UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Scienze Statistiche

Corso di Laurea Triennale in Statistica per le Tecnologie e le Scienze

Relazione finale

Assicurazione della qualità del software: una esperienza in Siav

Relatore
prof. Guido Masarotto
Dipartimento Scienze Statistiche

Correlatore
ing. Auro Rolle
Quality Assurance Siav

Laureando
Aglind Reka
Matricola: 1146857

Anno accademico 2021 – 2022

Sommario

Al giorno d'oggi le applicazioni software sono parte integrante della nostra vita. Basti pensare a ciò che permette il funzionamento non solo di computer o telefoni cellulari, ma anche di automobili, elettrodomestici, giocattoli, sistemi di produzione industriale, ecc. Per questo motivo la produzione di software di qualità e nel minore tempo possibile svolge un ruolo di fondamentale importanza all'interno dell'economia mondiale. Da anni esistono metodologie a supporto della produzione software che guidano sviluppatori e responsabili, lungo tutto il processo di realizzazione di applicazioni e sistemi, dalla raccolta dei requisiti alla distribuzione al cliente. Alcune di queste metodologie promettono una grande agilità nella gestione dei processi di sviluppo software, specie nelle situazioni in cui i requisiti sono incerti o sono facilmente soggetti al cambiamento. Pianificare, conoscere e prevedere sono aspetti chiave per l'abbassamento del rischio di fallimento di un'impresa o di un progetto. Per non incorrere in tali rischi un'azienda moderna deve pertanto ricorrere a diversi modelli, che verranno esposti in questa trattazione. Questa tesi ha l'obiettivo di esporre quella parte del ciclo di sviluppo denominata "Quality Assurance", prendendo come focus principale il caso reale dell'azienda SIAV Spa. L'esposizione si presenta divisa in cinque capitoli.

Nel primo capitolo viene introdotta l'azienda e l'inizio del mio percorso in all'interno di essa.

Nel secondo capitolo viene discusso il concetto di Project Management, mostrando l'evoluzione delle sue pratiche, a partire dal modello waterfall fino ad arrivare alla metodologia Scrum Agile.

Nel terzo capitolo si descrivono in dettaglio le attività di controllo qualità in un'azienda di sviluppo software concentrandosi poi sul particolare caso di SIAV Spa.

Nel quarto capitolo si parla delle varie tipologie di test: unitari, manuali, funzionali e test di performance.

Nel quinto capitolo viene fatto un focus sull'automazione dei test, concentrandosi sul caso di Siav.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

1.1 L'azienda

Siav è un'azienda informatica specializzata nella dematerializzazione, nella gestione elettronica dei documenti e nei processi digitali. Si caratterizza per le competenze specialistiche maturate nella realizzazione di progetti complessi per Aziende ed Enti, e si distingue per le capacità di garantire con risorse proprie le attività di analisi, implementazione e supporto. Fondata nel 1990 a Rubano (Padova) da Alfieri Voltan, attuale Presidente, con più del 20% di quota di mercato Siav è oggi la prima azienda italiana nel settore dell'Enterprise Content Management, e offre software, soluzioni in cloud e servizi di outsourcing per la Gestione Elettronica dei Documenti, il Protocollo Informatico, il Workflow Management, la Fatturazione Elettronica e la Conservazione Digitale. Le soluzioni software e i servizi di Business Process Outsourcing di Siav consentono ad aziende ed enti di gestire l'intero ciclo di vita dei documenti, semplificare i processi e diminuire i costi, tramite soluzioni in house, in full outsourcing o ibride. In particolare, Siav ha consolidato forti competenze verticali nella Fatturazione Elettronica, nella gestione del ciclo passivo e nella Conservazione Digitale. Prodotti di Siav sono Archiflow e Silloge. Archiflow è un software di gestione documentale, disponibile in house o in cloud (SaaS) su architetture Windows. Consente di gestire i documenti e automatizzare i processi aziendali. Archiflow snellisce la gestione dei documenti, semplifica i processi di business, organizza le attività con il workflow, consente l'accesso istantaneo alle informazioni. Grazie alla massima integrabilità e flessibilità, Archiflow consente di creare progetti innovativi di Document & Process Management, anche grazie ad una serie di funzionalità verticali e best practices tramite le quali è particolarmente semplice e

veloce realizzare le più diffuse applicazioni in ambito pubblico (a partire dal Protocollo Informatico) e privato (come la gestione del ciclo delle fatture). Archiflow è il prodotto storico di Siav dove ha ottenuto molto successo negli anni ed ancora è il prodotto principale dell'azienda. Silloge è il prodotto più recente dell'azienda, è un software per la gestione documentale cloud native, quindi compatibile con tutti i cloud provider. Siav sta investendo tantissimo in questo prodotto e si aspetta che raggiunga il livello funzionale di Archiflow in modo da poter far convergere i due prodotti su un'unica soluzione, portando ai clienti un prodotto più dinamico con molte più semplificazioni nella gestione documentale.

Figura 1.1. Logo dell'azienda.

1.2 Esperienza in Siav

La mia esperienza in Siav è iniziata il 15 ottobre 2020, partecipando alla terza edizione di Siav Academy, il percorso formativo dedicato ai giovani talenti. Siav Academy nasce dalla collaborazione tra Umana ed il reparto Human Resources di Siav SpA, avendo come fine ultimo la ricerca, la formazione ed il successivo inserimento di nuove figure in possesso di competenze specialistiche che possano entrare nell'organico di Siav e soddisfare le esigenze dettate dal trend di crescita che da alcuni anni l'azienda sta vivendo. L'Academy mira allo sviluppo di Innovation Developers, figure tecnicamente elevate e in grado di utilizzare un insieme di competenze tecnologicamente avanzate. Per fare ciò si affiancheranno nozioni di carattere tecnico con l'approccio del training on the job grazie al supporto di case study reali, facendo entrare così in contatto i giovani talenti con le attività di delivery e del ciclo di vita del prodotto software. Dopo due mesi di academy, sono stato selezionato per far parte di Siav dove ho cominciato il lavoro il 13 gennaio 2021. Nei primi due mesi all'interno dell'azienda ho continuato a fare formazione e sono stato seguito in tutta questa fase. Alla fine dei due mesi sono stato scelto per fare parte del team Quality Assurance "QA" come Junior Quality Assurance Tester.

Figura 1.2. Siav Academy.

Capitolo 2

Processi e metodologie

2.1 La metodologia Waterfall

In ingegneria del software, il **modello a cascata** o **ciclo di vita a cascata** (waterfall lifecycle) è il più tradizionale modello di ciclo di vita del software. Secondo questo modello, il processo di realizzazione del software è strutturato in una sequenza lineare di fasi o passi, che comprende:

- analisi dei requisiti
- progettazione
- sviluppo
- collaudo
- manutenzione

Questo modello riprende la sequenza di passi tipica della produzione manifatturiera, e fu il primo a essere applicato quando lo sviluppo del software cominciò a essere concepito come attività industriale. Il modello è stato progressivamente abbandonato dall'industria del software, ma rimane un importante riferimento storico. a una gestione del Il metodo Waterfall corrisponde progetto di tipo tradizionale e sequenziale. Si basa su una successione a cascata di fasi distinte dello sviluppo del software documentate, ognuna delle quali generalmente termina prima che inizi la successiva. In tale metodologia il prodotto viene consegnato al cliente dopo la fase di collaudo.

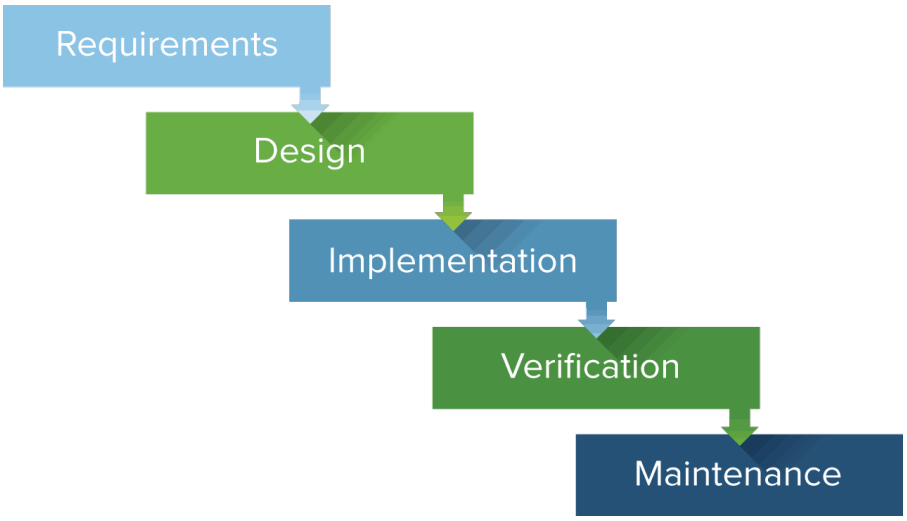

Figura 2.1. Modello waterfall.

