalide e la ricerca di feedback da parte degli utenti. Inoltre, ritengo che sia importante rivedere e valutare regolarmente l'efficacia dei nostri processi di controllo della qualità per identificare le aree in cui possono essere migliorati.

Domanda (punti 8/30)

Nel vostro progetto didattico, il PoC è post come punto di congiunzione tra le attività di analisi di requisiti e quelle di progettazione e codifica. Nasce quindi la domanda se lo sviluppo del PoC sia da vedere come supporto al consolidamento definito dei requisiti (e quindi parte dell'attività AR) oppure come focalizzato alla costruzione della baseline iniziale di prodotto tra parentesi e quindi parte dell'attività di progettazione software chiusa parentesi esperimenti in modo ragionato la vostra visione a riguardo, anche eventualmente deflettendo dalle ipotesi di cui sopra altra, fornendo argomenti articolati a supporto di essa punto per farlo strutturate la risposta in questo modo:

- Come abbiamo capito la domanda
- Come la pensiamo su di essa, precisamente
- Perché la pensiamo in questo modo

Risposta

Nella comprensione della domanda, sembra che ci sia qualche incertezza sul fatto che lo sviluppo di un Proof of Concept (PoC) debba essere considerato come parte delle attività di analisi dei requisiti o come parte delle attività di progettazione e codifica.

A mio avviso, il PoC è incentrato sulla costruzione della linea di base iniziale del prodotto e quindi fa parte delle attività di progettazione e codifica. Lo scopo di un PoC è quello di testare la fattibilità e la praticabilità di un prodotto o di una soluzione proposta, in genere costruendo una versione su piccola scala del prodotto. Ciò comporta la definizione dei requisiti e dei vincoli del prodotto, nonché lo sviluppo di un progetto preliminare e l'implementazione di un codice iniziale. Per questo motivo, riterrei che lo sviluppo di un PoC sia più strettamente allineato alle attività di progettazione e codifica che alle attività di analisi dei requisiti.

Ci sono alcune ragioni che mi spingono a pensarla in questo modo. In primo luogo, lo sviluppo di un PoC è tipicamente incentrato sul test e sulla verifica del prodotto o della soluzione proposta, piuttosto che sul semplice consolidamento dei requisiti. Ciò comporta la costruzione e la sperimentazione attiva del prodotto, piuttosto che la semplice raccolta e documentazione dei requisiti. In secondo luogo, lo sviluppo di un PoC richiede spesso la creazione di un progetto preliminare e l'implementazione di un codice iniziale, che sono tipicamente considerati parte delle attività di progettazione e codifica. Infine, lo scopo di un PoC è quello di fornire una prova di concetto per un prodotto o una soluzione proposta, il che è più strettamente legato alla progettazione e allo sviluppo del prodotto rispetto al processo di raccolta dei requisiti.

Domanda (punti 8/30)

Un lavoro scientifico in ambito SWE, recentemente candidato alla presentazione a un incontro di esperti di dominio, propone l'immagine qui riportata come rappresentazione concettuale di un modello di sviluppo agile e incrementale.

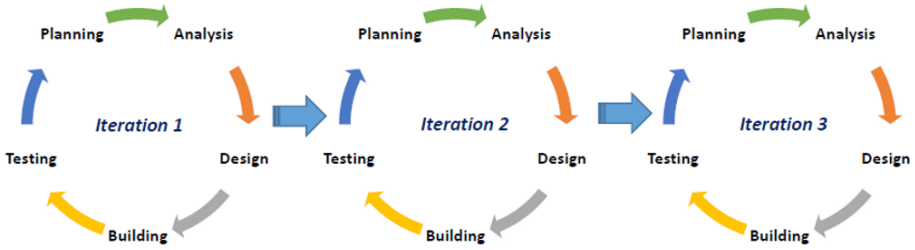

Fig. 1. The Agile model from [1, 2, 12]

1. Ambler, S., Nalbone, J., Vizdos, M., Enterprise Unified Process: Extending the Rational Unified Process. Prentice Hall (2005)
2. Ambler, S.,: Agile software development at scale. In: IFIP Central and East European Conference on Software Engineering Techniques, pp. 1–12. Springer (2007)
12. Kruchten, P.,The Rational Unified Process: An Introduction. Addison-Wesley (2004)

A voi è chiesto di discutere, in modo argomentato, la plausibilità dell'affermazione di cui sopra.

Risposta

Partendo dall'immagine presente, possiamo determinare i modelli agili come un concentrato di azione all'interno di singoli periodi, in cui si riuniscono i vantaggi dei modelli incrementali (produrre valore aggiunto ad ogni incremento) e iterativi (il fatto di poter essere flessibili e quindi correggersi "lungo la strada") secondo fasi singole e agili. Si prevede che questi processi siano principalmente basati sulla collaborazione, tra individui e interazioni determinate, al fine di avere un software funzionante rispetto alla documentazione comprensibile, rispondendo alle esigenze del cliente in modo successivo e rispondendo ai cambiamenti quando possibile. Si intende che quanto rappresentato dallo schema realizzi idealmente queste caratteristiche; all'interno delle singole iterazioni, viste successivamente in modo incrementale, si ha la realizzazione delle cosiddette "user story"/esigenze dell'utente, visti come needs. Le iterazioni comportano loro raffinazione, integralmente continua, al fine di poter garantire come schematizzato un'analisi integrativa a più passi, composti di fasi ripetute e omogenee tra loro.

La fase di analisi è certamente il primo passo, tale da fissare il design, la costruzione ed il test. Essendo basato su passi scomponibili ed essere le attività ripetute per loro definizione, Agile unisce entrambe le caratteristiche dette simbolicamente sopra al fine di concentrare completezza e iterazione all'interno di incrementi a valore aggiunto; in particolare, anche grazie alla facilità di modifica e continua collaborazione, l'integrazione continua risulta semplificata e attivamente utile a tutti gli stakeholders presenti. Se prendessimo ad esempio in considerazione il framework Scrum, potremmo facilmente vedere e definire le fasi graficamente rappresentate in periodi piccoli e predefiniti come sprint, al fine di pianificare, incontrarsi, rivedere tali pratiche e visualizzare, in retrospettiva, quanto fatto fino a quel momento e confrontare le azioni all'interno del backlog, determinando azioni riuscite, azioni in ritardo e producendo risultati attraverso azioni utili ma ripetibili come descritto e previsto, sia dallo schema che infine da modelli agili/incrementali.

Scritto da Gabriel

Domanda (punti 8/30)

Proporre concisamente una strategia di verifica tramite test sostenibile all'interno di un impegno di taglia analoga al progetto didattico, per dimensioni, livello di qualità atteso, disponibilità ed esperienza delle risorse. Concentrare la risposta sul tipo di test da prevedere, i loro obiettivi, il grado di automazione, e ogni altro fattore ritenuto di interesse. Corredare la risposta motivando ogni elemento della proposta.

Risposta

Una strategia di test-retesting sostenibile per un progetto didattico di dimensioni simili comporterebbe una combinazione di test manuali e automatici, incentrati su diversi aspetti del sistema.

Ecco alcuni elementi che potrebbero essere inclusi nella strategia:

1. Test unitari: Il test delle unità comporta la verifica di singoli componenti o unità del sistema per assicurarne il corretto funzionamento. Questo tipo di test può essere automatizzato ed è tipicamente responsabilità degli sviluppatori scrivere e mantenere i test unitari. L'obiettivo dei test unitari è quello di individuare i difetti nelle prime fasi del processo di sviluppo, prima che vengano integrati nel sistema.
2. Test di integrazione: Il test di integrazione consiste nel verificare il funzionamento dei diversi componenti del sistema. Questo tipo di test può essere in parte automatizzato, ma può anche comportare test manuali. L'obiettivo del test di integrazione è garantire che i diversi componenti del sistema siano in grado di comunicare e funzionare correttamente insieme.
3. Test di sistema: Il test del sistema prevede la verifica dell'intero sistema per assicurarsi che soddisfi i requisiti specificati. Questo tipo di test può comprendere sia test automatici che manuali. L'obiettivo del test del sistema è garantire che il sistema sia adatto allo scopo e soddisfi le esigenze degli utenti.
4. Test di accettazione dell'utente: Il test di accettazione da parte dell'utente consiste nel testare il sistema con un gruppo di utenti rappresentativi per assicurarsi che sia accettabile e utilizzabile. Questo tipo di test è tipicamente manuale e viene eseguito dagli utenti o da un gruppo di tester che li rappresenta. L'obiettivo del test di accettazione è garantire che il sistema soddisfi le esigenze e le aspettative degli utenti.
5. Automatizzare il più possibile i test può contribuire a rendere il processo di test-retesting più efficiente e sostenibile. I test automatizzati possono essere eseguiti in modo rapido e coerente e non richiedono lo stesso livello di intervento umano dei test manuali. Tuttavia, è importante bilanciare l'uso dell'automazione con quello dei test manuali, poiché i test automatici potrebbero non essere in grado di coprire tutti gli aspetti del sistema e di identificare problemi che possono essere scoperti solo attraverso i test manuali.

In generale, una strategia di test-retesting sostenibile dovrebbe mirare a coprire tutti gli aspetti del sistema e garantire che sia di alta qualità e che soddisfi le esigenze degli utenti. Dovrebbe inoltre essere efficiente e fare uso dell'automazione laddove possibile, pur consentendo ai test manuali di identificare i problemi che non possono essere rilevati attraverso l'automazione.

Scritto da Gabriel

Varie:

- *Presentare almeno tre qualità da perseguire nella progettazione software, indicando anche come esse possano essere verificate e misurate quantitativamente. Ove possibile, motivare la risposta tramite riferimenti espliciti e concreti, positivi o negativi, alla propria esperienza di progetto didattico*

1) Manutenibilità: Questa qualità si riferisce alla facilità con cui un sistema software può essere modificato o aggiornato nel tempo. È importante progettare i sistemi software in modo da renderli facilmente modificabili, in modo da poterli aggiornare e adattare al mutare dei requisiti aziendali. La manutenibilità può essere verificata e misurata attraverso tecniche come la revisione del codice, i test e la documentazione.

2) Scalabilità: Questa qualità si riferisce alla capacità di un sistema software di gestire un carico di lavoro o una base di utenti sempre più ampia senza che le prestazioni diminuiscano. È importante progettare i sistemi software tenendo conto della scalabilità, in modo che possano continuare a funzionare efficacemente anche quando la domanda di utilizzo cresce. La scalabilità può essere verificata e misurata attraverso test delle prestazioni, test di carico e monitoraggio dell'utilizzo delle risorse.

3) Sicurezza: Questa qualità si riferisce alle misure adottate per proteggere un sistema software da accessi non autorizzati o usi impropri. È importante progettare i sistemi software tenendo conto della sicurezza, poiché qualsiasi vulnerabilità può avere gravi conseguenze sia per l'utente che per l'azienda. La sicurezza può essere verificata e misurata attraverso test di penetrazione, revisioni del codice e l'uso delle migliori pratiche di sicurezza.

- *Discutere le differenze e le correlazioni tra le attività dei processi di versionamento e configurazione nel ciclo di vita dei sistemi software*

Il versioning è il processo di assegnazione di identificatori unici alle diverse versioni di un sistema software. In genere viene utilizzato per tenere traccia delle modifiche apportate al software nel corso del tempo e per consentire agli utenti di scegliere la versione del software che desiderano utilizzare. Può essere particolarmente utile per la correzione di bug e per mantenere la compatibilità con altri sistemi o dipendenze.

