Metodi e Tecnologie per lo Sviluppo Software - Summary

Leonardo Baldo

Contents

1 Issue Tracking System

Issue Tracking System: computer software package that manages and maintains lists of issues, as needed by an organization.

- **Issue:** criticità, attività/evento da gestire
- **Tracking:** registrare, lasciare delle tracce

1.1 Utilizzo

- **Condividere** le informazioni
  - unica repository dove trovare le informazioni
  - sistema di notifica
  - dashboard
- Implementare un processo per misurarne la **qualità**
- Avere un'istantanea dello **stato del progetto**
  - attività da fare
  - in corso d'opera
  - completate
- Decidere quando e cosa **rilasciare**
- Assegnare e dare **priorità alle attività**
- Consultare il **tempo impiegato**
- Avere una chiare **assegnazione delle attività**
- **Memoria storica** di tutti i cambiamenti del progetto

1.2 Work Item

Work item: singola attività minima del progetto, gestita mediante un workflow e mantenuta all'interno di un'unica piattaforma e di un'unica repository.

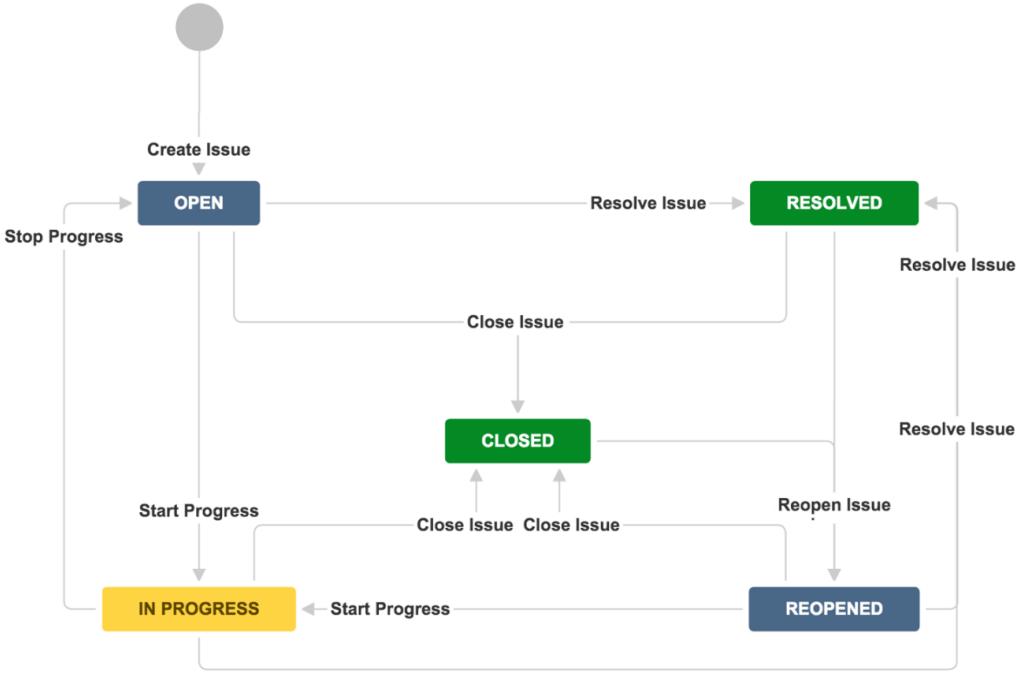
1.2.1 Caratteristiche

- **Progetto:** progetto a cui si riferisce
- **ID:** identificativo univoco
- **Descrizione:** descrizione dell'attività
- **Tipo:** categoria del work item
- **Stato:** stato all'interno del workflow in cui si trova il work item
- **Priorità:** importanza del work item in relazione con gli altri work item del progetto
- **Tag:** permettono di classificare i work item, anche di diversi tipi
- **Collegamenti:** permettono di collegare tra loro i work item
- **Assegnatario:** identifica chi è il responsabile per svolgere l'attività
- **Segnalante:** identifica chi ha segnalato l'attività
- **Data:** data di creazione, di ultimo aggiornamento, di risoluzione
- **Allegati:** file allegati

1.3 Workflow

Workflow: insieme di stati e transizioni che un Work Item attraversa durante il suo ciclo di vita.

- Permette di implementare il processo da seguire per completare l'attività
- Viene associato ad un progetto e può essere associato a uno o più tipi
- Permette di registrare tutte le transizioni e cambi si stato



1.4 Funzionalità

Gestione:

- Ricerca avanzata dei work item
- Salvataggio di ricerche
- Esportazione
- Reporting

Integrazione:

- Integrazione con il Source Code Management
- Integrazione con l'ambiente di sviluppo

Condivisione:

- Notifiche
- Bacheche o Board
- Dashboard
- Definizione di Road Map e Release Notes

1.4.1 Filtri

- Ricercare i work item in base ai campi
- Salvati per facilitare le ricerche più frequenti
- Risultati possono essere esportati
- Base per creare report, board e dashboard

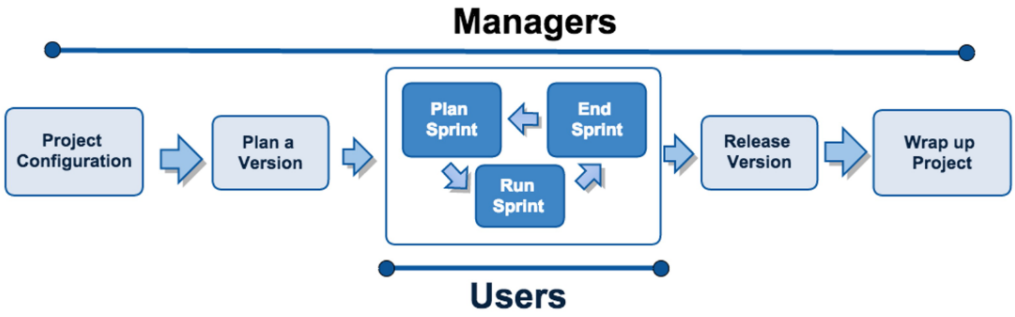
1.4.2 Board

- Visualizzare i work item di uno o più progetti, offrendo in modo flessibile e iterativo di visualizzazione, gestione e visualizzare dati di sintesi sulle attività in corso
- Configurare e visualizzare i work item ricercati con un filtro
- Interagire velocemente con i work item

1.4.3 Report

- Monitorare e avere una visione d'insieme del progetto

1.5 Configurazione



1.5.1 Obiettivi

- Identificare i processi richiesti per la gestione del progetto:
  - Procedure e best practices definiti dai framework di qualità presenti in azienda o richiesti dal cliente
  - Vincoli imposti dal cliente
  - Modalità di gestione del progetto del team
- Identificare e configurare gli strumenti che permettono di implementare i processi:
  - Identificazione e definizione dei tipi, campi custom, workflow e collegamenti che ci permettono di tracciare le informazioni richieste dal processo

1.5.2 Configurazine

Admin:

- Crea un nuovo progetto
- Definisce il processo da seguire:
  - tipi di work item, campi custom, workflow, collegamenti
  - seleziona il modello di stima
  - board e report per processo
- Aggiunge gli utenti e assegna ruoli/permessi

Capo progetto:

- Definisce le versioni [release]
- Definisce le componenti del progetto
- Definisce il lavoro da svolgere [backlog]:
  - priorità
  - assegnatario
  - stima
- Definisce la prima iterazione

1.5.3 Utilizzo

Team di Sviluppo:

- Riceve le notifiche dei work item assegnati
- Selezionano i work item in base alle priorità
- Avviano e completano la lavorazione:
  - avanzano gli stati del workflow
  - aggiornano la stima a finire
  - registrano il tempo impiegato
- Documentano lo stato dell'attività (commenti) e compilano e campi nel work item
- Completano tutte le attività presenti nell'iterazione
- Effettuano il rilascio

Capo progetto:

- Monitora l'avanzamento e il completamento delle attività (filtri, board. dashboard, report)
- Definisce le nuove versioni
- Definisce le nuove iterazioni
- Definisce, aggiorna e monitora le attività (priorità, verifica stima)
- Produce i report richiesti dal cliente

2 Version Control System

VCS: a component of software configuration management, is the management of changes to documents, computer programs, large web sites, and other collections of information.