2.2 La metodologia Agile

Il metodo Agile fa riferimento ad un tipo di gestione innovativo e iterativo in quanto comprende gli stessi step del metodo Waterfall, ma in chiave iterativa: si lavora su pezzi di requisiti (e non su tutto in un'unica soluzione) rendendo così possibile eseguire simultaneamente le attività di sviluppo e test e di effettuare più deploy durante l'intera vita del progetto. A differenza del metodo a cascata, in cui la pianificazione delle attività è definita all'inizio, nel metodo Agile il tempo è suddiviso in fasi di durata definita (settimane di solito) chiamate 'Sprint'. All'inizio di ogni sprint viene pianificato e definito, con il cliente, un elenco di finalità da consegnare entro la durata del relativo sprint. Se non è possibile completare tutto il lavoro pianificato, il lavoro viene ridistribuito e le informazioni vengono utilizzate per la pianificazione del successivo sprint. Una volta completato, il lavoro può essere rivisto e valutato dal team di progetto e dal cliente, attraverso build giornaliere e dimostrazioni di fine sprint.

2.3 Agile Scrum

Scrum è un metodo di gestione del clclo di produzione che aiuta tutti i soggetti coinvolti nello sviluppo ad organizzarsi attraverso soluzioni adattive per

problemi complessi. Il ciclo di sviluppo è adattativo in funzione di peculiari necessità. Scrum richiede uno Scrum Master per favorire un ambiente in cui:

- Un Product Owner ordina il lavoro, relativo ad un problema complesso, in un Product Backlog
- Lo Scrum Team trasforma una parte del backlog in un Increment delle attvità durante uno Sprint
- Lo Scrum Team e gli stakeholders ispezionano i risultati ed adattano il lavoro per lo Sprint successivo
- Si ricomincia dall'inizio

Il framework Scrum è volutamente incompleto e si limita a definire le linee di attività necessarie per definire il ciclo di sviluppo. E' fondato sull'intelligenza collettiva delle persone coinvolte in ogni attività. Piuttosto che fornire alle persone istruzioni dettagliate, le regole di Scrum guidano le loro relazioni ed interazioni in modo da creare nuovi processi, tecniche e metodi vari che portano alla definizione di un framework dinamico. Nel corso del tempo alcune pratiche esistenti potrebbero divenire non più necessarie. Rende visibile l'efficacia relativa dell'attuale gestione, ambiente e tecniche di lavoro affinché eventuali miglioramenti possano essere apportati. Scrum si basa sull'empirismo ed il pensiero Lean. L'empirismo afferma che la conoscenza derivi dall'esperienza e dal prendere decisioni basate su ciò che si è osservato. Il pensiero Lean riduce gli sprechi e si focalizza su ciò che è essenziale. Scrum utilizza un approccio iterativo ed incrementale per ottimizzare la produttività e controllare il rischio. Coinvolge gruppi di persone che, collettivamente, dispongono di tutte le competenze ed esperienze per eseguire il lavoro e condividere o acquisire tali competenze al bisogno.

2.3.1 Gli Scrum Team in Siav

L'unità fondamentale di Scrum è un piccolo gruppo di persone, uno Scrum Team. Gli Scrum Team in Siav per il momento sono 8, composte da 7 fino ad un massimo di 9 membri, dove ogni Team è composta da uno Scrum Master, un Product Owner e dai Developer. All'interno dei team non ci sono sottogruppi o gerarchie. Ogni Team è una unità coesa di professionisti concentrati su un unico obiettivo alla volta, il Product Goal. I Team sono cross-funzionali, ovvero ogni componente del Team ha tutte le competenze necessarie a creare valore ad ogni Sprint. Ogni Team inoltre è autogestito,

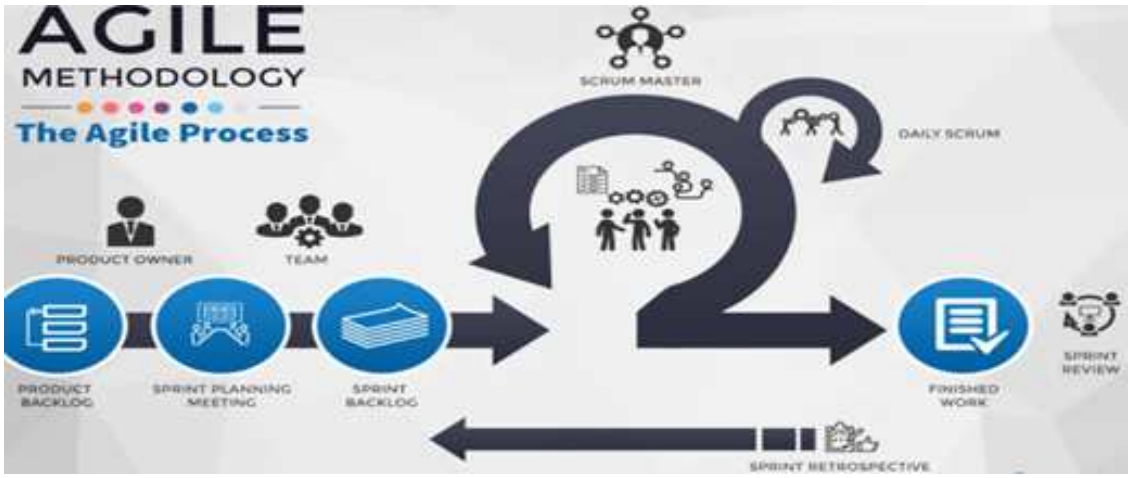

Figura 2.2. Scrum.

cioè si decide ad ogni riunione chi fa cosa, quando e come. I Team in azienda sono abbastanza piccoli da essere agili e grande abbastanza per portare a termine un lavoro significativo all'interno dello Sprint. Si è visto che, in genere, team piccoli comunicano meglio e sono più produttivi. Molte volte quando si vede che il Team diventa molto grande i membri del team decidono di riorganizzarsi in più Scrum Team coesi, ognuno dei quali concentrato sullo stesso prodotto. Per questo motivo i membri del team condividono lo stesso **Product Goal, Product Backlog e Product Owner.** Ogni Team è responsabile di tutte le attività correlate al prodotto come la collaborazione con gli stakeholder, la verifica, la manutenzione, il funzionamento, la sperimentazione, la ricerca, lo sviluppo e qualsiasi altra cosa possa essere richiesta. Sono strutturati e resi autonomi dall'organizzazione nel gestire il loro lavoro. Lavorare in Sprint a un ritmo sostenibile permette di migliorare la concentrazione e la coerenza dei Team. Ogni Team è responsabile nel creare un Increment utile e di valore ad ogni Sprint. Scrum definisce tre specifiche responsabilità all'interno dello Scrum Team: **i Developer, il Product Owner e lo Scrum Master.**

I Developer

I Developer sono le persone all'interno dei Team che hanno il compito di creare qualsiasi aspetto di un Increment usabile ad ogni Sprint. Le caratteristiche specifiche necessarie ai Developer sono spesso trasversali e varieranno

in base al dominio del lavoro. Comunque, i Developer, all'interno di ogni Team sono sempre responsabili di:

- Creare una pianificazione per lo Sprint, lo Sprint Backlog
- Instillare la qualità aderendo ad una Definition of Done
- Adattare quotidianamente la pianificazione allo Sprint Goal
- Ritenersi reciprocamente responsabili come professionisti

Certamente il Product Owner non è l'unico responsabile di tutto. In azienda ogni componente di ogni singolo team è responsabile di essere il più produttivo possibile, di migliorare le sue pratiche, di fare le domande giuste e di aiutare il Product Owner nel raggiungere gli obiettivi prefissati.

Lo Scrum Master

Lo Scrum Master è responsabile di promuovere Scrum così com'è definito nella Guida Scrum. Per far questo egli aiuta tutti a comprendere la teoria e le pratiche Scrum, sia all'interno dei Team che all'interno dell'organizzazione. Lo Scrum Master è responsabile dell'efficacia del Team. Questo viene fatto permettendo al Team di migliorare le proprie pratiche, definite nel framework Scrum. Gli Scrum Master sono dei veri leader al servizio dei Team e dell'organizzazione in generale. Lo Scrum Master offre un servizio ai Team in diversi modi, fra i quali:

- Allenare i membri del team all'autogestione e alla cross-funzionalità.
- Aiutare lo Scrum Team a concentrarsi nel creare Increment di grande valore che incontrino la Definition of Done.
- Eliminare gli impedimenti all'avanzamento del Team.
- Assicurare che tutti gli eventi Scrum siano svolti in maniera positiva, produttiva e che siano mantenuti entro i limiti temporali (timebox).