La configurazione è il processo di adattamento di un sistema software per soddisfare le esigenze specifiche di un particolare utente o ambiente. Ciò può comportare la regolazione delle impostazioni, l'installazione di funzionalità o componenti aggiuntivi o l'integrazione con altri sistemi. La configurazione viene generalmente eseguita per personalizzare il comportamento o la funzionalità del software in modo da soddisfare le esigenze specifiche dell'utente o dell'ambiente.

Esiste una chiara correlazione tra versioning e configurazione nel ciclo di vita dei sistemi software. Quando un sistema software viene aggiornato o modificato, alle modifiche viene in genere assegnato un nuovo numero di versione. Questo permette agli utenti di tenere traccia delle modifiche apportate e di scegliere la versione più adatta alle proprie esigenze. Allo stesso tempo, potrebbe essere necessario aggiornare la configurazione del software per riflettere le modifiche apportate, al fine di garantire che sia pienamente funzionale e soddisfi le esigenze dell'utente.

In generale, il versionamento e la configurazione sono entrambe attività importanti nel ciclo di vita dei sistemi software. Il versionamento aiuta a tenere traccia delle modifiche e a mantenere la compatibilità, mentre la configurazione aiuta a personalizzare il software per soddisfare le esigenze specifiche dell'utente. Entrambe le attività sono fondamentali per garantire che i sistemi software siano efficaci e affidabili nel tempo.

Scritto da Gabriel

- *È da ritenersi che la buona (efficace ed efficiente) gestione di progetto, project management, si giovi di strumenti tecnologici di supporto; confutate tale affermazione con appropriate argomentazioni oppure provate ad illustrare i requisiti principali che ritenete debbano essere soddisfatti da soluzioni professionali. Qualora nel vostro progetto didattico siate rimasti soddisfatti di qualche particolare strumento di supporto rilevante al quesito, descrivetene succintamente le caratteristiche pertinenti più significative.*

È generalmente riconosciuto che una buona gestione dei progetti può trarre beneficio dall'uso di strumenti tecnologici di supporto. Questi strumenti possono aiutare i project manager a pianificare, monitorare e organizzare le attività e le risorse in modo più efficace, migliorando l'efficienza e l'efficacia.

Ci sono diversi requisiti che le soluzioni professionali devono soddisfare per essere efficaci nel supportare la gestione dei progetti. Tra questi vi sono:

1. Integrazione: Lo strumento deve essere in grado di integrarsi con altri sistemi e strumenti utilizzati dal team, come il software di gestione dei progetti, le piattaforme di collaborazione e gli strumenti di comunicazione.
2. Collaborazione: Lo strumento deve facilitare la collaborazione e la comunicazione tra i membri del team, consentendo loro di condividere file, monitorare i progressi e comunicare in tempo reale.
3. Personalizzazione: Lo strumento deve essere personalizzabile per soddisfare le esigenze e le preferenze specifiche del team e del progetto.
4. Reporting e analisi: Lo strumento deve fornire solide funzionalità di reporting e analisi, consentendo ai project manager di monitorare i progressi, identificare i problemi e prendere decisioni informate.
5. Scalabilità: Lo strumento deve essere in grado di adattarsi alle esigenze del progetto, sia che si tratti di un progetto di piccole, medie o grandi dimensioni.

Uno strumento che ho trovato particolarmente utile per supportare la gestione dei progetti è Asana. Asana è una piattaforma di collaborazione e gestione dei progetti che consente ai team di tenere traccia di attività, progetti e risorse in un unico luogo. Dispone di un'ampia gamma di funzioni, tra cui la possibilità di creare e assegnare compiti, impostare scadenze e dipendenze e monitorare i progressi. Si integra anche con altri strumenti, come Google Drive e Slack, e offre visualizzazioni e report personalizzabili per aiutare i project manager a monitorare i progressi e a prendere decisioni informate.

- *Immaginando di dover redigere un glossario come quello del progetto, dare una definizione dei seguenti termini: progetto, processo, attività, fase. Ove possibile citare fonti autorevoli.*

Progetto: Un progetto è uno sforzo temporaneo intrapreso per creare un prodotto, un servizio o un risultato unico. Secondo il Project Management Institute (PMI) A Guide to the Project Management Body of Knowledge (PMBOK), un progetto è definito come "uno sforzo temporaneo intrapreso per creare un prodotto, un servizio o un risultato unico".

Processo: Un processo è una serie di attività che lavorano insieme per raggiungere un risultato specifico. Secondo il PMBOK del PMI, un processo è definito come "una serie di attività che interagiscono per produrre un risultato".

Attività: Un'attività è un compito o un'azione specifica che viene eseguita come parte di un processo più ampio. Secondo il PMBOK del PMI, un'attività è definita come "un compito o un'azione specifica che viene eseguita come parte di un processo più ampio".

Fase: Una fase è un periodo di tempo distinto all'interno di un progetto, durante il quale si svolge un insieme specifico di attività. Secondo il PMBOK del PMI, una fase è definita come "un periodo di tempo distinto all'interno di un progetto, durante il quale si svolge un insieme specifico di attività".

- *Dare una definizione ben fondata del concetto di "architettura software". In relazione a tale concetto, dare una definizione ai termini "framework" e "design pattern" spiegando come questi si integrino fra loro E all'interno di una architettura*

L'architettura del software si riferisce alle strutture fondamentali di un sistema software e alla disciplina di creazione di tali strutture e sistemi. Comporta l'identificazione dei componenti del sistema software e delle relazioni tra di essi, nonché dei principi che guidano la loro progettazione ed evoluzione.

Un framework è un insieme di librerie, componenti e strumenti che forniscono una struttura comune per lo sviluppo di un sistema software. In genere include un insieme di classi, interfacce e altri elementi predefiniti che possono essere utilizzati come punto di partenza per lo sviluppo di un sistema software.

Un design pattern è una soluzione generale e ripetibile a un problema comunemente riscontrato nella progettazione del software. Un modello di progettazione non è un progetto finito che può essere trasformato direttamente in codice. Si tratta di una descrizione o di un modello per la soluzione di un problema che può essere utilizzato in molte situazioni diverse.

I design pattern sono utilizzati per descrivere le soluzioni ai problemi che si presentano ripetutamente nella progettazione del software. Forniscono un linguaggio comune che gli sviluppatori possono utilizzare per comunicare sulla progettazione e possono essere usati come punto di partenza per la costruzione di un sistema software.

In relazione all'architettura del software, i framework e i design pattern possono essere utilizzati per aiutare a implementare e organizzare i componenti e le relazioni identificate nell'architettura del software. Essi forniscono un insieme di strutture e soluzioni predefinite che possono essere utilizzate per costruire in modo efficiente ed efficace il sistema software.

- *Gran parte dei gruppi che intraprende il progetto didattico dichiara di aderire al modello di sviluppo incrementale, per poi deflettere significativamente da esso. Descrivere le caratteristiche salienti di quel modello di sviluppo e discutere in breve le principali difficoltà che si possono incontrare nell'adesione ad esso.*

Il modello di sviluppo incrementale è un approccio allo sviluppo del software in cui un sistema viene sviluppato attraverso cicli ripetuti di pianificazione, implementazione, test e consegna di incrementi funzionali. Ogni incremento si basa sui precedenti, aggiungendo nuove funzionalità e perfezionando quelle esistenti.

Il modello di sviluppo incrementale presenta diverse caratteristiche salienti:

1. La funzionalità viene fornita per incrementi: Invece di fornire l'intero sistema in una sola volta, la funzionalità viene fornita in piccoli incrementi, ognuno dei quali aggiunge valore al sistema.
2. I requisiti vengono raccolti e classificati in base alle priorità: I requisiti del sistema vengono raccolti e classificati in base alle priorità, e quelli più importanti vengono affrontati per primi.
3. Feedback e iterazione continui: Il processo di sviluppo è iterativo, con feedback e aggiustamenti continui basati sui test e sui suggerimenti degli utenti.
4. Il rischio è ridotto: Fornendo incrementi funzionali presto e spesso, si riduce il rischio di consegnare un sistema difettoso.

L'adesione al modello di sviluppo incrementale può presentare diverse difficoltà. Una difficoltà è rappresentata dalla raccolta accurata e dalla definizione delle priorità dei requisiti, poiché il sistema si

evolve e cambia nel tempo. Può anche essere difficile gestire la complessità del sistema, che cresce a ogni incremento. Inoltre, può essere difficile convincere gli stakeholder a impegnarsi in un processo di sviluppo che non prevede la consegna dell'intero sistema in una sola volta.

- *Illustrare le principali differenze, per obiettivi e modalità di svolgimento, tra le tecniche di inspection e di walkthrough.*

Partiamo con una premessa: le attività di verifica a qualsiasi livello adottano tali modalità, sia per quanto riguarda questa che la detta validazione. Possiamo evidenziare le loro principali differenze, dicendo che la *inspection* trattasi di tecnica formale attraverso lettura mirata, normalmente alla presenza di verificatori. Prevede normalmente una *checklist* automatizzabile, che definisce cosa vada verificato, poi una successiva fase di lettura (a cura degli autori) e correzione dei difetti presenti. Questa segue un insieme di linee guida e procedure, al fine di pianificare accuratamente il processo di revisione e di coordinarlo. Essa poi agisce principalmente a grana fine, focalizzandosi su presupposti ("error guessing") e documentare attività svolte e risultanze.

D'altro canto, il processo di *walkthrough* rileva la presenza di difetti con un'analisi ad ampio spettro, come tale meno formale, secondo un esame privo di assunti e certezze. Prevede un insieme di autori e verificatori, che eseguono una pianificazione, una successiva lettura, un'attenta discussione e successiva correzione dei difetti. L'obiettivo di un walkthrough è quello di identificare eventuali problemi o aree di miglioramento e di fornire feedback e suggerimenti per il miglioramento.

In sintesi, la differenza principale tra le tecniche di ispezione e di walkthrough è che l'ispezione è un processo più formale e strutturato che si concentra sulla ricerca di difetti, mentre il walkthrough è un processo meno formale e interattivo che si concentra sulla comprensione e sulla valutazione di un prodotto o di un documento.

- *Fornire una definizione di "requisito", applicabile al dominio dell'Ingegneria del Software e descrivere, succintamente ma con precisione, il ciclo di vita all'interno di un progetto del tipo di quello didattico, rappresentandolo come un'apposita macchina a stati, specificando anche le attività poste sugli archi di transizione.*

Un requisito viene definito, nel dominio dell'Ingegneria del Software, come un'affermazione che risponde doppiamente a specifici bisogni (lato utente, al fine di rappresentare una richiesta esplicita o un problema da risolvere rispetto a chi ha esigenze di fornitura e lato soluzione, al fine di rispondere chiaramente alle aspettative presenti). Questo stabilisce un confine di precisione entro il quale sviluppare il sistema che si intende creare, che permetta di capire dove spingere la complessità del prodotto e assicurare che i bisogni vengano tutti soddisfatti. Allo stesso tempo, è comprensione del prodotto al suo profondo: non tutti i requisiti sono espliciti, ma richiedono seria ed attenta analisi su quanto presente per poter compiere correttamente tutte le attività.