2.1 Caratteristiche

- Sono sistemi software
- Registrano modifiche avvenute ad un insieme di file
- Condivisione di file e modifiche
- Funzionalità: merging, tracciamento modifiche

2.2 Tipologie di VCS

2.2.1 Local VCS

- Tool più vecchi
- Registrano solo storia cambiamenti
- Non gestiscono la condivisione
- Esempi: SCSS, IDE(Eclipse, IntelliJ)

2.2.2 Centralized VCS - CVCS

- Meno vecchi e molto diffusi
- Gestiscono sia la condivisione, che il tracciamento della storia
- Ogni sviluppatore è un client che ha nel suo spazio di lavoro solo una versione del codice
- Facili da apprendere
- Esempi: CVS, Subversion(SVN), Perforce, TFS

2.2.3 Distributed VCS - DVCS

- Version Database distribuito per duplicazione in ogni nodo
  - quando il nodo principale non è disponibile, è possibile continuare a lavorare e registrare i cambiamenti
  - migliore risoluzione dei conflitti
  - diversi tipi di flussi di lavoro
- Difficili da apprendere
- Esempi: Git, Mercurial

2.2.4 Cloud-Based DVCS

- VCS as a Service
- Version Database gestito in un servizio Cloud
- Esempi: GitHub, GitLab

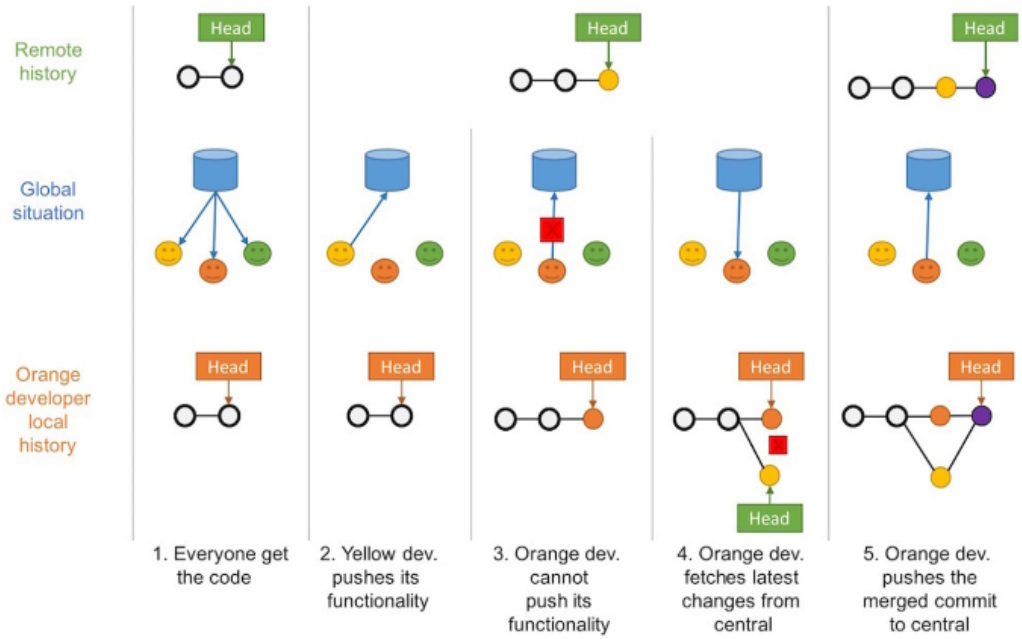
2.3 Nozioni

- **DIFF:** each set of changed lines
- **COMMIT:** set of DIFF
- **HEAD:** last commit
- **BRANCH:** pointer to a single commit
  - HEAD is the latest branch, know as **main** branch
  - to integrate a branch, you have to merge it
- **PULL REQUEST:** way of handling branch merges to main
  1. branch pushed to the central server
  2. ask to be merged on a central repo
  3. review change before merging
  4. pull request closed or merged to destination branch

2.4 Tipologie di Workflow

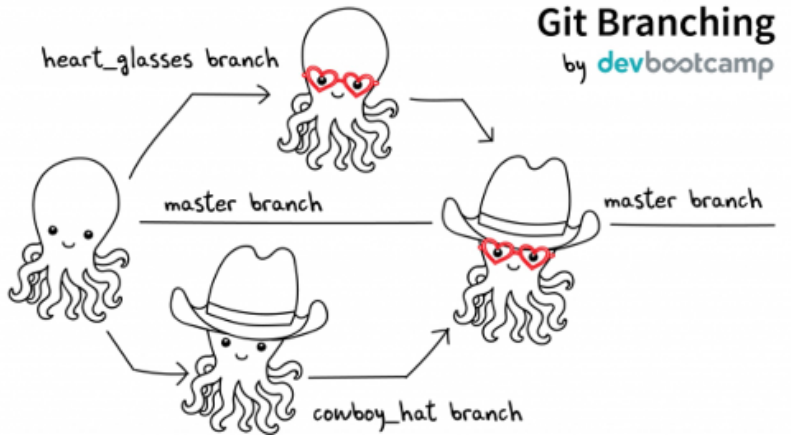
2.4.1 Centralized WF

- Utilizzo naturale di un CVCS come SVN o CVS
- Facile da capire e da usare
- Collaborazione bloccata quando il server centralizzato è fuori uso o la cronologia è interrotta



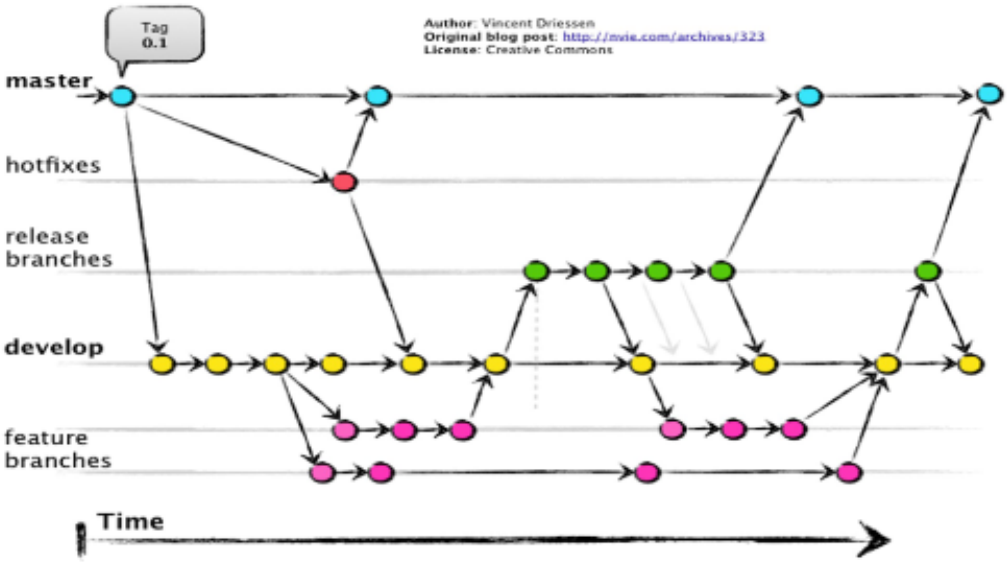
2.4.2 Feature Branch WF

- L'obiettivo è quello di utilizzare un solo ramo per caratteristica (DVCS)
- L'incapsulamento consente di lavorare senza distribuire la base di codice principale
- Collaborazione più facile
- Più facile da tracciare



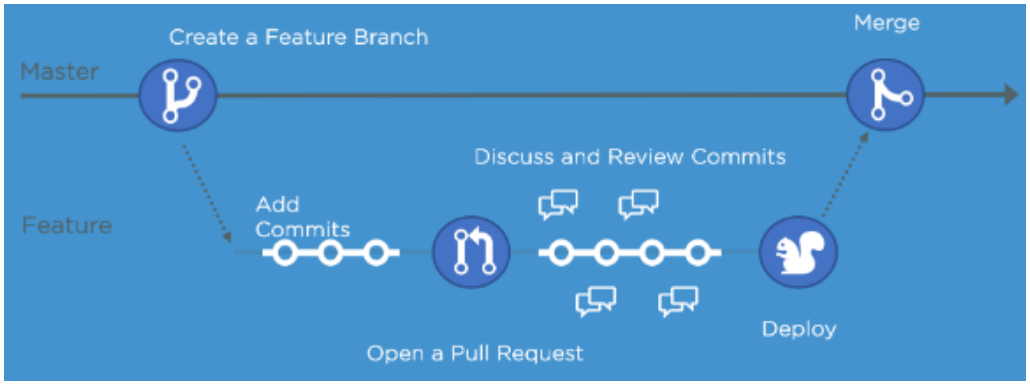
2.4.3 Gitflow Model WF

- main branch: codice rilasciato
- developer branch: snapshot per la prossima release
- feature branch: nuova funzionalità