Lo Scrum Master fornisce un servizio al Product Owner in diversi modi, fra i quali:

- Aiutarlo nel trovare tecniche per la definizione efficace del Product Goal e per la gestione del Product Backlog.

- Aiutare il Team a comprendere il bisogno di elementi del Product Backlog chiari e concisi.
- Aiutare a stabilire una pianificazione empirica del prodotto per un contesto complesso.
- Facilitare la collaborazione con gli stakeholder se richiesto o necessario. Lo Scrum Master fornisce un servizio all'organizzazione in diversi modi, fra i quali:
- Guidare, formare ed assistere l'organizzazione nell'adozione di Scrum.
- Pianificare e consigliare l'implementazione di Scrum all'interno dell'organizzazione.
- Aiutare i dipendenti e gli stakeholder a comprendere ed attuare un approccio empirico per il lavoro complesso.
- Rimuovere le barriere fra gli stakeholder e gli Scrum Team.

2.3.2 Il ciclo delle attività di sviluppo e la mia esperienza

Durante l'attività di ogni giorno, cerco di assimilare nella maniera migliore possibile i principi dell'Agile Programming, tuttavia, non avendo mai avuto esperienze lavorative in aziende dove venivano applicati le metodologie Agile Scrum, e quindi nemmeno di sviluppo agile, non si può certo dire che tali principi siano facili da apprendere all'inizio. La strada verso la completa assimilazione di questa metodologia di sviluppo è lunga e tortuosa. Personalmente ho il modo di provare in prima persona alcuni dei dettami agile, essendo che in questo momento stiamo vivendo un momento particolare come la pandemia di covid-19, ogni riunione viene fatta in via telematica, essendo che la maggioranza dei dipendenti Siav lavora da casa, ossia:

- **Online daily meeting**: ogni giorno, prima di iniziare abbiamo la riunione con il gruppo di sviluppo, essendo che ogni tester segue un gruppo diverso di sviluppo, per fare il punto della situazione, programmare il lavoro quotidiano e discutere, se necessario, di problemi verificatisi nei giorni precedenti. Successivamente abbiamo un'altra riunione con il gruppo Quality Assurance per discutere sui problemi che abbiamo incontrato, e raccontare anche su quello che abbiamo fatto, cosi ogni giorno ripotiamo le attività del giorno precedente.

- **Retrospective meeting**: ogni due settimane c'è la riunione della squadra Quality Assurance dove ognuno di noi evidenzia idee per un miglioramento continuo dei processi. Evidenziamo ciò che è andato male o bene, le cose da migliorare e le nuove idee, proposte. Il team leader cerca di risolvere i problemi della squadra. Questa fase secondo me è molto utile nella squadra perché ti fa sentire parte della squadra e dell'azienda, che la tua opinione viene ascoltata ed ha un valore.
- **Dialogo continuo con il cliente**: personalmente non sono in contatto col cliente essendo una figura junior in fase di apprendimento, il nostro team leader dialoga frequentemente con il cliente e adatta le attività a seconda delle problematiche riscontrate nell'applicazione, ci da le indicazioni sulle funzionalità da testare.
- **Software over documentation**: Ogni test che facciamo viene documentata secondo le normative ISO 90003 di ingegneria del software. Viene creato un test case dettagliato specifico per ogni test in cui vengono tracciati tutti i vari passi in accordo con gli analisti che hanno progettato la funzionalità. Ciò permette di ripetere il test allo stesso modo anche in futuro o di poterlo automatizzare in un secondo momento.

Figura 2.3. Ciclo continuo di tutte le attività di sviluppo.

Capitolo 3

Software testing

Come qualunque altro prodotto esistente sul mercato, anche un programma software non è mai perfetto: una volta sviluppato, prima di essere distribuito, commercializzato o messo in produzione all'interno di un'organizzazione, richiede una meticolosa fase di collaudo, serve il software testing. Questa affermazione è ancora più attuale in un momento, come quello che stiamo attraversando, dove ogni azienda sta diventando una impresa digitale, dove la parte di software è preponderante e i sistemi software based sono sempre più complessi. La fase di collaudo, nei modelli di sviluppo tradizionali, viene pianificata al termine dello sviluppo del codice, e può servire, ad esempio, a verificare la corrispondenza delle funzionalità del software rispetto ai requisiti richiesti dall'utente, oppure se ci sono difetti in grado di pregiudicare la correttezza del funzionamento, o anche a controllare se il livello di usabilità del programma è basso, al punto da compromettere la produttività dell'utilizzatore a cui è indirizzato. Il testing del software è quindi imprescindibile per garantirne la qualità ed è indispensabile per garantire all'utente una user experience soddisfacente. Quest'ultimo è un fattore sempre più importante basti pensare che, secondo un'analisi Stardust, il 72% degli utenti che abbia una brutta esperienza di un servizio software o di una app la abbandona.

3.1 Processi di test

Sia che venga eseguito manualmente o attraverso metodi automatici, il processo di testing del software costituisce un tassello fondamentale nel ciclo di sviluppo del codice. Le attività di software testing sono indirizzate ad assicurare la qualità dei prodotti software e, in particolare nei contesti di business, l'analisi della qualità funzionale del codice, come detto, permette di verificare

quanto il software è in grado di soddisfare gli specifici requisiti per cui è stato progettato, quindi se le funzioni richieste non presentano difetti, ed anche, ad esempio, se sono conformi alle normative di un determinato settore. In altri termini si effettua una **Verification** (tutte le funzionalità oggetto dello sviluppo sono state completate) e una **Validation** (tutte le funzionalità non presentano anomalie).

Le fasi di verifica e validazione possono avvenire in momenti diversi a seconda del modello SDLC (ciclo di vita dello sviluppo software) utilizzato. Tutti i tipi di SDLC hanno le loro caratteristiche e in base ai requisiti del progetto, alle esigenze, alla fattibilità, alla durata, possiamo scegliere il particolare modello da utilizzare per lo sviluppo del software. *I principali modelli SDLC sono il waterfall e l'agile.*

Test a waterfall vs test agile

Waterfall: la fase di test si verifica solo dopo il completamento della fase di sviluppo.
Agile: i test vengono generalmente eseguiti in parallelo con lo sviluppo per garantire la qualità continua.

Pilastri dello Scrum

Scrum è una particolare declinazione di Agile. Si basa su tre pilastri:

- **Trasparenza**: significa che tutti coloro che partecipano ad un progetto sanno quale è lo scopo (trasparenza verticale) e sanno che cosa fanno gli altri (trasparenza orizzontale). Trasparenza implica anche che il lavoro e le misure delle loro performance siano visibili a tutti a qualsiasi livello organizzativo.
- **Ispezione**: basato sul concetto di controllo empirico di processo significa che ogni iterazione ed incremento vengano verificati in base alle metriche di misurazione decise per apportare modifiche alle iterazioni successive rendendo estremamente adattabile l'andamento del processo. Naturalmente occorre non superare la soglia di tolleranza del processo all'ispezione per non vanificarne la validità.
- **Adattamento**: è la conseguenza dell'ispezione e significa che al posto di seguire un piano preordinato i team ripianificano in base ai risultati dell'ispezione per apportare il maggior valore al cliente finale orientato al miglioramento continuo delle proprie performance.

Figura 3.1. Waterfall.

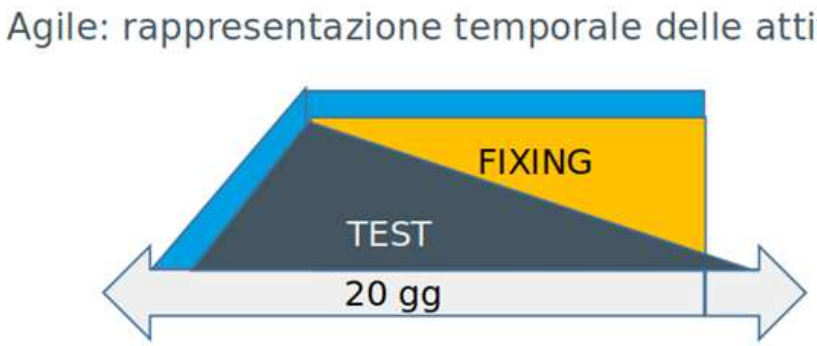

Figura 3.2. Agile.