In questo, possiamo individuare correttamente il ciclo di vita come insieme di stati percorsi tra concepimento e realizzazione/ritiro del prodotto finale, secondo processi (che codificano stati e transizioni al fine di individuare bisogni e fornire soluzioni) e modelli (per capire quali stati/transizioni privilegiare, pianificando correttamente le attività). Esso permette di sviluppare un prodotto e continuare a mantenerlo sviluppabile/*usabile*, al fine di poter effettuare attività di manutenzione. Questa mantiene la cosiddetta qualità di prodotto, assicurandosi che venga aggiornato e modificato quando necessario per rimuovere difetti e aggiungere nuove caratteristiche

Vedendolo come macchina a stati, possiamo individuare le seguenti attività come composizione:

Stati:

- Nuovo: il requisito è stato identificato ma non è ancora stato analizzato o documentato.
- Analizzato: Il requisito è stato analizzato e il suo impatto sul progetto è stato valutato.
- Documentato: Il requisito è stato documentato ed è pronto per la revisione.
- Revisionato: Il requisito è stato rivisto e sono state apportate le modifiche necessarie.
- Approvato: Il requisito è stato approvato ed è pronto per essere implementato.
- Implementato: Il requisito è stato implementato ed è pronto per essere testato.
- Testato: Il requisito è stato testato ed è pronto per essere distribuito.
- Distribuito: Il requisito è stato distribuito ed è in uso.

Idealmente, va in loop; si parla di manutenzione, quindi di aggiornamento e modifica per rimuovere difetti e aggiungere nuove caratteristiche.

Archi di transizione:

- Analizzare: Il processo di analisi del requisito e di valutazione del suo impatto sul progetto.
- Documentare: Il processo di documentazione del requisito.
- Revisione: Il processo di revisione del requisito e di apporto delle modifiche necessarie.
- Approvare: Il processo di approvazione del requisito.
- Attuare: Il processo di implementazione del requisito.
- Test: Il processo di verifica del requisito, affinché risponda ai bisogni richiesti
- Distribuire/Deploy: Il processo di distribuzione del requisito, installato e reso disponibile agli utenti

- *Fornire una definizione del concetto di framework, specificamente collocato in relazione alla progettazione software, insieme ad una descrizione sintetica dei criteri applicati per selezione ed eventuale uso di qualcuno di essi, come sperimentato nel proprio progetto didattico.*

Un framework software è un insieme di componenti o librerie riutilizzabili che forniscono un'infrastruttura comune per la costruzione di applicazioni software. Serve come base su cui gli sviluppatori di software possono costruire, estendere e personalizzare il framework per soddisfare le esigenze specifiche delle loro applicazioni. Essi vengono normalmente utilizzati in quanto semplici da usare e da imparare, con chiari esempi di customizzazione e documentazione. Ne esistono di diverso tipo: backend web per velocizzare il processo di accesso dati, gestione sessioni e template di pagine (Django) oppure frontend, per definire strutture di componenti e specifici stili (Angular, React

Ci sono diversi criteri che possono essere applicati quando si seleziona un software per un progetto didattico. Questi criteri possono includere

- Facilità d'uso: Un framework dovrebbe essere facile da imparare e da usare, con una documentazione ed esempi chiari.
- Flessibilità: Il framework dovrebbe consentire la personalizzazione e l'estensione per soddisfare le esigenze specifiche del progetto.
- Prestazioni: Il framework deve essere efficiente e scalabile, in grado di gestire il carico di lavoro del progetto senza intasare il sistema.
- Sicurezza: Il framework deve fornire funzioni e pratiche di sicurezza integrate per proteggere dalle vulnerabilità più comuni.
- Supporto: Il framework dovrebbe disporre di una solida comunità di sviluppatori e utenti in grado di fornire supporto e indicazioni in caso di necessità.

Scritto da Gabriel

In definitiva, la scelta di un framework software per un progetto didattico dipenderà dai requisiti e dagli obiettivi specifici del progetto, nonché dal livello di competenza e dalle preferenze del team di sviluppo. Può essere utile ricercare e confrontare diversi framework per determinare quello più adatto al progetto. Se ne sono dati alcuni esempi in ottica reale.

- *Presentare, per obiettivi, criteri di valorizzazione, possibilità di automazione, due metriche significative per la misurazione della qualità della progettazione software e del codice (quindi almeno una metrica per ciascun ambito). Ove possibile, fare riferimento a esperienze specifiche e personali per valutare l'esito osservato dall'eventuale uso pratico delle metriche discusse.*

Per la misurazione della qualità del software, l'obiettivo è mantenere un software conciso, evidente dimostrazione degli obiettivi presenti, al fine di renderlo manutenibile e scalabile e possibilmente modulare, quindi scomposto in singole parti coerentemente legate ed organizzate. Si intende che le metriche sono strumenti documentati, che permettono di quantificare un prodotto secondo delle scale di misurazione attendibili e dimostrabili dal contesto presente. La loro valorizzazione è data dal perseguimento continuo di questi obiettivi, puntando alla automazione, ripetendo più volte gli stessi controlli in modo preciso e misurato.

Alcune metriche per misurare la qualità della progettazione del software:

- Complessità del codice: Una metrica per misurare la qualità della progettazione del software è la complessità del codice, che si riferisce al livello di difficoltà nella comprensione e nella manutenzione del codice. Questa può essere misurata utilizzando strumenti come la complessità ciclomatica, che calcola il numero di percorsi indipendenti attraverso un pezzo di codice. Un punteggio di complessità più basso indica che il codice è più facile da capire e da mantenere.

- Riutilizzo del codice: Un altro parametro per misurare la qualità della progettazione del software è il riutilizzo del codice, che si riferisce alla misura in cui il codice viene riutilizzato o condiviso in diverse parti dell'applicazione. Questo può essere misurato calcolando la percentuale di codice riutilizzato o tenendo traccia del numero di volte in cui viene utilizzato un particolare pezzo di codice. Livelli più elevati di riutilizzo del codice possono indicare una progettazione più efficiente e modulare.

Metriche per misurare la qualità del codice:

- Copertura del codice: Una metrica per misurare la qualità del codice è la copertura del codice, che si riferisce alla percentuale della base di codice che viene esercitata da test automatici. Una copertura del codice più elevata indica che una parte maggiore del codice è stata testata, il che può fornire fiducia nella stabilità e nella correttezza del codice.

- Densità dei difetti: Un'altra metrica per misurare la qualità del codice è la densità dei difetti, che si riferisce al numero di difetti o bug per unità di codice. Si può calcolare dividendo il numero totale di difetti trovati nel codice per il numero totale di linee di codice. Una minore densità di difetti indica che il codice è di qualità superiore.

Tali metriche risultano utili per raccogliere informazioni sulle singole parti che compongono il software e poter confrontare, a cruscotto disponendo di informazioni di qualità, di un andamento oggettivo del prodotto e delle sue componenti. Tendono ad essere molto specifiche in genere, raccogliendo dati che possono dipendere dal contesto stesso e le sue parti; normalmente, sono molteplici e lavorano in parallelo. Concretamente, per esempio, si parla di SLOC (Source Delivered Lines of Code) per il software consegnato, metrica non del tutto affidabile e dipendente dall'abilità di chi lo scrive e dai linguaggi utilizzati, a metriche oggettive/automatiche, ad es. gli indici di leggibilità dei documenti o la citata copertura del codice, fino ad

arrivare a metriche complesse, come le dipendenze del codice. Tutti questi dati si completano, determinando strutturalmente coerenza di informazioni riportate e, all'esterno, un sistema in grado di essere manutenibile e ben implementato.

- *Facendo riferimento allo standard ISO/IEC 12207, discutere la differenza di obiettivi, attività coinvolte, strategie di conduzione e strumenti, tra i processi di verifica e validazione.*

Partiamo con una premessa di fatto di questo standard, essendo il modello più noto nell'ambito del ciclo di vita dei processi software, specificando i processi che devono essere seguiti durante lo sviluppo, mantenimento e ritiro di sistemi software, compresi ruoli/responsabilità degli stakeholder (migliorando la collaborazione/comunicazione con essi). Mantiene una struttura modulare, specializzandosi caso per caso. Si intende che un prodotto abbia uno scopo e debba rispondere a degli obiettivi (needs/bisogni); in questi individuiamo pratiche ripetibili che ci possano permettere all'interno di periodi di tempo scanditi e limitati (fasi) un'applicazione ripetuta e possibilmente automatizzabile, sotto forma di processi. Questi ultimi sono divisi in categorie, ognuna a supporto dell'altra, con apposite tecniche.

Questa è importante per parlare di entrambe le cose:

- la verifica è concentrata sull'assicurare che un prodotto/sistema/componente sia fatta nel modo corretto e che risponda ai giusti bisogni/requisiti (did I build the system right?). Cerca di verificare difetti e devianze dal percorso giusto il prima possibile, normalmente prima che il prodotto possa essere consegnato. Essa fornisce prove oggettive e si compone di processi singoli in grado di individuare quanto in oggetto, per esempio ad ampio raggio (walkthrough) oppure a tappeto/in profondità/a pettine (inspection), sulla base di quanto fatto in periodi, verificando in retrospettiva tramite riunioni gli specifici artefatti prodotti nelle fasi, incrementali/iterative/agili

- la validazione conforma che il sistema risponda alle esigenze degli utenti e agli scopi voluti, considerando che ogni cosa deve essere dimostrabile e come tale tracciata, per affermare con certezza la correttezza (did I build the right system). I periodi certificano, sulle base di obiettivi, l'importanza delle cose raggiunte (milestone) al fine di consolidare risultati di lungo termine (baseline). Questo avviene spesso con la presentazione/collaudo con gli stakeholders, verificando con tecniche formali la corretta esecuzione del prodotto, assicurando abbia i requisiti voluti (analisi dinamica), verificando che ogni azione che non produce esecuzione sia alla base corretta (analisi statica). I test sono ripetibili ed automatizzati

- *Fissando l'attenzione sulla definizione di processo associata allo standard ISO/IEC 12207, indicare quali processi sia possibile e opportuno istanziare su una attività di tagli analoga al progetto didattico, e con quale istanziazione concreta (cioè verso quale insieme di attività, obiettivi, e flussi di dipendenza).*

Il processo viene definito come insieme di attività coese per raggiungere determinati obiettivi in modo correlato, secondo regole che consumano risorse ed ottengono uscite in modo economico (efficace ed efficiente, dunque impiegando in modo ottimale le risorse e minimizzando i tempi previsti). Ciascuno di questi è sottoposto ad un'attività di controllo opportuna.

Il taglio analitico a cui sottoporre un progetto intende in quali classificazioni si può pensare di istanziare le domande, in particolare individuando:

- Processo di pianificazione: che include l'identificazione degli obiettivi del progetto, la definizione dei requisiti e la creazione di un piano di progetto.

- Processo di acquisizione: che include l'acquisizione di tutti i componenti del software necessari per il progetto, come i software di terze parti, i tool di sviluppo e i servizi di supporto.

- Processo di sviluppo: che include l'analisi, la progettazione, la codifica, il testing e la documentazione del software.

- Processo di configurazione: che include la gestione e la controllo delle versioni del software, la gestione dei problemi e la gestione della configurazione del progetto.

- Processo di gestione della qualità: che include la valutazione della qualità del software e la gestione dei problemi di qualità.