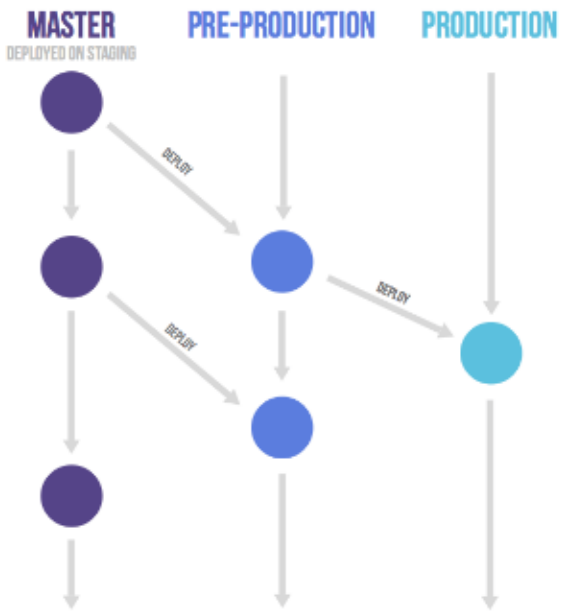
2.4.4 GitHub Flow

- Approccio più veloce di sviluppo
- Focalizzato sulle caratteristiche per unire i nuovi rami con il ramo master
- Flusso di lavoro perfetto per piccoli team e progetti



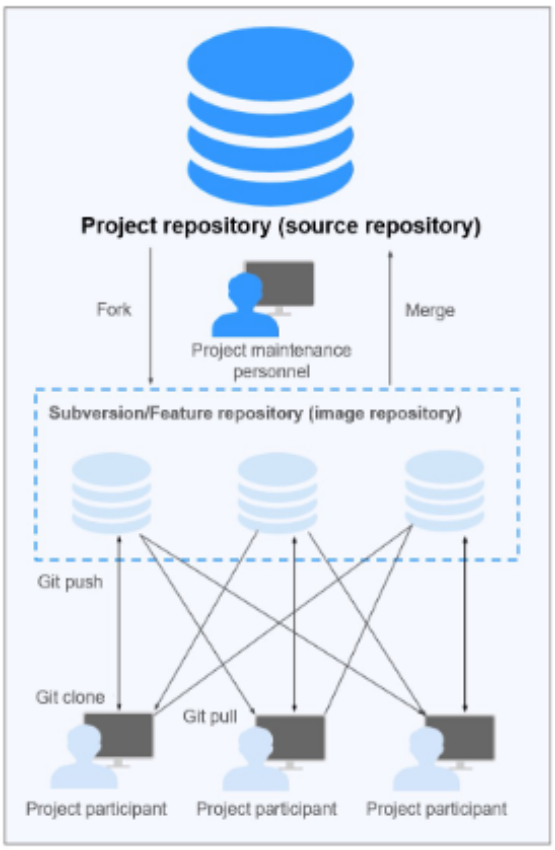
2.4.5 GitLab Flow

- Approccio di sviluppo più attento all'affidabilità
- Processo di test a più fasi



2.4.6 Forking WF

- Concetti di push forward del file system distribuito
- Ogni utente fa il fork del repo principale e può proporre richieste di pull tra i repo
- Gestione delle autorizzazioni migliorata
- Autonomia per un migliore processo di collaborazione
- Decentrato per nuovi modelli



2.5 CVCS vs DVCS

CVCS:

+ apprendimento più semplice
+ lock file
- meno recenti
- centralized workflow
- commit più lenti
- single point of failure

DVCS:

+ più recenti
+ distribuiti
+ migliori workflow
+ commit veloci
- no lock file
- apprendimento difficile

3 Framework Scrum

Scrum: processo agile che nasce per lo sviluppo di progetti complessi, che ci permette di concentrarci sulla consegna del maggior valore business nel più breve tempo.

3.1 Caratteristiche

- Leggero
- Facile da capire
- Difficile da padroneggiare

3.1.1 3 Pilastri

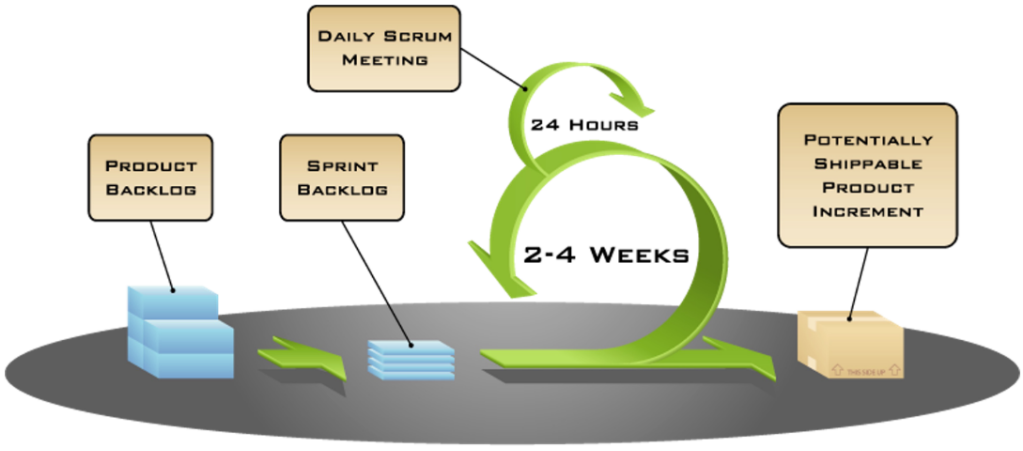
- **Trasparenza:** linguaggio comune per una conoscenza condivisa
- **Controllo:** ispezioni pianificate per prevenire variazioni non desiderate
- **Adattamento:** aggiustamenti per minimizzare ulteriori deviazioni tramite feedback continuo

3.1.2 Proprietà

- Gruppi che si **auto-organizzano**
- Il prodotto evolve attraverso **sprint** di durata fissa
- I requisiti sono trattati come elementi di una lista detta "product backlog"
- Non vengono prescritte particolari pratiche ingegneristiche
- Si basa sull'attività empirica cioè la conoscenza si basa sull'esperienza e le decisioni si basano su ciò che è conosciuto
- **Processo iterativo e incrementale** per ottimizzare il controllo dello sviluppo e il controllo del rischio

3.2 Sprint

- I progetti Scrum progrediscono attraverso una serie di sprint
- Durata tipica di 2-4 settimane: una durata costante favorisce un ritmo migliore
- Il prodotto è progettato, realizzato e testato durante lo sprint



3.3 Ruoli

3.3.1 Product Owner

- Definisce le caratteristiche del prodotto
- Rappresenta il desiderio del committente
- Decide date e contenuto del rilascio
- È responsabile della redditività del prodotto (ROI)
- Definisce le priorità delle caratteristiche del prodotto in base al valore che il mercato gli attribuisce
- Adegua le caratteristiche e la priorità ad ogni iterazione, secondo quanto necessario
- Responsabile che il Product Backlog sia chiaro e ordinato
- Accetta o rifiuta i risultati del lavoro

3.3.2 Scrum Master

- Rappresenta la conduzione del progetto
- Responsabile dell'adozione dei valori e delle pratiche Scrum
- Rimuove gli ostacoli
- Si assicura che il gruppo di lavoro sia pienamente operativo e produttivo
- Favorisce una stretta cooperazione tra tutti i ruoli e le funzioni
- Protegge il gruppo di lavoro da interferenze esterne
- Servant leader: aiuta Product Owner e Team di sviluppo condividendo la gestione e le decisioni con il team