3.2 Shift-left testing e shift-right testing

3.2.1 Shift-left testing

Come abbiamo visto nella rappresentazione temporale delle attività del modello waterfall, il testing si aveva solo al termine dello sviluppo applicativo. Nel corso degli anni ci si è resi conto che, nel caso di SIAV in cui si ha **sviluppo di applicazioni complesse con integrazioni hardware specifiche**, questo modello risulta inefficiente: non permette la coesistenza dell'improrogabilità delle date di consegna del software con un alto livello qualitativo. Accorgersi di eventuali gravi anomalie al termine dello sviluppo comporta

inevitabilmente un ritardo sulle consegne, oppure notevoli costi per la gestione dei bug in produzione.

Per questo motivo nel corso degli anni sono nati nuovi approcci al testing denominati "shift left": Il test con *spostamento a sinistra* nella scala temporale delle attività di controllo è quello più diffuso nell'ambito dello *sviluppo di sistemi complessi.*

Il modello di sviluppo agile incrementale, dove l'applicazione viene suddivisa in più parti testabili singolarmente rende possibile garantire che ogni segmento del programma sia funzionante e testato prima del completamento del prodotto e complessivamente rende possibile lo spostamento temporale a sinistra della scala temporale visibile in figura 3.2.
Più in dettaglio, il testing in modalita' Shift left è progettato per rilevare i problemi nelle prime fasi del ciclo di sviluppo, lasciando tempo a disposizione per le correzioni opportune, che avvengono più rapidamente perché lo sviluppo è ancora in corso.
La diminuzione dei tempi e degli eventuali bug in produzione porta ad una maggiore affidabilità del prodotto e ad una fidelizzazione dell'utente.

Il Team di controllo qualità collabora tempestivamente con tutti gli stakeholder del ciclo di sviluppo al fine di:
capire chiaramente i requisiti e progettare i casi di test per aiutare il software 'Fail Fast' e consentire di correggere tutti i guasti al più presto.

L'approccio Shift Left non è altro che il coinvolgimento dei tester molto prima nel ciclo di vita dello sviluppo del software, il che a sua volta consentirebbe loro di comprendere i requisiti, la progettazione del software, l'architettura, la codifica e le sue funzionalità, porre domande difficili a clienti, analisti aziendali e sviluppatori , chiedere chiarimenti e fornire feedback ove possibile per supportare lo sviluppo del prodotto.

Questo coinvolgimento e comprensione porteranno i tester ad acquisire una conoscenza completa del prodotto, a riflettere su vari scenari, a progettare scenari in tempo reale basati sul comportamento del software che aiuterebbero i team di sviluppo ad identificare i difetti anche prima che la codifica sia completata.

In poche parole, il processo di test Shift Left è:

- Trovare i difetti in anticipo riducendo così il costo del progetto.
- Testare continuamente ancora e ancora per ridurre i difetti finali.
- Automatizzare le attività di controllo e migliorare il time to market.
- Concentrarsi sulle esigenze del cliente per definire casi di test reali e migliorare quindi l'esperienza dell'utente.

Figura 3.3. Shift Left.

Il concetto ha portato una grande trasformazione per l'intero ruolo del tester che ultimamente prende il nome di Quality Enginner.

Oggi, il nuovo termine "QE" (ingegnere della qualità) è ciò che ritengo rappresenti al miglio il tester di software. Questo perché vedo il ruolo del QE come l'evoluzione di quello di un tester in un ruolo più tecnico, essendo coinvolto in molte aree del sviluppo. Allo stesso modo come i DevOps, l'ingegnere della qualità aiuta nel processo shift-left. Un ingegnere della qualità è qualcuno che applica l'ingegneria a diverse parti del processo di sviluppo a vantaggio della qualità. Tra le altre cose, un QE ha conoscenza delle operazioni (infrastruttura, server, piattaforme) che aiutano a eseguire test di sicurezza, test delle prestazioni, controlli di integrazione in uno schema CI/CD, ecc. Un ingegnere di qualità bravo potrebbe anche essere giustamente chiamato tester full stack.

Fino ad allora, l'unico obiettivo del test era solo il 'rilevamento dei difetti' e ora l'obiettivo del 'Spostamento a sinistra' dalla prospettiva del test è un viaggio di 'Rilevamento precoce dei difetti'.
Pertanto, *Shift Left* è un grande passo avanti nell'industria del software nella metodologia di sviluppo software verso la velocità di commercializzazione, il miglioramento della qualità del software e la riduzione del 'Time to Market'.

Introducendo il tester di oggi - Quality Engineer	
Tipologia	Descrizione
Definire un piano per la qualità + strategia	Come verra testato il software, dati gli obiettivi, rischi e contesto
Revisiona il codice	Comprendere il codice e suggerire miglioramenti, utilizzare gli strumenti migliori ed essere in grado di elaborare un piano d'azione basato su un'analisi del codice e un rapporto tecnico.
Crea rapporti sugli incidenti efficaci	Non trovare solo bug, ma risolverli. Questo significa essere un maestro della comunicazione
Eseguire test funzionali, automatizzati, di prestazioni e di sicurezza	Si, un bravo QE può farli tutti, anche se possono essere tante volte molto complicati.
Conoscere + gestire il flusso git	Il codice negli test dovrebbe essere trattato come parte del codice di sistema, utilizzando gli stessi strumenti di uno sviluppatore e la stessa metodologia di gestione delle versioni
Utilizzare ambienti CI	Tutti i controlli automatizzati devono essere conservati in un ambiente di test ci/cd come jenkins o qualcosa di simile per eseguirli frequentemente
Pensa con la mentalità di un imprenditore	Ultimo ma non meno importante, ciò che rende eccezionale un QE è la capacità di comprendere la visione globale orientata al business.

Tabella 3.1. QE

3.2.2 Shift-right testing

Nello sviluppo di applicazioni complesse con integrazioni hardware specifiche

Il testing durante lo sviluppo applicativo non è sufficiente ad anticipare tutte le possibili anomalie perché alcune di queste possono dipendere dall'ambiente di produzione specifico del cliente.

L'approccio Shift-right è un metodo per **testare continuamente il software in un ambiente di pre-produzione** e si riferisce alla definizione di

alcune attività di test a destra della sequenza temporale. Conosciuto anche come “test in produzione“, questo approccio aiuta gli sviluppatori a scoprire scenari inaspettati che potrebbero non essere stati rilevati nell’ambiente di sviluppo.

L’obiettivo del test shift-right è garantire il comportamento, le prestazioni e la disponibilità corretti duran-te l’uso di produzione di un’applicazione. Come funziona? *Questo tipo di approccio mira a incorporare prima possibile i test*, con un focus sulla prevenzione dei problemi piuttosto che sul rilevamento. I team di sviluppo software conducono esperimenti controllati verso la fine del ciclo di vita dello sviluppo software (SDLC) per esaminare funzionalità, prestazioni, tolleranza ai guasti e esperienza utente (UX). Il test Shift-right è importante per garantire che un’applicazione funzioni in tutte le circostanze e in tutti gli ambienti, come le diverse opzioni cloud. Inoltre, il test durante la pre-produzione è fondamentale per garantire la qualità del software perché consente un ciclo di feedback continuo da parte degli utenti e l’analisi di casi d’uso che non possono essere anticipati, come arresti anomali, guasti e prestazioni lente. Questo test è effettuato negli ambienti del cliente e con copie di dati reali. Da notare che questo tipo di attività non è prevista dalle normative ISO 90003 e per rispettare le norme sulla sicurezza dei dati, non può essere fatta in ambienti di sviluppo o di laboratorio esterni.

3.3 Shift-left e Shift-right testing nel team QA in Siav

Per i tester spingersi in una direzione o nell’altra può essere destabilizzante ma applicare tecniche di shift-right e shift-left è vantaggioso per garantire la qualità del software. Questa duplice modalità, infatti, funziona sia per lo sviluppo di software a ciclo infinito che per una programmazione che prevede il rilascio di piccoli gruppi di funzionalità e può essere gestita in un ciclo di sviluppo agile-scrum adattativo. Uno spostamento a sinistra e uno spostamento a destra ben eseguiti riducono i difetti del prodotto software, velocizzando il ciclo di sviluppo rispetto al relegare il test a una fase dedicata del processo. I test Shift-Left e Shift-Right sono spesso strategie complementari, piuttosto che concorrenti. Tuttavia, ogni approccio compor-ta rischi.