- Processo di gestione dei rischi: che include l'identificazione e la valutazione dei rischi associati al progetto, nonché la definizione delle azioni per mitigare i rischi identificati.

- Processo di gestione della consegna e della manutenzione: che include la pianificazione e la gestione della consegna del software e la gestione della manutenzione del software.

- *Spiegare concisamente la differenza tra il modello di sviluppo "iterativo" e quello "incrementale", fornendo sufficienti elementi per comprendere sia l'uno che l'altro. Nella spiegazione, descrivere anche le condizioni al contorno che rendano preferibile l'uno o l'altro*

Il modello di sviluppo iterativo è un approccio allo sviluppo del software in cui il processo di sviluppo viene ripetuto in una serie di iterazioni o cicli. Ogni iterazione comporta la progettazione, l'implementazione, il collaudo e il perfezionamento di una parte del software. L'obiettivo di ogni iterazione è quello di fornire un incremento utilizzabile e valido del software, raccogliendo al contempo feedback e apportando miglioramenti in base a questi ultimi.

Il modello di sviluppo incrementale è simile al modello iterativo, in quanto prevede la consegna di piccoli incrementi del software nel tempo. Tuttavia, nel modello incrementale, ogni incremento si basa su quelli precedenti, aggiungendo nuove caratteristiche e funzionalità al software. L'obiettivo del modello incrementale è quello di fornire un prodotto completo e utilizzabile per gradi, piuttosto che tutto in una volta.

Nella mia esperienza di progetto didattico, il modello di sviluppo iterativo sarebbe più adatto al caso. Il modello iterativo, infatti, consente una maggiore flessibilità e adattabilità, in quanto permette al team di sviluppo di raccogliere feedback e apportare modifiche al software mentre viene sviluppato. Questo può essere particolarmente utile in un progetto didattico, dove le esigenze e gli obiettivi degli utenti possono evolvere nel tempo e il software può dover essere modificato per soddisfare tali esigenze.

Il modello di sviluppo incrementale può essere più adatto a progetti in cui i requisiti sono più definiti e la portata è più limitata. In un progetto didattico, in cui i requisiti possono essere più aperti e l'ambito più dinamico, il modello iterativo può essere più efficace per consentire al team di sviluppo di rispondere alle mutevoli esigenze e priorità.

- *Un numero crescente di domini applicativi predilige l'adozione dei metodi di sviluppo agile (nelle loro svariate declinazioni). Richiamandone concisamente le caratteristiche distintive, discutere come esse si adattino alle esigenze del progetto didattico, ove possibile confrontandole con quelle del metodo effettivamente adottato*

I metodi agili sono un insieme di metodi e pratiche che danno priorità a flessibilità e collaborazione e basano il loro modello in piccoli incrementi a valore aggiunto realizzati in sequenza continua, dimostrando trasparenza di progresso e continuo miglioramento in periodi appositi (chiamati Sprint in framework Agile come Scrum). Questo permette al software di essere funzionante e permettere un feedback frequente ed aggiustamenti continui. In particolare, i team sono in grado di autoorganizzarsi, sapendo che i membri possono prendere decisioni, confrontarsi in modo continuativo, coinvolgendo il cliente e permettendo un adattamento in modo incrementale e continuo e dialogando continuativamente con lui.

Nell'attività di progetto didattico, tutte queste caratteristiche emergono pienamente, dato che si ha una strutturazione ben definita in ruoli e mansioni, garantendo continuo coinvolgimento del cliente rispetto al "fornitore" gruppo di progetto, team che decidono i migliori strumenti per controllare e gesture al meglio le proprie attività in periodi di tempo limitati e producendo valore migliorante e adattandosi al feedback continuo/cambiamenti, in modo flessibile e ben strutturato.

- *Fornire una definizione concisa e ben fondata del concetto di "architettura software" e "software framework", radicati nel dominio dell'ingegneria del software. Nella risposta, evidenziare gli eventuali punti di contatto tra essi, possibilmente corroborando l'argomento con almeno un esempio concreto.*

L'architettura del software si riferisce alla struttura di alto livello di un sistema software, compresi i componenti che lo costituiscono, le relazioni tra questi componenti e i principi che guidano la progettazione del sistema. Essa è la soluzione utile a quanto voluto da parte di tutti gli stakeholders, sulla base di una checklist di requisiti. È la progettazione complessiva di un sistema software, che si concentra sul quadro generale e fornisce uno schema dei componenti del sistema e delle loro interazioni. L'architettura di un sistema software è spesso rappresentata mediante diagrammi e modelli, come i diagrammi UML, che forniscono una rappresentazione visiva della struttura del sistema. Ne esistono di diversi tipi; una buona architettura è in grado di essere sufficiente, comprensibile, robusta e modulare, scomponibile in pezzi in grado di essere coerenti nel loro complesso, garantendo semplicità ed incapsulazione (minimizzando possibile accoppiamento).

Un framework software, invece, è un insieme di componenti e protocolli che forniscono una struttura comune per la costruzione di un particolare tipo di software. Un framework fornisce una struttura predefinita per l'architettura di un sistema e un insieme di componenti e librerie riutilizzabili che possono essere utilizzati per costruire il sistema. I framework forniscono un insieme comune di interfacce che lo sviluppatore deve utilizzare per interagire con il framework e utilizzare le funzionalità fornite. Ciò fornisce una struttura comune, rendendo l'applicazione più facile da mantenere e da capire. Un framework impone anche una certa architettura, che lo sviluppatore deve rispettare.

I punti di contatto tra l'architettura del software e i framework sono i seguenti:

- L'architettura del software guida la progettazione del sistema e fornisce uno schema dei componenti del sistema e delle loro interazioni.
- Un framework software è un insieme di componenti e protocolli che forniscono una struttura comune per la costruzione di un sistema software e fornisce un'architettura predefinita per un sistema.

Un esempio di framework software che include anche l'architettura è l'architettura Model-View-Controller (MVC), che è un modello di progettazione comunemente usato nello sviluppo del software. MVC è sia un pattern che un framework e viene utilizzato per separare le preoccupazioni di un sistema software, come i dati (modello), la presentazione (vista) e la logica (controllore). Questa separazione delle preoccupazioni è il principale vantaggio di MVC, in quanto promuove una separazione netta della logica, rendendo l'applicazione più facile da mantenere e da capire. Molti framework web come Ruby on Rails, AngularJS e Spring si basano sull'architettura MVC.

- *Fornire una definizione del concetto di "versionamento" applicabile al dominio dell'ingegneria del software. Elencare le principali attività in esso coinvolte. Ove possibile, presentare brevemente lo strumento utilizzato a tal fine nel propri progetto didattico o in altra esperienza personale*

Nell'ambito dell'ingegneria del software, il versioning si riferisce al processo di tracciamento e gestione delle modifiche apportate a un prodotto o sistema software nel corso del tempo. È un modo per tenere traccia delle diverse versioni del software, in modo che diversi team o utenti possano lavorare sulla stessa base di codice senza interferire gli uni con gli altri.

Le attività principali coinvolte nel versioning sono diverse:

- Version control: Il controllo delle versioni è il processo di tracciamento e gestione delle modifiche alla base di codice. Questo può essere fatto manualmente, tenendo traccia delle modifiche in un foglio di calcolo o in un documento, o automaticamente, utilizzando un sistema di controllo delle versioni (VCS). Un VCS è uno strumento software che tiene traccia delle modifiche alla base di codice, memorizza diverse versioni del codice e consente agli utenti di eseguire il rollback delle modifiche, se necessario.
- Branching: il branching è il processo di creazione di una copia separata della base di codice, o "ramo", su cui lavorare. Ciò consente a più team o utenti di lavorare contemporaneamente su versioni diverse del codice, senza influire sulla base di codice principale.
- Merging: Il merging è il processo di combinazione delle modifiche provenienti da rami diversi per riportarle nella base di codice principale. Ciò può comportare la risoluzione dei conflitti tra le modifiche apportate in rami diversi e la verifica del codice unito per assicurarne la stabilità e la correttezza.
- Tagging: Il tagging è il processo di marcatura di una versione specifica del codice con un'etichetta o "tag". Questo può essere usato per identificare una particolare versione del codice, come una versione di rilascio, o per contrassegnare un punto della base di codice per riferimento.

Nel mio progetto didattico, abbiamo usato GitHub e Git, noto sistema di controllo di versione distribuito, per tracciare e gestire le modifiche alla base di codice. Git ci permette di creare e passare da un ramo all'altro, unire le modifiche e contrassegnare versioni specifiche del codice. Fornisce inoltre un repository centrale per l'archiviazione e la condivisione del codice e ci permette di collaborare con altri membri del team in un ambiente di sviluppo distribuito.

- *Fornire una definizione del concetto di qualità, applicabile al dominio dell'ingegneria del software. Discutere concisamente le sue ramificazioni e, per una di esse a scelta, illustrare le principali attività da svolgere per il suo perseguimento. Ove disponibile, riferire nella discussione l'esperienza acquisita al riguardo nel proprio progetto didattico.*

Nell'ambito dell'ingegneria del software, la qualità si riferisce al grado in cui un sistema software soddisfa le esigenze e le aspettative dei suoi utenti e delle parti interessate, nonché alla sua capacità di funzionare in modo corretto ed efficiente. La qualità può essere vista da diverse prospettive, tra cui funzionalità, affidabilità, usabilità, prestazioni, manutenibilità e sicurezza.

Scritto da Gabriel

Le conseguenze della qualità nell'ingegneria del software sono le seguenti:

- Miglioramento della soddisfazione dei clienti: Un software di alta qualità che soddisfa le esigenze degli utenti e delle parti interessate ha maggiori probabilità di essere ben accolto dai clienti.
- Riduzione dei costi di manutenzione: Un software di alta qualità, ben progettato e facile da mantenere, richiederà meno risorse per continuare a funzionare nel tempo.
- Aumento della competitività: Un software di alta qualità può dare a un'azienda un vantaggio competitivo rispetto ad altre aziende dello stesso mercato.

Per perseguire la qualità nell'ingegneria del software, sono necessarie una serie di attività. Una di queste è il collaudo, ovvero il processo di valutazione di un sistema o dei suoi componenti con l'intento di scoprire se soddisfa o meno i requisiti specificati. Un modo per ottenere qualità è perseguirla in modo continuato e definito; in particolare, definiamo Sistema qualità come insieme di procedure atte al perseguimento di tali obiettivi, secondo un piano che ne fissa gli obiettivi (Piano di qualità), in senso di impegno. Successivamente viene stabilito, a monte del metodo di lavoro/way of working un Controllo di qualità, per controllarne gli effetti ed accertarne il funzionamento e specializzare il miglioramento continuo, basato sull'evidenza. Tutto ciò, nell'ambito di progetto didattico, è esplicitato dal Piano di Qualifica.

Una buona pratica è quella di scrivere casi di test prima di scrivere qualsiasi codice; questo approccio è chiamato Test-Driven Development (TDD). Grazie a questa pratica, lo sviluppatore avrà una visione chiara di ciò che ci si aspetta dal software e si assicurerà che il software si comporti come previsto.