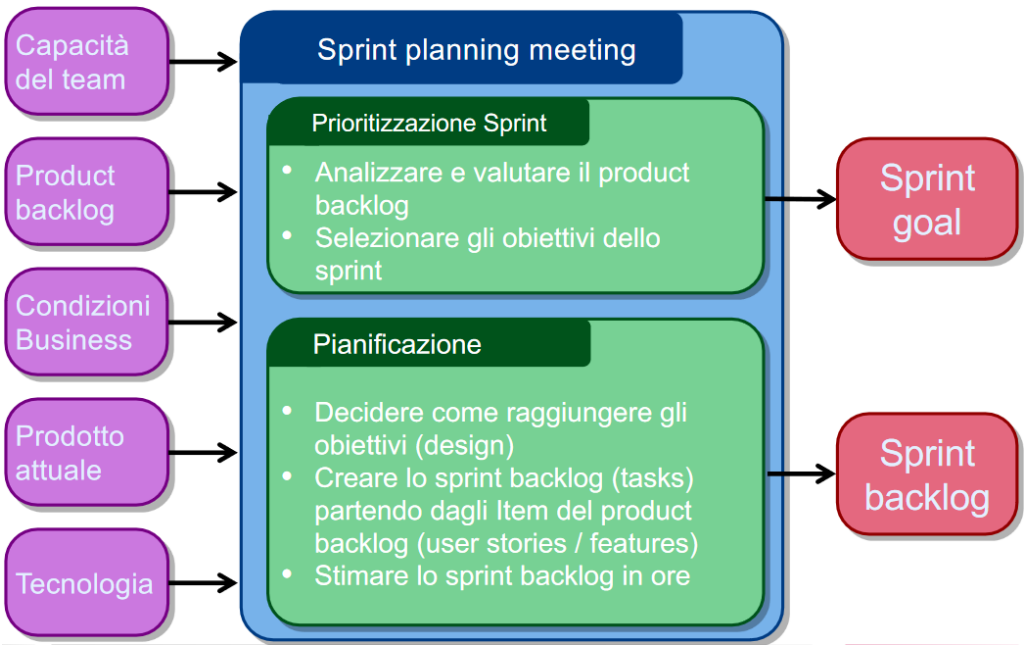
3.3.3 Development Team

- Tipicamente 5-9 persone
- Responsabili di realizzare l'incremento in conformità alla Definition of Done
- Competenze trasversali (cross functional): programmatori, tester, progettisti di user experience, ...
- Membri di progetto dovrebbero lavorare full-time
- Possono esserci eccezioni (e. amministratori di database)
- Il gruppo di lavoro si auto-organizza: idealmente senza titoli, ma in rari casi può essere una possibilità

3.4 Eventi

3.4.1 Sprint Planning

- È un evento "Time boxed" di 8h per Sprint di 1 mese
- Cosa può essere realizzato durante lo Sprint? Il Team seleziona dal product backlog gli item che può impegnarsi a completare
- Viene creato lo Sprint backlog collaborativamente da tutto il team
- Vengono identificate le Task, e ciascuno di questi viene stimato (1-16 ore)
- Come completare il backlog?
- Decomposizione delle User Story



3.4.2 Daily Scrum Meeting

- Incontro giornaliero di 15 minuti, fatto in piedi
- Non per la soluzione di problemi, ma per sincronizzarsi su quanto fatto e pianificare la giornata per il raggiungimento dello Sprint Goal
- Si aggiorna la scrumboard
- Aiuta ad evitare altre riunioni non necessarie
- In caso può partecipare anche il Product Owner
- Domande:
  - Cosa hai fatto ieri?
  - Cosa farai oggi?
  - C'è qualcosa che ti impedisce di farlo?

3.4.3 Sprint Review

- Time boxed: 4h per Sprint di 1 Mese
- Il gruppo di lavoro presenta ciò che ha realizzato durante lo sprint
- Viene validato e accettato quanto realizzato
- Tipicamente in forma di demo delle nuove caratteristiche o dell'architettura sottostante
- Informale:
  - Regola delle 2 ore per la preparazione
  - Niente slide
- Partecipa tutto il gruppo
- Tutti sono invitati (anche gli esterni)

3.4.4 Sprint Retrospective

- Si celebra dopo la Sprint Review e prima del prossimo Sprint Planning
- Time boxed: 3 ore per Sprint di 1 mese
- Si valuta ciò che sta funzionando e ciò che non sta funzionando
  - Come migliorare la qualità del prodotto?
  - La Definition of Done è appropriata?
  - Che miglioramenti possiamo apportare al prossimo Sprint?
- Partecipa tutto il gruppo di lavoro:
  - Scrum Master
  - Product Owner
  - Development Team

3.5 Artefatti

3.5.1 Product Backlog

- I requisiti, funzionalità, miglioramenti, fix da realizzare nei prossimi rilasci
- Una lista di tutti i "desiderata"
- Idealmente espressa in modo che ciascun elemento ha valore per gli utenti o I clienti del prodotto
- Priorità assegnate dal Product Owner mentre il Dev. Team stima ogni item
- Priorità rivalutate all'inizio di ogni sprint con il Development Team
- Raffinamento continuo, è una lista dinamica che evolve con il prodotto

User Stories: Item che compongono il Product Backlog e andranno scomposte in Task

3.5.2 Sprint Backlog

- Ogni componente del Development Team si sceglie qualcosa da fare
- La stima del lavoro rimanente è aggiornata ogni giorno
- Ogni membro del gruppo di lavoro può aggiungere, cancellare o modificare parti dello sprint backlog
- Il lavoro da svolgere durante lo sprint "emerge"
- Se il lavoro non è chiaro, definire un elemento dello sprint backlog con una stima temporale più ampia, e decomporlo successivamente
- Aggiornare il lavoro rimanente man mano che diventa più chiaro
- Massima visibilità della scrumboard

3.5.3 Definition of Done

- Il minimo set di attività per definire che un'attività è completata
- Può variare per gruppo di lavoro
- Deve essere bene chiaro per tutti i membri del gruppo di lavoro
- È utilizzato per verificare se un'attività è da ritenersi completata

3.5.4 Acceptance Criteria

- Permette di confermare se la storia è completa e funziona come voluto
- Frasi semplici condivise da Product Owner e Development Team
- Possono essere incluse con la User Story
- Rimuovono l'ambiguità dei requisiti

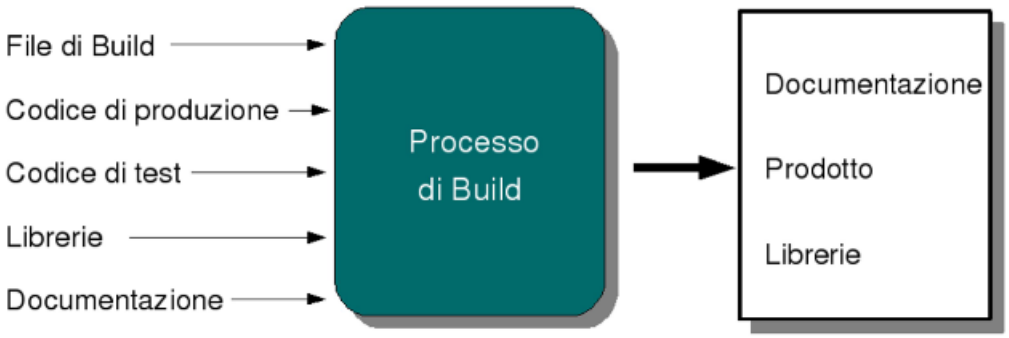
4 Build Automation

Build automation: processo di automazione della creazione di una software build e dei processi associati, tra cui la compilazione del codice sorgente del computer in codice binario, il confezionamento del codice binario e l'esecuzione di test automatici.

- **Build-automation utility:** whose primary purpose is to generate build artifacts through activities like compiling and linking source code.
  - scripting tools: `.sh`, `.bat`, `Makefile`, Gradle, ...
  - artifact oriented tools: Apache Maven, ...
- **Build-automation server:** strumenti generali web based che eseguono utilità di build-automation su base programmata o attivata.

4.1 Processo di Build

Processo di build: insieme di passi che trasformano gli script di build, il codice sorgente, i file di configurazione, la documentazione e i test in un prodotto software distribuibile.