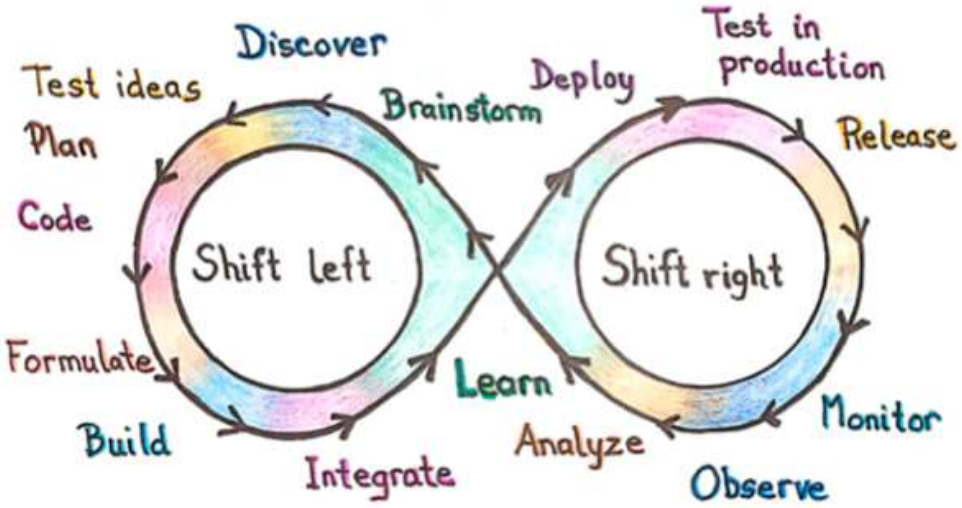

Figura 3.4. Shift-left e Shift-right testing.

Capitolo 4

Tipologie di test

Il processo di test si differenzia in due principali categorie:

- test di unità
- test di integrazione, che si dividono in sotto tipologie

Lo scopo dei test di unità è testare singole unità del programma come funzioni o oggetti. I test dei componenti da parte degli sviluppatori si basa sulla comprensione di come i componenti dovrebbero funzionare utilizzando solamente dati di prova.

I test di integrazione verificano che le varie unità comunichino correttamente e che il sistema complessivo soddisfi i requisiti funzionali e non si comporti in modo inaspettato. Tali test si basano sull'interazione dei vari componenti tra loro ed ogni componente fornisce dati reali in ingresso al componente successivo.

Il processo di test del software ha i seguenti obiettivi principali:

- Dimostrare che il software soddisfa i suoi requisiti funzionali. Ciò significa che dovrebbe essere fatto almeno un test a singolo utente per ogni requisito.
- Scoprire difetti del software. L'obiettivo è l'esplorazione ed eliminazione di tutti i tipi di comportamenti indesiderati del sistema, come ad esempio arresti anomali del sistema, interazioni indesiderate con altri sistemi, calcoli errati e corruzione dei dati.

Il primo obiettivo è chiamato test di convalida, in questo caso il sistema è testato da un insieme di casi di test che riflettono l'uso previsto del codice applicativo. Questo obbiettivo può essere raggiunto mediante i test di unità e test funzionali in ambiente integrato.

Il secondo obiettivo viene raggiunto con:

- test funzionali, che provano percorsi di test non documentati nelle funionalità. I casi di test sono progettati per esporre dei difetti. Si verifca il comportamento dell'applicazione con dati in ingresso non previsti, ad esempio caratteri speciali in su alcuni parametri.
- test di performance, che provano l'applicativo con molti utenti in concorrenza. Questi test possono portare a perdita di dati o arresti anomali del sistema.

Tipicamente queste due attività fanno parte dei test di integrazione

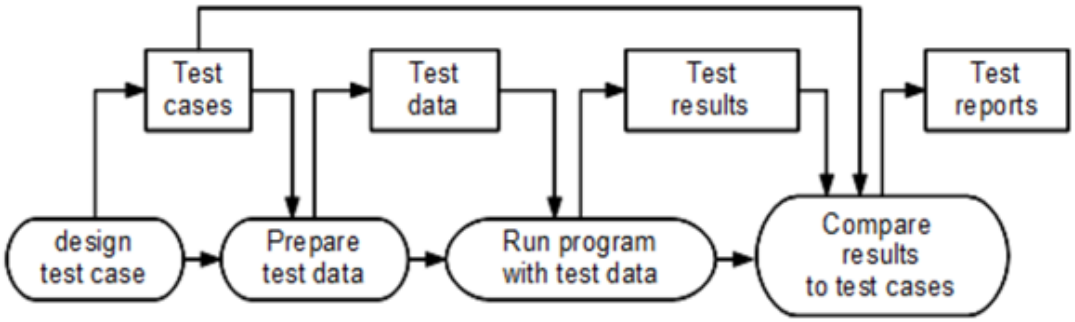

Figura 4.1. Modello generale del processo di test.

I casi di test dipendono da un certo numero di parametri di input (Es. username e password) e da alcuni dati iniziali che devono essere preesistenti nel sistema , l'output atteso dal sistema è una dichiarazione di ciò che è stato testato con gli eventuali errori riscontrati.

I test in ambiente integrato tipicamente si basano su un sottoinsieme di possibili dati di test. Questo perché la creazione degli script per l'esecuzione dei vari scenari di test e il popolamento automatico dei dati necessari è un operazione onerosa che comporta un lavoro periodico.

Nella parte destra della figura seguente viene riportato uno schema a piramide detto "piramide dei test" in cui compaiono le varie attività in ordine

di complessità / costi. Dal basso verso l'altro i costi periodici nelle varie iterazioni del ciclo di sviluppo aumentano.

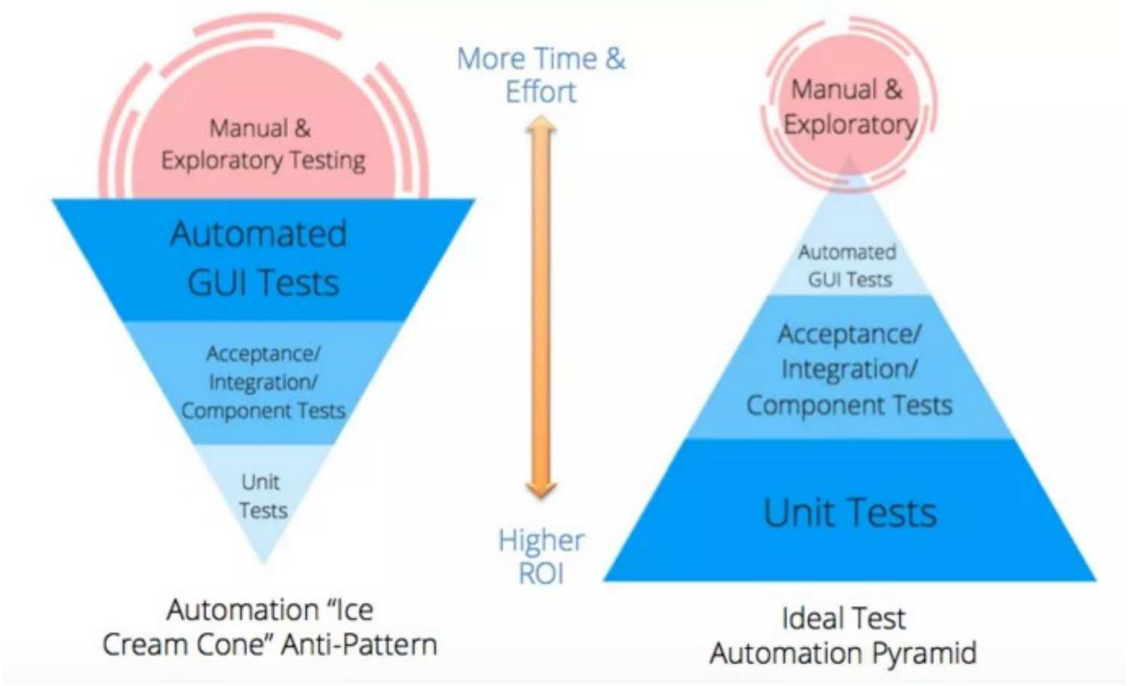

Figura 4.2. Piramide dei test.

Nella parte sinistra della figura precedente compare un “anti-pattern” cioè una proiezione che evidenzia un aumento dei costi nel caso in cui si sbagli la definizione delle attvità di test e il relativo controllo all’interno del ciclo di sviluppo. Questa errata gestione porta ad una non scalabilità delle attvità a fronte di sviluppi crescenti delle applicazioni software: viene chiamato effetto “cono gelato”.

Vediamo in dettaglio le attvità di test del ciclo di sviluppo in SIAV.

4.1 Test di unità

Il test di unità è il primo livello di test ed è spesso eseguito dagli stessi sviluppatori. È il processo per garantire che i singoli componenti di un software a livello di codice siano funzionali e funzionino come sono stati progettati. Gli sviluppatori in un ambiente guidato dai test generalmente scriveranno ed eseguiranno i test di unità prima che il software o la funzionalità vengano passati al team di test. I test di unità sono vere e proprie parti del codice applicativo che vengono istanziate all’atto della chiusura di un componente fornendo dei dati di input fittizi necessari per verificare il corretto funzionamento dei metodi applicativi dello specifico componente. Una volta sollecitati i metodi applicativi si verifica il risultato atteso. Tali test sono completamente automatici e permettono di individuare eventuali anomalie in una fase precoce del ciclo di sviluppo.