Nel mio progetto, ho constatato che l'implementazione delle pratiche di test ha migliorato notevolmente la qualità del software prodotto dal mio gruppo. L'uso del TDD, in particolare, ha aiutato i nostri componenti a pensare in modo più critico ai requisiti e alla progettazione del loro codice, il che ha portato a un software più ben progettato e privo di bug. Inoltre, l'esperienza con i test ci ha aiutati a capire l'importanza di disporre di metriche precisamente adeguate a rilevare e mettere a cruscotto una serie di osservazioni sul prodotto finale da consegnare.

- *Immaginando di redigere un glossario per un documento formale esterno, fornire definizioni concise, efficaci e valevoli nel dominio dell'ingegneria del software, dei termini: processo, attività, fase. Indicare fonti bibliografiche autorevoli a supporto delle definizioni fornite.*

Immaginando di redigere un glossario, possiamo determinare precisamente questi termini:

- Processo: Insieme di attività/azioni definite con l'obiettivo di ottenimento di specifici obiettivi. Essi vengono usati per guidare un insieme di attività normate dal way of working, al fine di controllare lo sviluppo, manutenzione, gestione documentale e dei singoli processi, per specifiche e configurazione. In particolare, distinguiamo varie categorie di processo: base, supporto e di organizzazione. Il processo può ricorrere in fasi differenti, intervallati da una serie di attività. Si cita lo standard ISO/IEC 12207 sul Software Lifecycle Processes, oppure anche modelli come CMMI (Capability Maturity Model Integration), al fine di misurare adeguatezza dei processi, coerenza, valutazione e successiva integrazione di ognuno
- Attività: Nell'ingegneria del software, un'attività è un compito o un'azione specifica che viene eseguita come parte di un processo o di un progetto più ampio. Le attività possono essere scomposte in compiti più piccoli e sono tipicamente utilizzate per organizzare e gestire il lavoro di un team di sviluppo software. Esempi di attività in un processo di sviluppo software possono essere la raccolta dei requisiti, la progettazione, la codifica, il collaudo e la distribuzione.
- Fase: Nell'ingegneria del software, una fase è uno stadio o un passaggio distinto di un processo che si concentra sul raggiungimento di obiettivi o risultati specifici. Le fasi sono spesso utilizzate per strutturare lo sviluppo di un sistema software, con ogni fase che si basa sul lavoro completato nella fase precedente. Le fasi più comuni nello sviluppo del software comprendono l'analisi dei requisiti,

la progettazione, l'implementazione, il collaudo e la manutenzione. In un periodo, sono presenti più fasi, in cui eseguire molteplici attività in un modo continuo ed incrementale

- *Fornire una breve definizione del cosiddetto ciclo di Deming (anche noto come PDCA) e discutere concisamente se e come i suoi principi siano stati applicati nel proprio progetto didattico.*

Il ciclo di Deming è un modello studiato per il miglioramento continuo della qualità in un'ottica a lungo raggio. Serve per promuovere una cultura della qualità che è tesa al miglioramento continuo dei processi e all'utilizzo ottimale delle risorse. Questo strumento parte dall'assunto che per il raggiungimento del massimo della qualità sia necessaria la costante interazione tra ricerca, progettazione, test, produzione e vendita. Per migliorare la qualità e soddisfare il cliente, le quattro fasi devono ruotare costantemente, tenendo come criterio principale la qualità.

• P - Plan. Pianificazione.

• D - Do. Esecuzione del programma, dapprima in contesti circoscritti.

• C - Check. Test e controllo, studio e raccolta dei risultati e dei riscontri.

• A - Act. Azione per rendere definitivo e/o migliorare il processo. Nel progetto didattico i principi del PDCA sono stati attuati per garantire una elevata qualità del prodotto, si sono quindi alternate in continuazione le attività di pianificazione come l'analisi dei requisiti, scelta del framework, gestione delle ore e del personale mediante piano di progetto, definizione delle norme di progetto si sono messe in atto le pianificazioni in base alla fase di lavoro (RR/RP/RQ/RA) suddivise in piccole sotto fasi controllando ogni azione e risultato per catalogare e risolvere i problemi riscontrati, i problemi trovati sono stati quindi stati registrati nel piano di progetto nei vari consuntivi di fase e discusse all'interno del team per alzare il livello qualitativo

- *Delineare una metodologia atta a trasporre i requisiti utente, espressi esplicitamente o implicitamente in un capitolato d'appalto, nei requisiti software da assumere in una proposta tecnica di fornitura. Indicare i principali obiettivi che tale metodologia debba proporsi.*

Una metodologia per trasporre i requisiti dell'utente da un capitolato d'appalto in requisiti software per una proposta tecnica di fornitura è nota come metodologia di "Requirements Elicitation and Analysis". Gli obiettivi principali di questa metodologia sono comprendere e documentare chiaramente i requisiti dell'utente e garantire che tali requisiti siano accuratamente riflessi nella proposta di fornitura tecnica.

Le fasi principali di questa metodologia sono le seguenti:

- Esaminare il capitolato d'appalto: Leggere e comprendere attentamente il documento di capitolato d'appalto, prestando attenzione a qualsiasi requisito esplicito o implicito. Individuare eventuali ambiguità o incongruenze che potrebbero richiedere un chiarimento.
- Identificare le parti interessate: Identificare tutte le parti interessate, compresi l'utente, il cliente, gli utenti finali e tutte le altre parti che hanno un interesse nel sistema software da sviluppare. Questo aiuterà a garantire che i requisiti di tutte le parti interessate siano presi in considerazione.
- Raccogliere i requisiti: Utilizzare il più possibile brainstorming ed analisi ad ampio spettro per raccogliere i requisiti dalle parti interessate. Occorre assicurarsi che i requisiti siano catturati in un formato strutturato, come user stories, casi d'uso o requisiti funzionali.
- Analizzare e convalidare i requisiti: Analizzare i requisiti per verificarne la completezza, la coerenza e la fattibilità. Convalidare i requisiti ottenendo il feedback delle parti interessate e apportare le revisioni necessarie.
- Specificare i requisiti del software: Tradurre i requisiti in una serie di requisiti software che guideranno lo sviluppo del sistema. Questi requisiti devono essere specifici, misurabili, raggiungibili, pertinenti e limitati nel tempo (SMART).

Scritto da Gabriel

- Definire le priorità dei requisiti: Privilegiare i requisiti in ordine di importanza, in modo che i requisiti più critici siano affrontati per primi.
- Comunicare e verificare i requisiti: Comunicare i requisiti del software a tutte le parti interessate e ottenere la loro approvazione. Verificare che i requisiti siano coerenti con le specifiche dell'offerta e che possano essere soddisfatti entro i vincoli del progetto.

Gli obiettivi principali di questa metodologia sono garantire che i requisiti dell'utente siano chiaramente compresi e accuratamente riflessi nella proposta tecnica di fornitura. Permette di coprire le esigenze dell'utente, coinvolgendolo nel processo e comprendendo il contesto di utilizzo. Inoltre, consente di ottenere una comprensione condivisa dei requisiti tra tutte le parti coinvolte e di garantire che il sistema software sviluppato soddisfi le esigenze dell'utente e delle parti interessate e sia coerente con le specifiche di gara.

- *Inquadrare la pratica nota come "continuous integration" nel dominio dell'ingegneria del software e illustrare concisamente alcuni dei metodi e degli strumenti che consentono di attuarla. Ove possibile, rapportare tali considerazioni all'esperienza personale guadagnata nella loro applicazione.*

La continuous integration è una pratica dell'ingegneria del software che mira a costruire, testare, consegnare software in modo continuo ed automatizzato. Questo si realizza con iterazioni di avanzamento continue che portano a migliorare progressivamente, assicurandosi che ogni cambiamento sia utile e non introduca alcuna regressione, ma anzi integri cambiamenti e applichi testing frequentemente e in modo immediato.

I principali metodi e strumenti che consentono l'implementazione della CI sono:

- Sistema di controllo delle versioni (VCS): Un VCS, come Git, viene utilizzato per gestire il codice sorgente del progetto software e per tenere traccia delle modifiche apportate dai diversi sviluppatori. Ciò consente agli sviluppatori di lavorare contemporaneamente sulla stessa base di codice e facilita l'integrazione delle loro modifiche al codice.
- Build automatiche: Gli strumenti di compilazione automatica, come Jenkins, Travis CI o CircleCI, vengono utilizzati per compilare il codice sorgente, eseguire test e generare codice eseguibile, in modo che gli sviluppatori possano essere sicuri che le loro modifiche al codice non interrompano la compilazione. La build garantisce la composizione di un insieme di parti (Configuration Items) secondo una specifica organizzazione, sia essa gerarchica o meno.
- Test continui: I test automatizzati vengono eseguiti in modo continuo sul codice compilato, in modo che gli sviluppatori possano individuare e correggere i bug non appena vengono introdotti. Questi test possono includere test unitari, test di integrazione, test di accettazione e così via.
- Strumenti di analisi e qualità del codice: Questi strumenti sono utilizzati per valutare la qualità della base di codice, identificare gli odori del codice e le aree di debito tecnico, come SonarQube, che permette di rilevare le violazioni delle best practice e i problemi di manutenibilità.
- Automazione della distribuzione: Per garantire che il software venga distribuito in modo rapido, sicuro e coerente, si possono usare strumenti come Ansible, Puppet, Chef o Salt Stack per distribuire il software in ambienti diversi, automaticamente e con un intervento umano minimo.

In base alla mia esperienza personale, la pratica di integrazione è particolarmente vantaggiosa nei progetti in cui più sviluppatori lavorano su parti diverse della base di codice e consente un ciclo di sviluppo più efficiente e rapido. Utilizzando un VCS come Git, siamo stati in grado di tracciare facilmente le modifiche e di unirle, mentre l'uso di strumenti di compilazione e test automatizzati ci ha aiutato a individuare tempestivamente i problemi e a risolverli prima che potessero diventare un problema. Inoltre, l'uso di strumenti di analisi del codice come SonarQube ci ha aiutato a mantenere la base di codice sana e a identificare le aree che richiedevano miglioramenti.

Scritto da Gabriel

- *Facendo riferimento allo standard ISO/IEC 12207, discutere la differenza di obiettivi, attività coinvolte, strategie di conduzione e strumenti, tra i processi di verifica e validazione.*

La verifica viene effettuata sui prodotti di ogni singola attività, è di interesse solamente interno all'azienda. Viene gestita ed effettuata dai verificatori di progetto e consiste nel controllo sistematico, disciplinato e misurabile dei prodotti di ogni attività (come documenti e/o codice). Tale controllo può essere effettuato a "pettine" (walkthrough) con impiego di tempo e costi maggiori (ma necessari se l'azienda non ha ancora esperienza sufficiente) o a "campione" (inspection), controllando cioè solo le parti più critiche (ovvero più inclini ad errori) del prodotto. Il ciclo PDCA è uno dei metodi per applicare la verifica, in quanto comporta controlli frequenti e cicli, con conseguente miglioramento del prodotto che sia rispondente nelle aspettative iniziali del cliente. Tale processo si svolge controllando che ogni requisito formulato in analisi sia stato effettivamente e completamente soddisfatto. Per fare ciò, una componente fondamentale la svolgono i test funzionali, che appunto verificano le funzionalità del prodotto e il collaudo del cliente. Per facilitare questo processo è essenziale il tracciamento dei requisiti verso le componenti del prodotto.