4.1.1 Caratteristiche CRISP

- **Completo:** Indipendente da fonti non specificate nello script di build
- **Ripetibile:** Accede ai file contenuti nel sistema di gestione del codice sorgente. Una esecuzione ripetuta dà lo stesso risultato
- **Informativo:** Fornisce informazioni sullo stato del prodotto
- **Schedulabile:** Può essere programmato ad una certa ora e fatto eseguire automaticamente
- **Portabile:** Indipendente il più possibile dall'ambiente di esecuzione

4.2 Maven

Apache Maven: strumento di gestione e comprensione dei progetti software. Basato sul concetto di **Project Object Model (POM)**, Maven è in grado di gestire la compilazione, la reportistica e la documentazione di un progetto da un'informazione centrale.

4.2.1 Caratteristiche

- **Build Tool:** sono definite delle Build Lifecycle che permettono di configurare ed eseguire il processo di build.
- **Dependency Management:** Le dipendenze di progetto vengono specificate nel file di configurazione `pom.xml`. Maven si occupa di scaricarle in automatico da dei Repository remoti e salvarle in un repository locale.
- **Remote Repositories:** sono stati definiti dei repository remoti dove sono presenti gran parte delle librerie di progetti opensource e dei plugin utilizzati da maven per implementare e estendere le fasi dei Build Lifecycle
- **Universal Reuse of Build Logic:** Le Build Lifecycle, i plugin Maven permettono di definire in modo riusabile i principali aspetti richiesti per la gestione di progetto. Tra cui: l'esecuzione del processo di build, l'esecuzione di framework di test (Junit/TestNG), la creazione di template di progetto.

4.2.2 Build Lifecycle

Esistono 3 Build Lifecycles:

- **Default Lifecycle:** gestisce la distribuzione del progetto.
- **Clean Lifecycle:** gestisce la pulizia del progetto.
- **Site Lifecycle:** gestisce la creazione della documentazione del sito del progetto.

La Default Build Lifecycle è composta dalle seguenti fasi:

1. **Validate:** convalidare la correttezza del progetto e la disponibilità di tutte le info necessarie.
2. **Compile:** compilare il codice sorgente del progetto.
3. **Test:** testare il codice sorgente compilato utilizzando un framework di unit testing adeguato.
4. **Package:** prendere il codice compilato e confezionarlo nel suo formato distribuibile (e. `JAR`).
5. **Verify:** eseguire eventuali controlli sui risultati dei test di integrazione per garantire il rispetto dei criteri di qualità.
6. **Install:** installa il pacchetto nel repository locale, per utilizzarlo come dipendenza in altri progetti a livello locale.
7. **Deploy:** eseguito nell'ambiente di compilazione, copia il pacchetto finale nel repository remoto per condividerlo con altri sviluppatori e progetti.

4.3 POM

Project Object Model: unità di lavoro fondamentale in Maven. È un file `XML` che contiene informazioni sul progetto e dettagli di configurazione usati da Maven per costruire il progetto.

Alcune delle configurazioni che possono essere specificate nel POM sono:

- Project dependencies
- Plugin o goals che possono essere eseguiti
- Build profiles
- Altre info come: project version, description, developers, ...
- Project ID (group ID + artifact ID + version)

4.3.1 Project Archetypes

In breve, Archetype è un toolkit di template per progetti Maven. Un archetipo è definito come un modello o uno schema originale da cui si ricavano tutte le altre cose dello stesso tipo.
Il nome si adatta al fatto che stiamo cercando di fornire un sistema che fornisca un mezzo coerente per generare progetti Maven.
Archetype aiuterà gli autori a creare modelli di progetti Maven per gli utenti e fornirà agli utenti i mezzi per generare versioni parametrizzate di tali modelli di progetto.

4.3.2 Maven Plugin

Maven è un framework di base per un insieme di plugin Maven. I plugin vengono utilizzati per:

- Creare file `jar`
- Creare file `war`
- Compilare codice
- Codicec di Unit Test
- Creare Project Documentation

Quasi tutte le azioni che si possono pensare di eseguire su un progetto sono implementate come plugin Maven.

5 Software Testing

Software testing: indagine condotta per fornire alle parti interessate informazioni sulla qualità del prodotto o del servizio software sottoposto a test.

Testing: processo che consiste in tutte le attività del lifecycle, sia statiche che dinamiche, che riguardano la pianificazione, la preparazione e la valutazione dei prodotti software e dei relativi prodotti di lavoro per determinare che soddisfino i requisiti specificati, per dimostrare che sono adatti allo scopo e per rilevare i difetti.

5.1 Difetti nel Software

Il difetto può essere inserito sia dal Programmatore che dall'Analista:

- **Programmatore** (45%)
  1. fa un errore (mistake) durante la fase di sviluppo
  2. inserisce un difetto (fault o bug) all'interno del programma
  3. quando il programma viene eseguito, e la condizione non considerata dal programmatore si verifica, il difetto provocherà un comportamento inatteso del programma
  4. il programma avrà quindi una failure
- **Analista** (20% analisi requisiti, 25% progettazione)
  1. può introdurre un difetto, interpretando non correttamente un requisito
  2. il difetto viene introdotto nella fase di analisi e progettazione
  3. la progettazione del programma e la codifica possono essere influenzate dal difetto

5.2 Categorie di Testing

- **Funzionale:** Test condotti per valutare la conformità di un componente o di un sistema ai requisiti funzionali.
  - rappresentano cosa fa la nostra applicazione
  - tipicamente più semplici da progettare perché collegati alle funzioni richieste dal cliente
- **Non Funzionale:** Test condotti per valutare la conformità di un componente o di un sistema ai requisiti non funzionali (performance, sicurezza, usabilità, accessibilità).
  - rappresentano come la nostra applicazione risponde alle esigenze
- **Statico:** Testare un prodotto di lavoro senza l'esecuzione del codice.
  - analisi statica del codice
  - analisi e revisione dei documenti
  - analisi e revisione dei requisiti
- **Dinamico:** Collaudo che prevede l'esecuzione del software di un componente o di un sistema.
- **Verifica:** Il prodotto è stato realizzato secondo le specifiche (tecniche) e funziona correttamente.
- **Validazione:** Il prodotto è stato realizzato rispettando le specifiche dell'utente (requisiti)

5.3 Processo di Test

1. **Test planning:** l'attività di definizione o aggiornamento di un piano di test
2. **Test control:** azioni correttive e di controllo se il piano non viene rispettato
3. **Test analysis:** cosa testare
4. **Test design:** come testare
5. **Test implementation:** attività propedeutica all'esecuzione
6. **Test execution:** eseguire il test
7. **Checking result:** verificare i risultati e i dati collezionati dalla test execution per capire se il test è stato superato/fallito
8. **Evalutating exit criteria:** verificare se sono stati raggiunti gli exit criteria definiti nel test plan
9. **Test results reporting:** riportare il progresso rispetto agli exit criteria definiti nel test plan
10. **Test closure:** chiusura del processo e definizione azioni di miglioramento

5.4 7 Testing Principles

5.4.1 Testing show presence of difects

Testare un'applicazione può solo rivelare che uno o più difetti esistono nell'applicazione. Il test non può dimostrare che l'applicazione sia priva di errori. Pertanto, è importante progettare casi di test per trovare il maggior numero possibile di difetti.

5.4.2 Exhaustive testing is impossible

A meno che l'applicazione in prova abbia una struttura logica molto semplice e un input limitato, non è possibile testare tutte le possibili combinazioni di dati e scenari. Per questo motivo, il rischio e le priorità vengono utilizzati per concentrarsi sugli aspetti più importanti da testare. Strategie per selezionare i test baste:

- sul rischio (risk based testing): funzionalità che hanno impatto sul business
- sui requisiti (requirement based)

5.4.3 Early testing

Avviare la fase di test il prima possibile permette di risparmiare sui costi del progetto. Il processo di test non deve essere eseguito quando il progetto è al termine, ma deve andare in parallelo con il processo di sviluppo.

5.4.4 Defect clustering

Durante i test, si può osservare che la maggior parte dei difetti segnalati sono legati a un numero ridotto di moduli all'interno di un sistema. Un piccolo numero di moduli contiene la maggior parte dei difetti nel sistema. Questa è l'applicazione del principio di Pareto ai test del software: circa l'80% dei problemi si trova nel 20% dei moduli.