4.2 Test di integrazione

Dopo aver completato i test di unità si procede con il collaudo dell’intero sistema. Effettuare dei test sull’intero sistema significa affermare con una certa misura la qualità del prodotto finale. Tali test solitamente si basano sulle funzionalità espresse nelle specifiche e nei requisiti dell’applicazione e possono coinvolgere diverse configurazioni software e hardware. Questa tipologia di test si differenzia a seconda della modalità di sviluppo applicativo utilizzata: waterfall o agile.

- Nel caso di un processo di sviluppo a waterfall, il test di sistema riguarda i test dell’intero sistema e di tutte le nuove componenti nello stesso momento.

- Nel caso di sviluppo agile viene testato il sistema complessivo ma con solo una o alcune nuove componenti. Questo è possibile specialmente nel caso di applicazioni basate su architetture a microservizi, che sono componenti atomiche ed indipendeti del prodotto complessivo.

I test di integrazione comprendono le seguenti tipologie di test:

- **Test funzionali**, che hanno come obiettivo verificare che il sistema soddisfi i requisiti funzionali. Tali test sono fatti al completamento delle singole funzionalità ed anche dal cliente finale poco prima della messa in produzione. Quest'ultimo test è chiamato test di accettazione (UAT user acceptant test). Se il test è positivo il cliente conferma la messa in produzione del prodotto. In Siav questi test utilizzano il software Katalon per il loro sviluppo ed esecuzione

- **Test prestazionali**, che hanno come obiettivo la verifica dei tempi di risposta delle transazioni applicative e la verifica della stabilità del sistema sottoposto ad un carico concorrente di utenti applicativi. In Siav questi test utilizzano il software Jmeter per il loro sviluppo ed esecuzione

L'obiettivo principale di questa tipologia di test è verificare la corretta interazione tra le unità / moduli applicativi.
Nel ciclo di sviluppo Agile l test di integrazione vengono condotti contemporaneamente allo sviluppo, non appena un singolo modulo è pronto per essere integrato.
Nelle applicazioni software odierne le funzionalità applicative vengono suddivise in categorie. Ogni categoria afferisce ad uno specifico servizio applicativo (modulo), ed ogni servizio è di norma sviluppato da un team diverso. Viene stabilito un contratto di comunicazione tra moduli diversi in modo che lo sviluppo degli stessi possa procedere contemporaneamente. Alla fine degli sviluppi i moduli vengono installati in un ambiente detto di integrazione e se ne verifica la corretta comunicazione tramite i test. Quando questo approccio viene proiettato su N moduli distinti diventa importante verificare se la logica implementata da uno sviluppatore è conforme alle aspettative e restituisce il valore corretto in conformità con i contratti di comunicazione prescritti. I test di integrazione sono i primi test effettuati a valle dei test di unità. Poichè i test di unità vengono definiti dagli stessi sviluppatori può accadere che questi non siano del tutto efficaci. Spesso test sviluppati da persone diverse hanno una validità maggiore. Per questi motivi i test di integrazione rivestono una particolare importanza.

4.2.1 Test Funzionali

Possono essere progettati dei casi di test per verificare che le specifiche funzionali siano state implementate correttamente. Queste tipologie di test sono chiamate in letteratura test di conformità, o test di correttezza, o test funzionale. In ogni caso, possono essere testate anche numerose altre proprietà non funzionali, tra cui le prestazioni, l'affidabilità e l'usabilità.

Il test funzionale è un tipo di test comportamentale (cioè focalizzato sul comportamento osservabile del prodotto) che guarda al sistema come ad una collezione di funzioni. Chi effettua i test funzionali potrebbe focalizzarsi esclusivamente sulla presenza e l'affidabilità delle funzioni, tralasciando le caratteristiche non funzionali, come usabilità, scalabilità, mantenibilità, velocità, sicurezza, adattabilità a condizioni locali, esigenze di supporto, eccetera.

La differenza principale tra test funzionali e non funzionali è che il test funzionale è il tipo di test che garantisce il corretto funzionamento del prodotto software mentre il test non funzionale è il tipo di test che verifica gli aspetti non funzionali quali prestazioni, usabilità, affidabilità, ecc. del software.

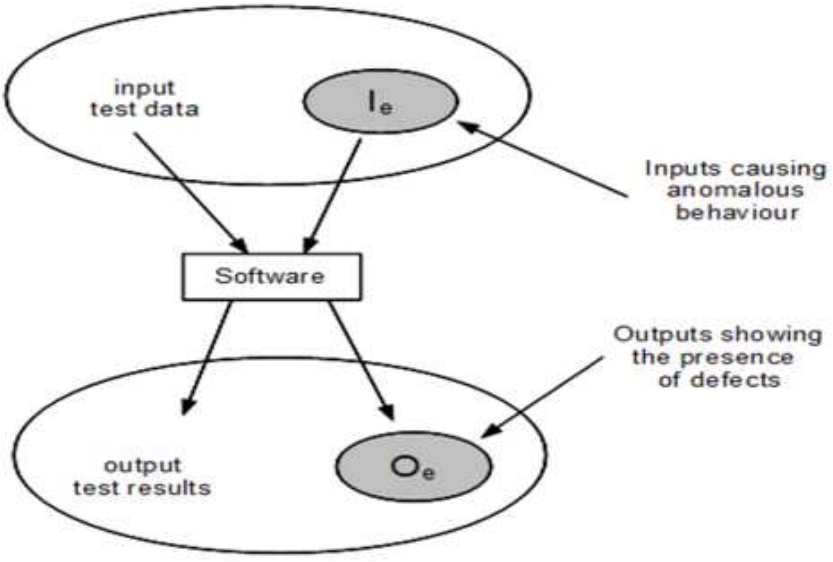

Figura 4.3. Input Output test funzionali.

Il test funzionale è il tipo di test che garantisce che il prodotto software funzioni in base ai requisiti. In questo test, testiamo tutte le funzionalità del prodotto. Il tester può fornire input appropriati e controllare le uscite.

Quindi può confrontare i risultati effettivi con i risultati attesi. Questo test coinvolge il controllo di interfacce utente, API, database, ecc.

Una versione del software è la sua versione finale che verrà distribuita ai clienti. L'obiettivo del test funzionale è dimostrare che il prodotto *software fornisce le funzionalità, le prestazioni e l'affidabilità specificate e che non fallisce durante il normale utilizzo.*

Il test funzionale riguarda solo la funzionalità e non l'implementazione del software. Un test funzionale è solitamente un processo di test della scatola nera in cui i test sono derivati dalle specifiche del sistema. La foto mostra il modello di test della scatola nera. Il processo di test presenta gli input al componente o al sistema ed esamina gli output corrispondenti. Durante i test il software viene spesso testato scegliendo i casi di test che si trovano in figura 4.3. Questi input hanno un'alta probabilità di generare guasti del sistema, ovvero output nel set Oe. Per convalidare che il sistema soddisfi i propri requisiti, l'approccio migliore da utilizzare è il test basato su scenari, in cui i casi di test vengono sviluppati in scenari, cioè possibili percorsi di utilizzo applicativo. Di solito, vengono testati prima gli scenari più probabili e successivamente gli scenari insoliti o eccezionali. Se casi d'uso vengono utilizzati per descrivere i requisiti di sistema, questi casi d'uso e i diagrammi di sequenza associati possono essere una base per il test del sistema.

4.2.2 Test di performance

L'obiettivo dei test di performance è garantire che il sistema possa elaborare il carico previsto di utenti applicativi concorrenti.

Esistono due principali attività di test prestazionale:

- **Load testing:** ovvero progettare dei test attorno al limite del sistema. Per limite si intende il valore massimo di utenti concorrenti entro il quale il sistema continua a funzionare normalmente per tutti gli utenti.

- **Stress testing:** ovvero progettare dei test che superino ii limite del sistema. In questo caso il sistema risponde in modo anomalo ma può continuare a funzionare oppure rompersi del tutto. Io scopo del test è capire come risponde il sistema in questo caso.

Genericamente per prestazioni di un applicativo si intende la sua capacità di fornire una funzionalità in tempi misurabili e brevi. Le prestazioni variano principalmente in funzione degli utenti contemporanei e della mole di dati che l'applicativo deve elaborare. Risulta naturale che le prestazioni, negli ambiti tecnologici moderni caratterizzati da un time to market brevissimo, siano particolarmente importanti nel testing in quanto agli occhi dell'utente finale rappresentano le migliori credenziali per giudicare l'affidabilità del proprio committente. Per questo quindi la preparazione dei test di performance deve essere sempre molto accurata.