- *Descrivere la tecnica di classificazione e tracciamento dei requisiti adottata nel proprio progetto didattico e le corrispondenti procedure, manuali o automatiche. Spiegare brevemente le scelte effettuate e discutere l'efficacia e i limiti riscontrati nella loro applicazione*

Per la classificazione dei requisiti, il team ha adottato varie tecniche, tra cui interviste con il cliente, discussioni creative di tipo "Brainstorming" (approccio maieutico), osservazione di comportamenti ed esigenze, ricerca su Internet di studi appropriati al nostro progetto. La tecnica più utilizzata è stata il Brainstorming, per cui il team ha utilizzato parte delle ore per definire e tracciare i requisiti. Il problema di questo approccio deriva dalla mancanza di esperienza di ciascun membro del team. Ci siamo quindi trovati di fronte a problemi ipotizzati dalla logica e da studi mirati all'occasione, dandoci un tempo di parola e mettendo a fuoco ogni idea, essendo un'analisi iniziale e di ampio spettro, mirata a non farsi sfuggire nessun dettaglio ma anzi pensare e agire in prospettiva di prodotto. La non piena comprensione è un attività di rischio preventivata dalle stesse norme progettuali. Ogni attività di scelta è automatizzabile considerando classificazione e tracciamento, a livello di limiti, efficacia, risorse ed organizzazione; ciò aiuta a decomporre a basso livello il problema, a fine di renderlo comprensibile nel modo corretto all'analisi.

- *Assegnare, giustificando la scelta, un peso percentuale di impegno da dedicare alle attività di verifica in un contesto di lavoro paragonabile al progetto didattico per numerosità di partecipanti e ore di lavoro mobilizzate. Ripartire tale quantità proporzionalmente tra le specifiche attività di verifica utilizzabili dal processo di sviluppo. Indicare quali di esse possono o debbano essere automatizzate, e come.*

La percentuale di impegno dedicata alle attività di verifica in un progetto di sviluppo software dovrebbe essere direttamente proporzionale alle dimensioni e alla complessità del progetto. In un contesto simile al progetto didattico da lei citato, con un numero ridotto di partecipanti e un numero limitato di ore di lavoro mobilitate, consiglierei di destinare almeno il 40% dello sforzo complessivo alle attività di verifica.

Le attività di verifica specifiche che possono essere utilizzate dal processo di sviluppo e le relative allocazioni sono:

- Test unitari: 15% - I test unitari vengono utilizzati per verificare singole unità o componenti del software. Questo tipo di test è solitamente automatizzato e può essere eseguito utilizzando un framework di unit testing come JUnit.
- Test di integrazione: 10% - I test di integrazione sono utilizzati per verificare le interazioni tra le diverse unità o componenti del software. Anche questo tipo di test può essere automatizzato, utilizzando strumenti come Selenium.

Scritto da Gabriel

- Test funzionali: 10% - I test funzionali servono a verificare il software rispetto ai requisiti funzionali. Anche questo tipo di test può essere automatizzato, utilizzando strumenti come Cucumber o TestComplete.
- Test delle prestazioni: 5% - Il test delle prestazioni serve a verificare le prestazioni del software in diverse condizioni di carico. Anche questo tipo di test può essere automatizzato, utilizzando strumenti come Apache JMeter o LoadRunner.
- Test di accettazione: 10% - Il test di accettazione serve a verificare il software rispetto ai criteri di accettazione e di solito viene eseguito dall'utente finale o dal cliente. Questo tipo di test di solito non è automatizzato, in quanto si concentra principalmente sull'esperienza dell'utente e sull'accettazione del software, ma gli scenari di test possono essere automatizzati.
- Test di sicurezza: 10% - I test di sicurezza sono utilizzati per verificare il software contro le minacce e le vulnerabilità di sicurezza note. Questo tipo di test di solito non è automatizzato, in quanto richiede un test manuale, ma alcuni test di sicurezza possono essere automatizzati utilizzando strumenti come Nessus o OpenVAS.

Vale la pena notare che, oltre ai tipi di test sopra citati, sono necessarie anche altre attività non funzionali, come il test in ambienti diversi, il test su dispositivi diversi e il test di accessibilità. La percentuale assegnata a queste attività dovrebbe rientrare nel 40% delle attività di verifica e il loro livello di automazione può variare a seconda dei requisiti specifici del progetto.

In conclusione, l'allocazione dello sforzo per le attività di verifica deve essere adattata al contesto e ai requisiti specifici del progetto. L'obiettivo dovrebbe essere quello di automatizzare il maggior numero possibile di attività di verifica, soprattutto quelle ripetibili, prevedibili e che non richiedono l'intervento umano.

- *Fissato il ruolo di amministratore di progetto all'interno di un organigramma tipo di progetto, descriverne le mansioni principali rispetto a un normale ciclo di sviluppo. Delineare le competenze richieste da tale ruolo e, facendo riferimento alla propria esperienza di progetto didattico, ipotizzare come esse possano essere acquisite efficacemente in funzione dei vincoli derivanti dal piano di progetto e dal piano di qualifica*.

In una tipica organizzazione di progetto, l'amministrazione è data da un insieme di doveri tali da essere ricoperte da una persona, cosiddetta amministratore di progetto. In generale mantiene varie attività secondo un certo numero di persone e ore disponibili, considerando il numero al fine di renderlo equo e l'asincronia data dagli impegni e indisponibilità di ciascuno. In particolare, si devono:

- coordinare incontri e schedulare gli impegni di calendario
- organizzare le best practices da seguire nelle singole attività
- gestire le risorse presenti e gli strumenti per garantire la corretta funzionalità
- comunicare con gli stakeholder

Le abilità sono acquisite durante lo stesso processo di progetto, da un punto di vista organizzativo, di comunicazione e risoluzione dei problemi, nonché l'informatizzazione e automatizzazione di tutti i processi produttivi. L'apprendimento, spesso limitato a livello di tempo, deve essere effettuato nella costante pratica di processi di sviluppo e considerando l'incremento della qualità del piano di qualifica, integrando a livello temporale di calendario l'attività di autoapprendimento con il piano di progetto presente.

- *Illustrare qualche best practice personalmente sperimentata o riconosciuta come tale a posteriori, per il buon svolgimento delle attività di analisi dei requisiti*

Al fine di svolgere correttamente l'analisi dei requisiti, è indispensabile svolgere il più possibile attività di analisi, coinvolgendo gli stakeholders il più possibile con domande e riunioni e secondo un processo di

brainstorming svolto internamente al gruppo di progetto didattico, con sessioni di confronto aperto volto a dissezionare il più possibile ogni anfratto del capitolato ed espresso in modo chiaro e ben fatto. Occorre quindi documentare adeguatamente tutti i requisiti, tali da comprendere e seguire tutte le attività, evitando confusione e malintesi. È inoltre utile verificare e validare i requisiti, utilizzando strumenti precisi di tracciamento per poterli visualizzare facilmente, continuando ad espanderli e adattarli. Il modo migliore per esprimerli è attraverso gli UML, documentando in modo semplice, modulare e dettagliato le user story come funzionalità, separando tutto ciò che occorre nel modo migliore (soprattutto, scomponendo in modo atomico, facilmente verificabile e modificabile nel corso del tempo).

- *Fornire ragioni di metodo e di strategia a supporto della scelta del modello di sviluppo incrementale. Alla luce della propria esperienza nel progetto didattico, indicare le condizioni che devono valere per poter applicare fedelmente quel modello, e i fattori di rischio che lo possono far deragliare*

Il modello di sviluppo incrementale è una metodologia di sviluppo del software che prevede la costruzione progressiva del sistema attraverso l'aggiunta di funzionalità incrementali e produrre valore ad ogni iterazione, prevedendo una serie di piccole parti utili a ridurre il rischio di fallimento.

Ragioni di metodo a supporto della scelta del modello di sviluppo incrementale sono:

- Ottenere un feedback precoce dall'utente sulla funzionalità del sistema, che può essere utilizzato per migliorare il progetto in corso di sviluppo.
- Gestire meglio i rischi del progetto, poiché solo una piccola parte del sistema è in fase di sviluppo alla volta.
- Adattare il progetto alle modifiche dei requisiti in modo più flessibile rispetto ai modelli "big-bang", che mettono tutto insieme “di colpo”; qui la base è integrare poco per volta in modo continuo (continuous integration)
- Il modello incrementale consente di fornire una soluzione parziale al cliente in tempi brevi, andando ad evolvere e adattare gradualmente con successivi incrementi ed evoluzioni, versioni e prototipi da realizzare poco per volta e adattandosi gradualmente

Per poter applicare fedelmente il modello di sviluppo incrementale, le seguenti condizioni devono valere:

- I requisiti del sistema devono essere divisi in incrementi gestibili.
- Buona comprensione dei requisiti del sistema e delle loro priorità
- Buona capacità di pianificazione e di gestione del progetto.
- Buona capacità di comunicazione con il cliente e di gestione delle aspettative.

I fattori di rischio che possono far deragliare il modello di sviluppo incrementale sono:

- Cambiamenti significativi nei requisiti del sistema che possono causare una riprogettazione delle funzionalità già sviluppate.
- Problemi di gestione del progetto, come la mancanza di una buona pianificazione o la mancanza di una buona comunicazione con il cliente.
- Problemi di gestione delle risorse, come la mancanza di personale qualificato o la mancanza di budget adeguato.
- Complessità nella gestione, nella stima delle attività e richiede frequente comunicazione

- *Un elemento delle strategie di pianificazione di progetto concerne la ripartizione percentuale (quindi non la quantità ma la proporzione) delle risorse umane (ore/persona) disponibili sulle attività da svolgere. Discutere i criteri che avete utilizzato al riguardo nel progetto didattico, presentare le scelte fatte e valutarle criticamente alla luce di quanto appreso allo stato.*

La pianificazione nelle attività di progetto considera normalmente i seguenti fattori:

1. Importanza delle attività: le attività che sono più importanti per il successo del progetto ricevono una maggiore percentuale di risorse umane rispetto alle attività meno importanti.
2. Difficoltà delle attività: le attività che sono più complesse o che richiedono più tempo per essere completate ricevono una maggiore percentuale di risorse umane rispetto alle attività più semplici.
3. Dipendenze tra attività: le attività che sono strettamente dipendenti l'una dall'altra ricevono una maggiore percentuale di risorse umane rispetto alle attività che possono essere completate in modo indipendente.

Normalmente, al fine di capire al meglio il peso delle attività, è necessario avere alla base una buona organizzazione e capire possibilmente con il gruppo stesso l'importanza della stessa pianificazione. Ad esempio, le attività di documentazione sono attività svolgibili individualmente, frammentando i compiti in precise responsabilità (qualora si abbia alle spalle un gruppo serio, ndr). Ritengo infatti questa sia svolta bene individualmente se le persone sanno ciò che fanno e hanno competenza; è bene allocare un buon numero di risorse a livello di tempo e risorse nella fase di codifica e testing, meno alla documentazione, essendo attività più ignote e meno precise. La documentazione prevede la strada per le successive due, mentre le altre, a seconda della complessità del codice in sé e del rappresentare i singoli requisiti, può essere più o meno facile.