5.4.5 The pesticide paradox

Se continui a eseguire lo stesso set di test più e più volte (ad ogni nuova versione), siamo sicuri che non ci saranno più gli stessi difetti scoperti da quei casi di test. Poiché il sistema si evolve, molti dei difetti precedentemente segnalati vengono corretti. Ogni volta che viene risolto un errore o è stata aggiunta una nuova funzionalità, è necessario eseguire tutti i test (di non regressione) per assicurarsi che il nuovo software modificato non abbia interrotto vecchi errori. Tuttavia, anche questi casi di test di non regressione devono essere modificati per riflettere le modifiche apportate nel software per essere applicabili.

5.4.6 Testing is context dependent

Diverse metodologie, tecniche e tipi di test sono legati al tipo e alla natura dell'applicazione. Ad esempio, un'applicazione software in un dispositivo medico richiede più test di un software di giochi. Un software per dispositivi medici richiede test basati sul rischio, essere conformi con i regolatori dell'industria medica e possibilmente con specifiche tecniche di progettazione dei test.

5.4.7 Absence of errors fallacy

Solo perché il test non ha riscontrato alcun difetto nel software, non significa che il software sia perfetto e pronto per essere rilasciato. I test eseguiti sono stati davvero progettati per catturare il maggior numero di difetti? il software corrispondeva ai requisiti dell'utente?

5.5 V-model

Il modello a V spezza lo sviluppo del software in fasi successive. A ciascuna fase di sviluppo è associata lacorrispondente tipologia di test. Ci sono quattro tipologie di test Unit, Integration, System e Acceptance.

5.5.1 Unit testing

- Verificano l'unità: il più piccolo sottosistema possibile che può essere testato separatamente
- Veloci da eseguire
- Ogni modifica del codice sorgente dovrebbe scatenare l'esecuzione degli unit test
- Sono indipendenti tra di loro
- SUT è considerato come white box

5.5.2 Integration testing

- Verificano se sono rispettati i contratti di interfaccia tra più moduli o sub-system
- Verificano l'integrazione tra più sub-systems
- I Sub-Systems possono essere:
  - Sub-system Interni: già verificati dagli unit testing
  - Sub-system Esterni: database, filesystem, ...
- Gli "Integration testing" sono più lenti da configurare e da eseguire
- SUT è considerato come white box

5.5.3 System testing

- Verificano il comportamento dell'intero sistema
- Lo scopo principale dei system test è la verifica rispetto alle specifiche tecniche
- SUT è considerato come white box o black box

5.5.4 Acceptance testing

- Anche conosciuti come "UAT": User Acceptance Testing o "End User testing"
- Test suites su tutto il SUT, relativi agli use cases e ai requisiti concordati con l'utente finale/cliente
- Svolti con l'utente finale/cliente
- SUT è considerato come black box

6 Unit Testing

Unit Testing: Il testing di unità è il collaudo di singole unità software. Per unità si intende il minimo componente diun programma dotato di funzionamento autonomo (può essere una classe o una funzione a seconda del paradigma di programmazione).

Può essere sia manuale che automatico. Specialmente nel caso dello Unit Testing automatico, lo sviluppo dei test case è considerato parte integrante dell'attività di sviluppo.

6.1 A TRIP

6.1.1 Automatic

I test di unità devono essere eseguiti automaticamente. In ogni progetto deve essere disponibile un "automazione a comando" che permetta a tutti di invocare e far eseguire tutti o una parte dei test di unità in modo semplice. Durante la fase di sviluppo del progetto è importante che i test possano essere eseguiti:

- **In modo rapido:** i test devono essere semplici e la loro esecuzione deve richiedere pochi secondi.
- **Senza richiedere l'interazione umana:** il test non deve richiedere l'intervento umano, ad esempio per passare deiparametri.
- **In modo autonomo:** gli sviluppatori devono essere avvisati solo quando viene riscontrato un errore.

6.1.2 Thorough

Dei buoni test di unità devono essere esaustivi e accurati, devono verificare il comportamento di qualsiasi parte del progetto che potrebbe creare degli errori. Esistono degli strumenti che permettono di misurare se ogni parte del progetto è stata eseguita durante la fase di test, e possono calcolare:

- Percentuale di righe di codice
- Percentuale di possibili diramazioni
- Numero di eccezioni

6.1.3 Repeatable

I test di unità devono produrre sempre lo stesso risultato. Per essere ripetibili, i test di unità devono avere le seguenti caratteristiche:

- **Indipendenti dall'ordine di esecuzione:** l'ordine di esecuzione dei test di unità non deve influenzare il risultato. Per questo è necessario che i test siano indipendenti
- **Indipendenti dall'ambiente di esecuzione:** l'esecuzione dei test non deve dipendere da risorse esterne al progetto. Se alcune unità usano risorse esterne, è consigliato usare dei Mock Object per simulare il comportamento di queste componenti.

6.1.4 Independent

I test di unità devono essere indipendenti dall'ambiente di esecuzione, da elementi esterni al progetto e dall'ordine diesecuzione. Quando si scrive un test è consigliato verificare il comportamento di un singolo aspetto del progetto peridentificare univocamente l'errore. Se il test è indipendente, il suo comportamento sarà ripetibile nel tempo perchè non dipenderà dalle altre unità del progetto.

6.1.5 Professional

Devo essere scritti e manutenuti con lo stesso rigore con cui si scrive il codice di produzione. Il numero di righe di codice di test dovrebbe essere maggiore o uguale delle linee di codice in produzione.

6.2 Framework

Per creare i test di unità si sfruttano dei framework, con le seguenti caratteristiche:

- Configurazione l'ambiente di esecuzione del test.
- Selezione di un test o un insieme di test da eseguire.
- Analizzazione di valori aspettati, prodotti dalle unità.
- Standard per esprimere se il test è stato superato, se è fallito o se sono stati prodotti degli errori.

6.3 Right BICEP

6.3.1 Rigth

Dobbiamo porci la domanda "Are the results right?". In caso di ambiguità nei requisiti, i test di unità sono un buon punto di partenza per documentare il codice, ossia per documentare in che modo losviluppatore ha interpretato i requisiti e descrivere il comportamento delle unità realizzate.

6.3.2 Boundary Conditions

Solitamente gli errori accadono nelle condizioni limite, dove il programmatore non ha prestato attenzioni ai dettagli. Identificare le condizioni limiti è una delle parti più importanti nella creazione dei test di unità. I risultati devono essere **CORRECT**

- **C**onformance: i valori sono conformi al formato atteso?
- **O**rdering: i valori seguono o no un ordine?
- **R**ange: i valori sono dentro un minimo o massimo appropriato?
- **R**eference: i valori possono provenire da dati esterni non sotto il controllo del codice?
- **E**xistence: i valori esistono? non nulli, non zero, appartenenti ad un determinato insieme?
- **C**ardinality: i valori sono nella quantità desiderata?
- **T**ime: i valori rispettano un ordine temporale?

6.3.3 Check Inverse Relationship

Alcune unità possono essere controllate tramite l'applicazione della loro funzionalità inversa. Per esempio la radice quadrata: per testare una radice quadrata, elevo al quadrato l'output dell'unità e lo confronto col valore di input.

6.3.4 Cross-check Using Other Means

Usare un oracolo per verificare se la nuova unità ha lo stesso comportamento. Uso il vecchio sistema per verificareche il nuovo abbiamo lo stesso comportamento.

6.3.5 Force Error Condition

Nel mondo reale gli errori esistono e devono essere gestiti, per tanto è buona norma ricreare le condizioni di errore everificare che il progetto funzioni come previste in queste condizioni.