La preparazione ed esecuzione dei test si basa essenzialmente sui seguenti punti di rilievo:

- Contestualizzare la performance ossia definire un ambito temporale minimo e massimo di applicazione, ad esempio da 1 a 8 ore per applicazioni utilizzate durante l'orario di lavoro tipico. Definire il tempo di risposta accettabile per la funzionalità in oggetto.

- Identificare le funzionalità applicative maggiormente utilizzate nella finestra temporale precedentemente definita e metterle sotto test.

- Nel caso in cui il test evidenzi una violazione del criterio di accettazione dare immediata evidenza dell'anomalia. Questo tipo di anomalie comporta attività di sviluppo molto lunghe.

- Dopo che la funzionalità ha superato i load test si procede con gli stress test, Occorre applicare alla funzionalità un numero progressivamente crescente di utenti e/o dati dati applicativi per vedere come reagisce l'applicazione.

Per verificare che il test sia effettivamente superato occorre controllare che il tempo di risposta effettivo (poniamo 20 secondi) con gli utenti massimi previsti sia inferiore al tempo di risposta stimato.

Come esempio possiamo considerare il caricamento di file gradi, e caricarli in modo continuo, con molti utenti contemporaneamente. Dopo la fase di load test descritta sopra, si procede allo stress test per avere delle analisi statistiche di quanto il sistema rege prima del punto di rottura.
I load test vengono anche utilizzati come test di non regressione prestazionale, e quindi effettuati automaticamente e periodicamente su funzionalità già note per verificare che non ci siano degenerazioni delle prestazioni nel tempo.

Figura 4.4. Load Test.

Il **Performance Testing** determina la corretta esecuzione di un sistema informatico misurandone i tempi di risposta. Rientra nella serie di attività svolte dal **Performance Engineering.** Il suo obiettivo è quello di fornire metriche sulla velocità dell'applicazione intesa come transazioni al secondo e tempo di risposta delle transazioni.. Il test delle prestazioni risponde così ad un'esigenza di **velocità** richiesta dagli utenti applicativi e delle aziende.

4.2.3 I test di resistenza

Il test di resistenza è un tipo particolare di load test (test di carico) in cui la durata del test è molto lunga. Per rendere facilmente l'idea di cosa rappresenti il carico pensiamo ad una cosa nota a tutti: il motore di un'auto. Cosa fanno i progettisti? Dapprima studiano il massimo rendimento per raggiungere potenza e velocità da 0 a 100 km, poi devono affrontare lo scoglio più importante: l'affidabilità. Devono garantire cioè che il motore renda nel tempo in tutte le condizioni e climi. Il parallelo con il software è intuitivo: la velocità è la performance, l'affidabilità è la resistenza (endurance). Il test di carico quindi deve garantire una perfetta tenuta di performance nel tempo soprattutto in condizioni estreme.

Come raggiungere questi obiettivi?
Coniugando due aspetti, uno tecnico e uno più concettuale:

Da una parte, tenendo in considerazione il valore del limite superiore degli utenti concorrenti, l'esecuzione del test aggiunge gradualmente dati alla base dati. Questo potrebbe comportare rallentamenti delle richieste di accesso alla base dati (pensiamo in questo caso alle centinaia di migliaia di accessi effettuati su un sito di acquisti on line in cui vengono inseriti dati relativi agli ordini effettuati, oppure ad un server specifico di una azienda di consegne internazionali), e potrebbe portare ad un punto critico ossia il punto in cui la funzionalità si blocca totalmente. Una volta appurato questo, vanno eventualmente sviluppate delle strategie di retention dei dati (archiviazione periodica) o di caricamento parziale degli stessi.

Dall'altra, va tenuto presente che per raggiungere il massimo carico possibile senza che la funzione sia instabile, occorre accettare una adeguata e limitata perdita di performance tendenziale, in modo che la funzione o sotto funzione in test sia sempre e comunque fruibile.

Capitolo 5

Testing automatizzato in SIAV

Il test del software automatizzato è la metodologia che aiuta a convalidare il funzionamento del software prima che venga spostato in produzione. In questo processo, i team di controllo qualità utilizzano strumenti di test automatizzati per eseguire gli script di test.

Con l'uso di strumenti di test del software automatizzati, i team QA possono testare rapidamente il software, preparare i rapporti sui difetti e confrontare i risultati del software con i risultati attesi. Inoltre, questo processo di test automatizzato offre numerosi vantaggi come:

- una riduzione dei tempi di consegna data dalla riduzione dei tempi dei test di non regressione
- garantire una maggiore qualità
- ridurre il numero di test manuali

In particolare:

Assicura l'accuratezza del test
Il test automatizzato, a differenza del test manuale, è per sua natura ripetibile e quindi vi è la garanzia che la pratica di test e la convalida dell'applicazione vengano eseguite con una buona precisione poiché gli errori vengono identificati in ogni fase.

Riduce il tempo dei test di non regressione
L'esecuzione di test di non regressione con metodi manuali richiede molto tempo e questo può portare ad errate esecuzioni e a bug non identificati. La ri-petizione degli stessi casi di test più e più volte potrebbe causare disinteresse tra i tester e potrebbe persino ridurre l'efficienza complessiva del test.

Riutilizzo del codice di scripting
Nei processi di scripting più moderno vengono utilizzate delle librerie, cioè delle raccolte di script contenenti l'invocazione parametrica di singole funzionalità applicative. Tali librerie sono utilizzabili su molti script diversi, nel caso cambiasse l'invocazione di una singola funzionalità applicativa, un aggiornamento della libreria basterebbe ad aggiornare tutti gli script che la utilizzano. Versioni diverse della libreria corrispondono a versioni diverse del prodotto sotto test.

Aumenta il ROI
Ogni impresa mira a ottenere migliori ritorni dai propri investimenti. Con lo sviluppo di un framework di test di automazione efficace, i ritorni sono enormi poiché i test vengono eseguiti più velocemente con le funzionalità integrate. Inoltre, i test di automazione forniscono risultati più rapidi e di qualità, migliorano il time to market e infine garantiscono un maggiore ritorno sull'investimento.

Gli strumenti utilizzati nell'esecuzione dei test automatizzati sono:

Jenkins: un orchestratore di processi locali o remoti
Jmeter: un generatore di carico in grado di eseguire test funzionali e di performance sulle API applicative
Katalon: un software in grado di simulare le interazioni di un singolo utente con il browser

Grazie a Jenkins riusciamo ad eseguire Jmetter e Katalon ogni sera: viene creato un ambiente applicativo, vengono creati i dati di test, e vengono eseguiti i test. Ogni test ha un report associato e al termine dell'attività l'ambiente viene eliminato. Cosi riusciamo ad avere una visione chiara dei problemi che i nuovi sviluppi hanno portato e si può intervenire in tempi brevi. Nella foto 5.1 si vede il flusso degli strumenti collegati tra loro.

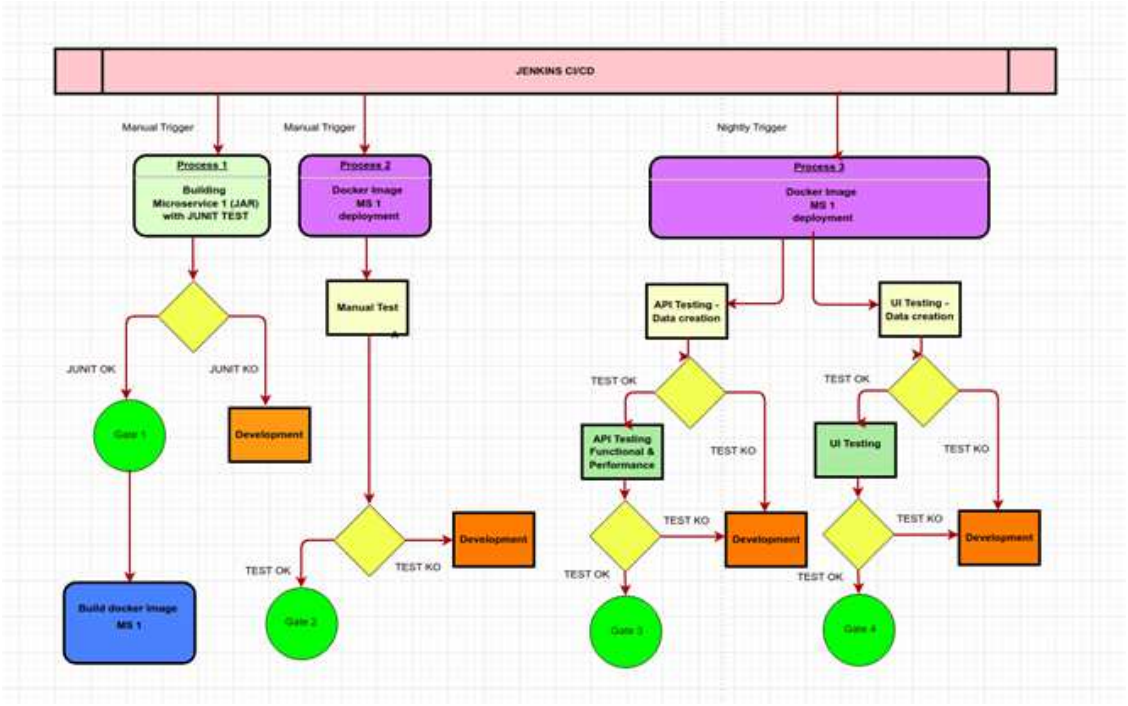

Figura 5.1. Framework di test

5.1 Jenkins

Jenkins è un server di integrazione continua open source scritto in Java per orchestrare una catena di azioni per ottenere il processo di integrazione continua in modo automatizzato. Jenkins supporta l'intero ciclo di vita dello sviluppo del software dalla creazione, test, documentazione del software, distribuzione e altre fasi del ciclo di vita dello sviluppo del software.