- *Indicare, giustificandolo, il peso percentuale di impegno che ritenete sia da dedicare alla verifica in un contesto di lavoro paragonabile al vostro progetto didattico. Ripartire tale quantità proporzionalmente tra le specifiche attività di verifica attraverso le varie fasi dello sviluppo. Indicare quali di tali attività possano o debbano essere automatizzate, e come.*

L'attività di verifica è un processo che deve essere implementato in modo continuativo all'interno di un progetto, dato che assicura il buon svolgimento delle azioni. Esso dà evidenza concreta del fatto che, in quel singolo periodo, le fasi portano alla soddisfazione dei singoli requisiti, assicurando una correttezza per costruzione e assicurandosi di non introdurre ulteriori errori. Essa prevede per la sua interezza sia il più possibile automatizzabile, aumentando la copertura e migliorando la qualità del prodotto finale. Per quanto riguarda le singole fasi:

- La fase di analisi richiede un'attività di verifica proporzionale alla creazione dei singoli documenti, in quanto ogni passaggio deve essere adeguatamente affinato per evitare difformità nella prosecuzione delle attività di progetto. Qui si ha principalmente analisi statica e si definiscono le metriche in profondità per come dovranno essere precisamente usate successivamente, tale da prevedere sempre la buona riuscita del prodotto software
- La fase di progettazione, essendo implementation, verifica la corretta applicazione dei requisiti ed esamina la loro concreta applicazione, affinché il codice sia considerato corretto e sia funzionante nel modo previsto. L'impiego delle attività di verifica, anche qui, è costante ma in proporzione maggiore, essendo che durante questa fase si utilizzano tante categorie di test che saranno ampiamente riprese e riutilizzate in modo continuativo. La qualità viene garantita in modo continuativo (quality assurance). Le metriche definite
- La fase di collaudo richiede la buona riuscita della verifica e la conferma delle caratteristiche del software rispetto ai requisiti utenti, come stabilito dalla validazione. Le varie tecniche di verifica fino ad ora implementate giungono a maturazione definitiva (legge del rendimento decrescente), tale da incontrare un punto dove i test non sono più necessari

Scritto da Gabriel

- *Durante lo svolgimento del progetto didattico avete sovente confuso la nozione di fase di progetto con quella di attività di progetto (o, meglio ancora, di processo software, inteso come aggregato di attività coordinate e coese). Provate qui a descrivere compiutamente ciascuna delle due nozioni così da evidenziare le differenze e relazioni tra esse.*

La fase si intende come stazionamento in uno stato o transizione del ciclo di vita del SW, definita in base alla metodologia di progettazione utilizzata e possono variare in base al tipo di progetto. Ad esempio, in un progetto software, le fasi di progetto comuni possono includere: la raccolta dei requisiti, la progettazione, la costruzione, la verifica e la manutenzione. Le attività di progetto, invece, si riferiscono alle azioni specifiche che vengono svolte all'interno di ciascuna fase di progetto per raggiungere gli obiettivi del progetto. Ad esempio, all'interno della fase di raccolta dei requisiti, le attività di progetto possono includere la condivisione dei requisiti con gli stakeholder, la creazione di un documento dei requisiti e la verifica dei requisiti.

Il processo software è un concetto più ampio che si riferisce all'insieme di attività coordinate e coese che vengono svolte per sviluppare software. Il processo software include tutte le fasi di progetto e le attività di progetto necessarie per sviluppare software di qualità.

- *Elencare quali attività di progetto software ritenete siano utilmente automatizzabili, in tutto e in parte, provando anche a rapportare i vantaggi che ne deriverebbero con i costi che l'ottenimento di tale automazione potrebbe comportare.*

Ecco alcune attività di progetto software che possono essere utilmente automatizzate, sia in tutto che in parte:

- Test di unità: l'automatizzazione dei test di unità consente di eseguire rapidamente un gran numero di test su singole unità di codice, riducendo il rischio di errori e aumentando la qualità del software.
- Build e deploy: l'automatizzazione del processo di build e deploy consente di ridurre i tempi necessari per la distribuzione del software e di aumentare la qualità del software rilasciato.
- Monitoraggio e gestione delle prestazioni: l'utilizzo di strumenti di automazione per il monitoraggio e la gestione delle prestazioni del software consente di individuare e risolvere rapidamente i problemi, migliorando le prestazioni del software.
- Integrazione continua: l'utilizzo di strumenti di integrazione continua consente di automatizzare il processo di integrazione tra le diverse componenti del software, riducendo i tempi necessari per la distribuzione del software e aumentando la qualità del software rilasciato.
- Analisi dei dati: l'utilizzo di strumenti di automazione per l'analisi dei dati consente di estrarre informazioni utili per migliorare il software, aumentando la qualità del software e rendendolo più adatto alle esigenze degli utenti.

L'automatizzazione di queste attività può comportare dei costi in termini di tempo e denaro per l'acquisizione e la configurazione degli strumenti di automazione e per la formazione del personale, ma i vantaggi che ne derivano sono molti, come ad esempio: aumento della qualità del software, riduzione dei tempi di distribuzione, miglioramento delle prestazioni del software, aumento dell'efficienza e dell'efficacia del processo di sviluppo e maggiore adattabilità del software alle esigenze degli utenti.

- *Fornire una definizione del formalismo noto come "diagramma di Gantt", discuterne concisamente le finalità e modalità d'uso, l'efficacia e i punti deboli eventualmente rilevati nell'esperienza del progetto didattico*

Il diagramma di Gantt è una rappresentazione grafica della pianificazione di un progetto, che mostra le date di inizio e fine di ogni attività e le dipendenze tra le attività. Esso cerca di visualizzare il programma del progetto e di seguire l'avanzamento delle attività nel tempo. Può essere utilizzato per identificare il percorso critico del progetto, ovvero la sequenza di attività che determina la durata del progetto, e per individuare eventuali colli di bottiglia o ritardi.

Per utilizzare un diagramma di Gantt, le attività e le loro dipendenze vengono prima definite ed elencate in una tabella, insieme alle date di inizio e fine e alle risorse necessarie per ogni attività. La tabella viene quindi utilizzata per creare il diagramma di Gantt, che di solito è un diagramma a barre orizzontali con le attività elencate a sinistra e la linea temporale tracciata in basso. Le barre del grafico rappresentano la durata di ogni attività e le eventuali dipendenze tra le attività sono indicate da frecce o linee che collegano le barre.

L'efficacia di un diagramma di Gantt dipende dall'accuratezza e dalla completezza dei dati utilizzati per crearlo. Se le attività e le dipendenze sono definite con precisione e le date di inizio e fine sono realistiche, il diagramma di Gantt può essere uno strumento efficace per monitorare l'avanzamento del progetto e identificare potenziali problemi. Tuttavia, se i dati sono incompleti o imprecisi, il diagramma di Gantt potrebbe non essere un riflesso accurato della pianificazione del progetto.

Nella mia esperienza di progetto didattico, abbiamo usato un diagramma di Gantt integrato in Jira per seguire l'avanzamento dei compiti e identificare eventuali problemi o ritardi. Il diagramma di Gantt è stato efficace nell'aiutarci a visualizzare il programma del progetto e a seguire l'avanzamento delle attività nel tempo. Tuttavia, abbiamo riscontrato alcuni punti deboli nell'utilizzo del diagramma di Gantt, tra cui la difficoltà di stimare con precisione la durata di alcune attività e la necessità di aggiornare continuamente il diagramma man mano che il progetto procedeva e le attività venivano completate. Nonostante queste limitazioni, abbiamo trovato il diagramma di Gantt uno strumento utile per gestire il progetto e assicurarci che rimanesse in linea con i tempi.

- *Illustrare concisamente la strategia di verifica tramite test adottata nel progetto didattico (quali tipi di test, quali obiettivi, quale grado di automazione, ecc.). Alla luce dei risultati ottenuti nel progetto da tali attività, in termini di rapporto costi/benefici, discutere gli spazi di miglioramento rilevati e le eventuali eccellenze raggiunte.*

L'attività di verifica in un progetto didattico si compone di molteplici tipi di attività e test al fine di integrare e rendere ad alto livello nel complesso e specializzarla nei singoli casi:

- Unit test: I test unitari sono piccoli test isolati che si concentrano su singoli componenti o unità di codice. Sono tipicamente automatizzati e vengono utilizzati per verificare la correttezza e l'affidabilità del codice.
- Integration test: I test di integrazione sono test che si concentrano sull'interazione tra diversi componenti o unità di codice. Vengono utilizzati per verificare che i componenti funzionino insieme come previsto e per identificare eventuali problemi che possono sorgere quando vengono combinati.
- System test: I test di sistema sono test che si concentrano sul comportamento e sulla funzionalità del sistema nel suo complesso. Vengono utilizzati per verificare che il sistema soddisfi i requisiti specificati e funzioni come previsto in diversi scenari.
- Test di regressione, accertando che la modifica alle singole parti non modifichi le altre e testa il livello di accoppiamento tra i singoli punti, a livello funzionale e strutturale, verificando a scatola

chiusa il funzionamento corretto e utilizzando le funzionalità nel complesso, controllando la loro correttezza

A livello di progetto didattico, come detto, ciascuno ha la sua utilità e il miglioramento si ottiene scrivendo codice che fa fallire i test e costruire un prodotto progressivamente secondo questo pensiero (test-driven development), testando caso per caso comportamenti e funzionalità e arrivando alla fase di accettazione e collaudo nel modo migliore. Progressivamente, trovano inesattezze e difficoltà, contribuendo a risolvere progressivamente i problemi. In particolare, si vuole progressivamente automatizzare anche il miglioramento, a livello di costi, rendendo efficace ed efficiente il dispendio di risorse astratte/umane. Il miglioramento causa eccellenza quando si è organizzata una strategia di test che copre tutte le funzionalità rilevanti, usando una combinazione di test automatici e manuali (personalizzabili ma prone to error).

- *Alla luce dell'esperienza acquisita nel proprio progetto didattico, discutere concisamente quali strategie di gestione di progetto hanno portato benefici e quali invece – attuate poco o male, o non tempestivamente – hanno causato problemi e difficoltà*

Nel mio progetto didattico, le strategie di gestione di progetto che hanno portato benefici sono state:

- La definizione chiara e condivisa degli obiettivi del progetto, che ha permesso a tutti i membri del team di comprendere il contesto e le aspettative del progetto, e di lavorare in modo efficiente verso gli stessi obiettivi.
- La definizione di un ambiente di lavoro che permetta il facile versionamento delle informazioni e per unione in cloud delle informazioni da usare
- La gestione di un ITS/Issue Tracking System per poter sempre monitorare le attività, i singoli compiti svolti, le persone impiegate e le milestone da raggiungere
- La comunicazione efficace tra i membri del team, che ha permesso di condividere informazioni e di risolvere rapidamente eventuali problemi.

Al contrario, le strategie di gestione di progetto che hanno causato problemi e difficoltà sono state:

- La mancanza di una pianificazione dettagliata delle attività, che ha portato a problemi di gestione delle risorse e di consegna in ritardo.
- La mancanza di una gestione attiva del rischio, che ha portato a problemi non previsti e alla difficoltà di gestirli. Ogni attività va preparata per tempo e a basso livello, individuando facilmente problematiche e difficoltà che possono emergere “lungo la strada”.

In generale, è importante che tutte le strategie di gestione di progetto vengano adottate tempestivamente e che vengano adattate alle esigenze del progetto e del team. Inoltre, è importante che ci sia una costante monitoraggio e revisione delle strategie per garantire che il progetto proceda come previsto e che eventuali problemi vengano gestiti in modo tempestivo.