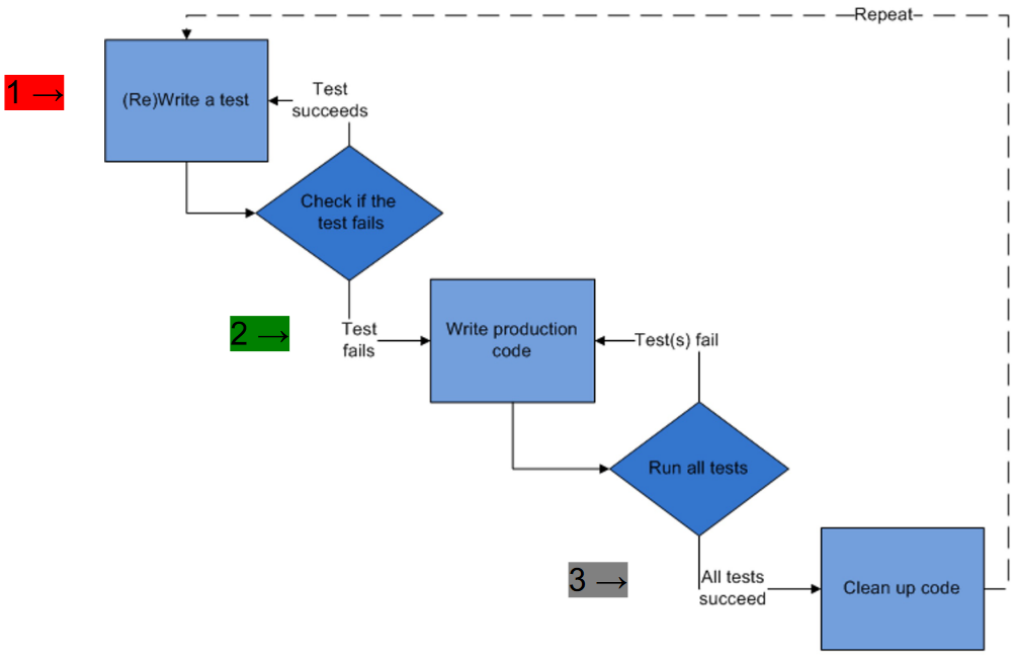
6.3.6 Performace Characteristic

Devono essere veloci perchè vengono eseguiti molto spesso.

6.4 TDD

Modello di sviluppo software che prevede che la stesura dei test automatici avvenga prima di quella del software, che deve essere sottoposto a test, e che lo sviluppo del software applicativo sia orientato esclusivamente all'obbiettivo di passare i test automatici precedentemente disposti.

1. (Re)Write a Test [rossa]: scrivi il test per la nuova funzione. Se ha successo riscrivi il test, se fallisce scrivi il codice
2. Write Production Code [verde]: sviluppa la quantita minima di codice per passare il test. Se fallisce, riscrivi il codice. Se riesce, pulisci il codice
3. Clean Up Code [grigia]: migliora la qualità de codice



7 Analisi Statica

Analisi Statica: analisi di un software che non richiede l'esecuzione del codice per essere eseguita.

A differenza della dinamica che viene eseguita quando il programma è in esecuzione. Può essere fatta sul codice sorgente o sul codice oggetto. Il termine è associato all'analisi effettuata da uno strumento automatico, mentre l'analisi fatta da un essere umano è detta, program understanding, program comprehension o code review. È un tipo di test:

- **Statico:** non ho bisogno di eseguire codice.
- **White box:** ho bisogno di vedere il codice.
- **Non funzionale:** controllo in che modo il è strutturato, non quello che fa.

7.1 Automated Code Review Software

Automated code review software: verifica il codice sorgente per la conformità con un insieme di regole predefinite obest practices.

7.1.1 Caratteristiche

- Rappresenta una pratica standard.
- Può essere fatto sia a mano che automatizzato.
- Solitamentemostra una serie di warning con l'elenco delle violazioni riscontrate.
- Può anche fornire un sistema per correggere gli errori trovati.

7.1.2 Strumenti

- Possono essere utilizzate per assistere la Automated Code Review.
- Non sono efficaci come il controllo umano ma possono essere eseguiti più velocemente.
- Incapsulano al loro interno una conoscenza profonda delle regole e della semantica per eseguire l'analisi
- Permettono al verificatore di non necessitare dello stesso livello di esperienza di un esperto del settore.

7.2 Teoria delle Finestre Rotte

La teoria delle finestre rotte è una teoria criminologica sulla capacità del disordine urbano e del vandalismo di generare criminalità aggiuntiva e comportamenti antisociali. La teoria afferma che mantenere e controllare ambienti urbani reprimendo i piccoli reati, gli atti vandalici, la deturpazione dei luoghi, il bere in pubblico, la sosta selvaggia o l'evasione nel pagamento di parcheggi, mezzi pubblici o pedaggi, contribuisce a creare un clima di ordine e legalità e riduce il rischio di crimini più gravi.

Ad esempio l'esistenza di una finestra rotta potrebbe generare fenomeni di emulazione, portando qualcun altro a rompere un lampione o un idrante, dando così inizio a una spirale di degrado urbano e sociale.

7.3 Tools

- **Checkstyle:** strumento di sviluppo che aiuta i programmatori a scrivere codice Java che aderisce a uno standard di codifica. Automatizza il processo di verifica del codice Java.
- **FindBugs/SpotBugs:** programma che utilizza l'analisi statica per cercare i bug nel codice Java.
- **PMD:** analizzatore statico di codice sorgente, che trova i più comuni difetti di programmazione.
- **SonarQube:** strumento di revisione automatica del codice per individuare bug, vulnerabilità e code smell nel codice.
  Funzionalità:
  - storicizza l'andamento della qualità
  - verifica se c'è un miglioramento o un deterioramento del progetto nel tempo
  - stabilisce un insieme di regole da applicare al progetto [quality profile]
  - verifica se la qualità del progetto rispetta determinati standard [quality gate]
  - classifica issue in base alla gravità (blocker, critical, major, minor, info)
  - classifica issue in: vulnerabilità , bug, code smell
  - revisiona le issue segnalate e segna i falsi positivi

7.3.1 Funzionalità

Le funzionalità sono simili a quelle di un correttore automatico che segnala un errore. Permettono di:

- Imporre il rispetto di convenzioni e stili
- Verificare la congruità della documentazione
- Controllare metriche e indicatori
  - complessità ciclomatica
  - grafo dipendenze
  - numerosità righe di codice
- Ricercare codice copiato in più punti
- Ricercare errori comuni nel codice
- Misurare la percentuale di codice testato
- Ricercare indicatori di parti incomplete

8 Continuous Integration

Continuous Integration: allineamento frequente dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso.

8.1 Motivazioni

- Per lunghi periodi di tempo, durante il processo di sviluppo, il progetto non è in uno stato funzionante o in uno stato utilizzabile. Sopratutto in progetti dove si sviluppa in un singolo ramo di sviluppo (centralized work-flow).
- Nessuno è interessato a provare ad eseguire l'intera applicazione fino a quando non è finito il processo di sviluppo.
- In questi progetti spesso viene pianificata la fase di integrazione alla fine del processo di sviluppo. In questa fase gli sviluppatori effettuano attività di merge ed effettuano attività di verifica e validazione.

8.1.1 Problema: Integration Hell

La fase di integrazione può richiedere molto tempo e nel caso peggiore nessuno ha modo di sapere quanto terminerò e di conseguenza quando l'applicazione può essere rilasciata

8.1.2 Soluzione: Continuous integration

La CI consente ad un team di intensificare l'attività di sviluppo e test integrando gli sviluppo il più spesso possibile.

8.2 Processo

- Al completamento di un'attività viene costruito il prodotto: ogni volta che uno sviluppatore invia un commit al VCS viene eseguito il processo di build (compilazione e test).
- 2 casi:
  - Successo: la modifica è integrata nel software senza causare errori tra quelli testati.
  - Fallimento: se il processo di costruzione fallisce l'attività non continua fino a che il prodotto non viene riparato. Se non è possibile ripararlo si ritorna all'ultima versione funzionante.
- In questo modo si assicura la presenza di un prodotto consistente potenzialmente pronto per essere validato e rilasciato

8.2.1 Prerequisiti

Per implementare la pratica di CI è necessario che:

- Il codice del progetto venga gestito in un VCS
- Il processo di build del progetto sia automatico
- Il processo di build esegua delle verifiche automatiche (test di unità, test di integrazione, analisi statica del codice)
- il team di sviluppo adotti correttamente questa pratica