Per tutte le aziende che si occupano di sviluppo software, anche solo in minima parte, il mantra di questi ultimi tempi è DevOps: come gestire in maniera integrata le fasi di sviluppo, test e distribuzione del software, con l'obiettivo finale di arrivare a rilasci che siano allo stesso tempo frequenti e stabili. Il tema collegato è quello della Continuous Integration o della Continuous Delivery: idealmente i rilasci successivi di nuovo codice costituiscono un flusso continuo che si installa in produzione con la (relativa) certezza di non bloccare il buon funzionamento delle applicazioni che muovono il business d'impresa.

Ci sono diverse piattaforme software commerciali che aiutano a creare ambienti efficaci di DevOps e CI/CD, ma c'è anche una soluzione open source molto diffusa che, tra l'altro, muove proprio alcune delle soluzioni commerciali. Si tratta di Jenkins, un progetto che viene citato spesso nel mondo dello sviluppo e che vale la pena conoscere anche solo sommariamente.

Jenkins nasce oltre dieci anni fa in Sun Microsystems con un nome diverso (Hudson). Allora è in sintesi un modulo server che il suo creatore Kohsuke Kawaguch ha creato per automatizzare i processi di test e rilascio del codice Java. Qualche anno dopo da Hudson nasce Jenkins, che ne è sostanzialmente un fork e che attualmente ha una diffusione molto superiore a quella del predecessore.

Utilizzando Jenkins, le società di software possono accelerare il processo di sviluppo del software, poiché Jenkins può automatizzare la creazione e il test a un ritmo rapido.

È un'applicazione basata su server e richiede un server web come Apache Tomcat. Il motivo per cui il software Jenkins è diventato così popolare è il monitoraggio delle attività ripetute che si verificano durante lo sviluppo di un progetto.

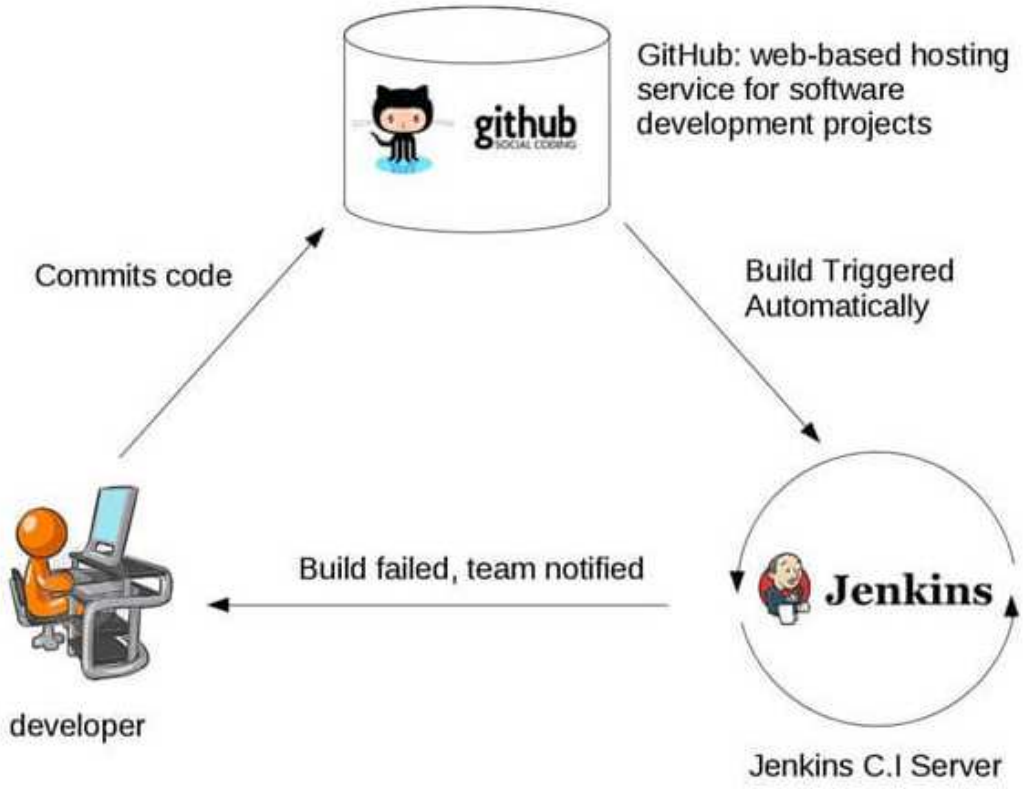

Figura 5.2. Strumenti collegati tra loro a partire dallo sviluppatore

5.2 Jmeter

Apache JMeter è un progetto open source destinato principalmente al test di carico e stress delle applicazioni web ma non solo. Col tempo, infatti, è stato esteso con molti connettori, plugin e script di terze parti che lo rendono un ottimo strumento per il test di database, server di posta, server Java Enterprise (JEE), LDAP, etc. Può essere utilizzato per simulare un carico consistente su un server, su un gruppo di server, su una rete o analizzare la performance complessiva per differenti tipi di carico.

Nello specifico, si può eseguire test di carico e di performance per varie tipologie di server e protocolli:

- Web – HTTP, HTTPS
- SOAP / REST
- FTP
- Database via JDBC
- LDAP
- Message-oriented middleware (MOM) via JMS
- Mail – SMTP(S), POP3(S) e IMAP(S)
- MongoDB (NoSQL)
- Script di shell nativi
- TCP

Inoltre JMeter è un framework completamente multithread e ciò consente di verificare la concorrenza tramite la generazione di vari thread e la simultaneità di differenti funzionalità attraverso gruppi di thread separati. E' dotato di un'interfaccia grafica che permette di creare e debuggare velocemente i test. JMeter conserva i propri piani di test in formato XML. Ciò significa che i piani di test possono essere generati utilizzando un semplice text editor.

Jmetter nel nostro team viene usato principalmente per i test di performance, ma anche per dei test funzionali.

5.3 Katalon

Katalon Studio è una potente soluzione di automazione con un set di funzionalità complete e integrate che vanno dalla registrazione delle fasi dei test alla generazione degli script all'esecuzione e alla creazione di report sui risultati dei test, per l'automazione dei test sulle app per Web, API, mobile e desktop. Questa soluzione è diffusamente utilizzata da una comunità globale di utenti ed è riconosciuta come uno dei migliori strumenti di automazione.

Le principali caratteristiche dello strumento:

- linguaggio di programmazione: Groovy (con supporto Java)
- stile di programmazione test: chiamare metodi di classi (parole chiave)
- supporta il cetriolo BDD per i test basati sul comportamento (stile When-And-Then)
- utilizza motori Selenium e Appium, due dei principali motori di esecuzione di test in commercio
- supporta SOAP e RESTful per testare API e servizi
- le capacità di test possono essere estese creando parole chiave personalizzate, plug-in dal Katalon Store, importando librerie di terze parti da file jar.
- visualizzazione dettagliata dei report in Katalon TestOps: un applicazione cloud di reportistica integrata con lo strumento.

Bibliografia e Sitografia

[1] Zsolt Dr. Ulbert, *Software development processes and software quality assurance*, University of Pannonia, February 2014, pg. 127

[2] Sito ufficiale Siav: https://www.siav.com/it/

[3] Why Shift-Left Testing? Pros and Cons: https://www.testim.io/blog/shift-left-testing/

[4] Shift Left: https://devopedia.org/shift-left

[5] Shifting left and right in our continuous world: https://lisacrispin.com/2020/11 left-right-in-our-continuous-world/

[6] What is a QE?: https://abstracta.us/blog/software-testing/infographic-what-is-a-qe/

[7] Mastering SCRUM?: https://www.invensislearning.com/blog/agile-scrum-methodology/