- *Discutere almeno 2 (due) varianti della nozione di "configurazione" di prodotto/sistema. Indicare se e quale relazione intercorra tra "configurazione" e "versionamento" e specificare i relativi obiettivi.*

La configurazione intende un processo continuativo che decide come combinare le singole parti, crearle indipendentemente e mantenerle in modo scalabile e separato, integrando un controllo sul come queste parti cambino e in che modo si definisca questo tipo di evoluzione, anche storicamente/temporalmente, con un insieme di transizioni regolate precisamente. L’intero processo di cambiamento prefigura il sistema; la configurazione di sistema intende la stessa cosa, ma in modo più ampio, scalando in base al software e identificando i cambiamenti e individuando metriche di confronto a livello dell’intero sistema, da cui potenzialmente dipende anche il prodotto. Le metriche individuano, a livello sistema, come un prodotto debba determinarsi, per poter effettuare con successo dei cambiamenti efficaci (change management):

In entrambi i casi, si spinge il software in segmenti nuovi del suo ciclo di vita, dove i periodi sono suddivisi in fasi e ogni attività è decomposta in parti configurabili e viste singolarmente come i configuration item, messe insieme tramite le build.

Il versionamento intende fondamentalmente lo stesso, nel senso del tracciare le modifiche, spesso con linee di lavoro in parallelo e automatico, risalendo sempre alle decisione, alla cronistoria dei cambiamenti e alle linee guida/standard che devono essere seguiti nel processo di cambiamento/evoluzione. Il fatto è che la configurazione riguarda principalmente l'insieme di impostazioni e di componenti utilizzati durante la fase di sviluppo, mentre il versionamento riguarda la gestione delle versioni del software.

- *Si può ipotizzare che esista una relazione tra le attività di "validazione" e quelle di "tracciamento dei requisiti". Discutere la natura di tale eventuale relazione e le tecniche che avete utilizzato nel progetto didattico per effettuare il tracciamento, analizzandone criticamente l'efficacia e i limiti riscontrati.*

La validazione e il tracciamento dei requisiti sono due attività strettamente correlate nella gestione di un progetto software. La validazione è il processo di verificare che i requisiti soddisfino le esigenze del cliente e che il prodotto sia conforme alle specifiche. Il tracciamento dei requisiti, d'altra parte, è il processo di monitorare e gestire i cambiamenti nei requisiti durante tutto il ciclo di vita del progetto e verificarne la corretta implementazione, guardando in avanti (definendo i requisiti, il loro tipo e la loro importanza) e indietro (per essere sicuri di averli individuati tutti al meglio).
In entrambi i casi, si parla di attività necessarie, sufficienti, non ambigue e il più possibile complete.

Nel mio progetto didattico, ho utilizzato diverse tecniche per effettuare il tracciamento dei requisiti, tra cui:

- La registrazione dei requisiti in una specifica formale: ho utilizzato un formato standard per registrare i requisiti, che ha permesso di identificare chiaramente le esigenze del cliente e di verificare che fossero soddisfatte durante il processo di validazione.
    - Il linguaggio usato sono gli UML e si cerca di andare ad un livello di dettaglio modulare, quindi individuando ogni funzionalità come caso d'uso e ramificando il più possibile ogni cosa sotto forma di moduli
- La tracciabilità: ho utilizzato un sistema di tracciabilità per monitorare i cambiamenti nei requisiti e per verificare che tutti i requisiti siano stati soddisfatti durante il processo di validazione.
    - Questo viene fatto sia integrando tale specifica nei documenti di progetto che, all'atto pratico, verificando che le funzionalità come moduli individuati dai casi d'uso corrispondano a quanto visto e presente in implementazione
- La revisione dei requisiti: ho coinvolto tutte le parti interessate nel processo di revisione dei requisiti, per garantire che tutti i requisiti fossero completi, coerenti e soddisfacenti.
    - Sia prima che dopo, fondamentale coinvolgere il più possibile il proprio gruppo e gli stakeholder, al fine di tracciare prima i requisiti e assicurarsi che tutto sia implementato al meglio in termini di casi e funzionalità.

L'efficacia di queste tecniche è stata generalmente buona, poiché ha permesso di gestire efficacemente i requisiti e di verificare che fossero soddisfatti durante il processo di validazione. Tuttavia, ci sono stati alcuni limiti nell'utilizzo di queste tecniche, tra cui:

- La formazione dei membri del team su come utilizzare le tecniche di tracciamento dei requisiti.
- La necessità di una forte collaborazione e comunicazione tra i membri del team per garantire che tutti i requisiti siano stati soddisfatti e registrati.
- La necessità di un sistema di tracciabilità adeguato a monitorare e gestire i cambiamenti nei requisiti.

Scritto da Gabriel

In generale, il tracciamento dei requisiti è una attività fondamentale per garantire che i requisiti siano soddisfatti e per garantire che i cambiamenti nei requisiti siano gestiti in modo efficace. Tuttavia, per ottenere il massimo beneficio da queste tecniche, è importante che siano adeguatamente supportate da una buona comunicazione e collaborazione tra i membri del team, e da un sistema di tracciabilità adeguato.

- *Illustrare gli obiettivi dei "test" di: (a) unità, (b) integrazione e (c) sistema, specificando quali siano per ciascuno di essi: (1) gli oggetti del test, (2) gli ingressi, (3) le uscite e (4) le attività da svolgere.*

Iniziamo col dire che un test è un insieme di fasi che prevede la verifica del corretto funzionamento e di rispetto dei requisiti iniziale del software, assicurandosi sia privo di difetti. Ora distinguiamo come richiesto le tipologie:

- Di unità, in cui l'oggetto sono le singole unità di codice (funzioni/metodi), avendo come input e come output quelli generati dalle stesse unità. Essi verificano tutte le unità e sono facilmente ottimizzabili ed automatizzabili. Ne esistono di diverso tipo:
    - Black-box, facendo solo riferimento alla specifica e usando classi di equivalenza per classificare i singoli input rispetto ad I/O attesi
    - White-box, verificando la logica interna del codice al fine di provarne la correttezza dei vari cammini di esecuzioni

  Sono solitamente di tipo dinamico, eseguendo l'unità in vari modi ed osservandone i risultati sulla base della scrittura di casi di test (si valuta la correttezza)
- Di integrazione, verificando che le singole componenti del sistema funzionino correttamente insieme e che questo comporti coesione (le funzionalità si completano in modo utile), verificando la compatibilità. Questa attività è per costruzione e si verifica con delle build, prendendo come ingressi quelli delle singole unità e automatizzando il processo di build per rilevare facilmente malfunzionamenti nei componenti integrati, trovando in output il risultato di tale composizione
- Di sistema, eseguendo una validazione che consiste nel verificare che tutto funzioni come previsto, in un'interazione complessiva (precede il collaudo e verifica che tutti i requisiti siano nel complesso stati coperti secondo la requirements coverage). Ingressi sono gli input con i quali ci si interfaccia nello specifico caso dell'interazione col sistema, output è il risultato di tale interazione. Nel complesso, si considerano tutti i comportamenti, messi insieme opportunamente dalle categorie di test precedenti.

- *Fornire una definizione dei seguenti termini e ciò a cui essa corrisponde in un linguaggio di programmazione ad alto livello (specificando quale): "Modulo", "Unità", "Componente"*

In un linguaggio di programmazione ad alto livello, i termini "modulo", "unità" e "componente" possono avere significati simili ma leggermente diversi.

- Un modulo è un componente software o una parte di un programma che contiene una o più routine. Uno o più moduli sviluppati in modo indipendente costituiscono un programma. Un'applicazione software di livello aziendale può contenere diversi moduli, ognuno dei quali serve per operazioni aziendali uniche e separate. Normalmente, ciascuno di questi ha una serie di funzionalità, offerte singolarmente dalle unità.
- Un'unità è una singola funzione o un piccolo gruppo di funzioni che svolgono un compito specifico. Esse sono raccolte come molteplici all'interno dei moduli.
- Un componente è una parte di un programma che può essere utilizzata in modo autonomo e che può essere riutilizzato in altre parti del programma. Ciò richiede la separazione tra singole unità/interfacce, affinché non si dipenda dall'implementazione ma anzi ne si implementino

funzionalità in modo combinato, sapendo che le componenti sono messe insieme al fine di offrire, con moduli decomponibili a loro volta in unità, una serie di possibilità a chi le utilizza.

- *Discutere le differenze informative tra il "diagramma di PERT" e il “diagramma di Gantt”.*

Il diagramma di PERT è una rappresentazione grafica dei task e delle dipendenze tra essi all'interno del progetto. Il diagramma rappresenta i task come nodi e le dipendenze come archi tra i nodi. Il diagramma PERT è utile per visualizzare la struttura logica del progetto e per identificare i task critici che possono influire sulla durata del progetto. In particolare, si cerca di individuare i cammini minimi per arrivare al buon funzionamento delle singole attività

Il diagramma di Gantt, invece, è una rappresentazione grafica dei task in relazione al tempo. Il diagramma mostra i task come barre orizzontali su un asse temporale. Il diagramma di Gantt è utile per visualizzare la programmazione del progetto e per identificare eventuali ritardi o risorse insufficienti. Esso è uno strumento potente per la sua estrema semplicità.

In generale, entrambi i diagrammi possono essere utilizzati insieme per fornire una visione completa dello stato del progetto e per supportare la gestione del progetto.

- *Discutere la relazione tra encapsulation e information hiding, esaminando criticamente l'affermazione: "encapsulation is a programming language feature; information hiding is a design principle".*

I due concetti sono molto legati alla programmazione orientata agli oggetti.

L'Encapsulation è una caratteristica del linguaggio di programmazione che consiste nel nascondere i dettagli di implementazione di un oggetto dietro un'interfaccia pubblica. In questo modo, gli utenti dell'oggetto possono utilizzare i suoi servizi senza conoscere i dettagli della sua implementazione. In altre parole, l'encapsulation è il meccanismo per proteggere i dati interni dell'oggetto dall'accesso e dalla modifica da parte degli utenti.

Information hiding è un principio di progettazione che consiste nel nascondere i dettagli di implementazione di un oggetto al fine di proteggere la sua integrità e di semplificare la sua interfaccia pubblica. In altre parole, l'information hiding è l'approccio per progettare gli oggetti in modo che siano facili da usare e da mantenere, nascondendo i dettagli di implementazione non necessari per l'uso dell'oggetto.

In sintesi, l'encapsulation è una caratteristica del linguaggio di programmazione, mentre l'information hiding è un principio di progettazione. L'encapsulation fornisce i meccanismi per realizzare l'information hiding, ma non è sufficiente da solo a garantire una buona progettazione.

In generale, l'affermazione che "encapsulation is a programming language feature; information hiding is a design principle" è vera in quanto l'encapsulation è una caratteristica del linguaggio di programmazione, mentre l'information hiding è un principio di progettazione. Tuttavia, è importante notare che l'encapsulation è un meccanismo fondamentale per realizzare l'information hiding, e che i due concetti sono strettamente correlati.

Il fine ultimo è quello di modularizzare, rendendo indipendente l’implementazione dal contesto; come tale, i dettagli vanno nascosti ed utilizzati per costruire le componenti a pezzi, adattando con l’aggiunta di nuovo codice e non di modifica del preesistente, nuove funzionalità; nascondendo dettagli ed informazioni, specializzo quando necessario un modulo affinché abbia una specifica funzione.