8.2.2 Processo in dettaglio

1. Controllo se il processo di build è in esecuzione nel sistema di CI:
    - se è in esecuzione, aspetto che finisca
    - se fallisce, lavoro con il team in modo da sistemare il problema
2. Quando il processo di build ha terminato con successo, aggiorno il codice nel mio workspace con il codice del VCS ed effettuo l'integrazione in locale.
3. Eseguo il processo di build in locale in modo da verificare che tutto funzioni correttamente:
    - se ha successo, invio le modifiche al VSC
    - altrimenti, risolvo il problema in locale e riprovo
4. Attendo che il sistema di CI esegua il processo di build con i miei cambiamenti:
    - se fallisce, sistemo il problema in locale e riprendo dal punto 3
    - se ha successo, passo allo sviluppo dell'attività successiva

8.3 Practices

8.3.1 Integrare frequentemente gli sviluppi

In questo modo:

- le modifiche da integrare saranno poche e più facile da gestire.
- Se viene segnalato un errore dall'esecuzione del processo di build sarà più semplice identificare il codice che ha introdotto l'errore

8.3.2 Creare una Automated Test Suite

Se non vengono eseguiti dei test automatici per verificare e validare il progetto, il processo di build può solo verificare se il codice integrato compila correttamente. I test che possono essere eseguiti nel processo di CI sono:

- Test di unità
- Test di integrazione
- Analisi statica

8.3.3 Avere un processo di Build and Test breve

Se il processo di Build è lento:

- Gli sviluppatori smetteranno di eseguire il processo di build e i test prima di inviare le modifiche al VCS.
- Ci saranno delle build eseguite in maniera concorrente quando il sistema è libero. Sarà più difficile individuarela build che ha causato l'errore.
- Si disincentiva l'invio frequente delle modifiche al VCS

8.3.4 Gestire il nostro Workspace

Ogni sviluppatore deve essere in grado di:

- Eseguire in locale il processo. Per questo motivo nel VCS devo esserci i file per:
  - build: codice di produzione.
  - test: codice di test.
  - deploy: script richiesti per configurare ambiente di esecuzione del progetto.
- **Always Be Prepared to Revert to the Previous Revision:** scaricare le modifiche dal VCS e essere in grado di ripristinare il progetto ad uno stato consistente.
- **Never Go Home on a Broken Build:** verificare l'esito della compilazione nel CI server. Se il processo CI fallisce lo sviluppatore deve risolvere il problema o ripristinare la versione VCS all'ultimo stato consistente.

8.3.5 Fix una build in errore subito

Se non è possibile correggere l'errore che ha fatto fallire la build velocemente, ripristinare lo stato del VCS all'ultima versione funzionante. Non è ammesso correggere il problema commentano le verifiche che hanno fatto fallire la build.

8.3.6 Tutti possono vedere cosa succede

Ogni componente del team deve poter vedere:

- Lo stato della build deve essere pubblicato in un servizio visibile a tutto il team di progetto.
- Lo stato del progetto e capire qual'è.
- L'ultima versione in cui è stato eseguito il processo di build con successo.

In caso di fallimento del processo di build deve essere possibile:

- Identificare chi ha introdotto l'errore.
- Avere a disposizione in log per identificare quale parte della build è fallita.
- Avere a disposizione una lista dei commit che hanno introdotto l'errore.

9 Continuous Delivery

Continuous Delivery: approccio di ingegneria del software in cui i team producono software in cicli brevi, assicurando che il software possa essere rilasciato in modo affidabile in qualsiasi momento e, quando lo rilasciano, lo fanno manualmente.

Lo scopo è costruire, testare e rilasciare software con maggiore velocità e frequenza. Serve per ridurre i costi, il tempo e il rischio di cambiamenti nel rilascio rendendo maggiormente incrementale il rilascio delle applicazioni in produzione.

CD è una disciplina di sviluppo software con la quale il software viene costruito in modo tale che possa essere rilasciato in produzione in qualsiasi momento. Nella pratica questo significa che:

- Il software è rilasciabile attraverso tutto il suo ciclo di vita.
- Il team da la priorità a mantenere il software rilasciabile rispetto ad aggiungere nuove funzionalità.
- Chiunque può ottenere un feedback automatizzato sulla readiness dei sistemi ogni qualvolta qualcuno fa una modifica.
- Si possono ottenere rilasci di qualsiasi versione del software in qualsiasi ambiente a comando.
- Si integrano le pratiche di CI del team di sviluppo integrando il building degli eseguibili e eseguendo test automatizzati sugli eseguibili.

L'obiettivo è trasformare il rilascio di un sistema di qualsiasi dimensione in un processo prevedibile che può essere eseguito a richiesta. Il modo per farlo è assicurarsi che il codice sia sempre pronto al rilascio, anche nel caso di cambiamenti continui da parte di team di sviluppatori molto grandi. Tutte le parti successive alla fine dello sviluppovengono eliminate.

9.1 Problemi

La Continuous Integration permette di avere feedback su problemi introdotti dagli sviluppatori. Si focalizza sulla parte DEV e assicura che:

- Il codice compili
- Vengono eseguiti i test di unità, integrazione, accettazione e l'analisi statica

Questo non è sufficiente per garantire la possibilità di rilasciare il prodotto ad ogni modifica perché le attività che di solito fanno perdere più tempo avvengono nella fase di rilascio e test (e nella comunicazione e la collaborazione tra DEV TEST e OPS)

9.1.1 Problemi Comuni

- **OPS:** i sistemisti aspettano tempo per riceve la documentazione con le procedure di rilascio.
- **TEST:** i tester attendono tempo per effettuare verifiche e validazione nella versione giusta.
- **DEV:** il team di sviluppo riceve segnalazioni di bug su funzionalità che sono state rilasciate da settimane.
- **Archiettura:** ci si rende conto solo alla fine dello sviluppo che l'architettura scelta non permette di soddisfare i requisiti non funzionali.

9.1.2 Conseguenze

- **Non rilasciabile:** perchè si è impiegato troppo a farlo entrare in produzione.
- **Contiene difetti:** contiene difetti perchè il ciclo di feedback tra team di development, testing e operations è troppo lungo.

9.2 Deployment Pipeline

L'obiettivo è quello di migliorare il processo che permette di rilasciare una modifica al codice sorgente del progetto in produzione.

Una **Value Chain** è una modellazione del processo per misurare:

- Il Valore Globale
- Il Valore residuo da inserire nella "Value Chain" (il miglioramento)

Una **Deployment Pipeline** dà gli stessi risultati di una Value Chain negli altri settori non software:

- Controllare le prestazioni
- Migliorarne l'efficienza
- "Fast is cheap": le Pipeline permetto di trovare i problemi il prima possibile

Deployment Pipeline: modellazione del processo di Deployment tramite una successione di fasi (stages) e verifiche (gates).

- Il passaggio da una fase all'altra viene verificato tramite il superamento di una verifica
- Il passaggio di una fase può scatenare una notifica
- È guidato dal concetto di Fail-Fast
- È Composta da stages mappate su attività misurabili (build, test)
- La transazione tra due stages viene definita gate
- Può essere automatica o manuale
- I gates scatenano il passaggio allo stage successivo
- Gli Stages possono essere eseguiti in parallelo e/o in sequenza
- I gates possono essere multi direzionali

9.3 Realizzazione

- Avere un rapporto di lavoro collaborativo con tutti i partecipanti (dev, test e ops)
- Utilizzare le Deployment Pipelines per definire il processo di build, test e deploy dell'applicazione.
- Distribuire il processo di build e deploy su più ambienti

9.3.1 Requisiti

- **Continuous Integration:**
  - VCS
  - Build automation
  - Unit Testing
  - Artifact Repository
- **Configuration Management:** strumenti che ci permettono, di gestire tramite codice, la configurazione degli ambienti dove dovrà essere rilasciato il nostro software
- **Continuous Testing:** test automatici a livello di sistema funzionali e non funzionali
- **Orchestratore:** sistema software che ci permette di modellare le esecuzioni delle pipeline

9.4 Practices

9.4.1 Only Build your Binaries once

Eseguire il processo di build UNA sola volta. Lo stesso artefatto verrà utilizzato per le verifiche in ogni ambiente. In questo modo:

- L'artefatto che verrà rilasciato in produzione è lo stesso che è stato verificato e validato nelle stages della pipeline
- Forma di ottimizzazione (faccio il lavoro solo una volta)

È consigliabile avere un Artifact Repository dove rilasciare gli artefatti. L'artefatto generato deve essere indipendente dall'ambiente di esecuzione. Tenere il codice separato dall'ambiente di esecuzione.

9.4.2 Deploy the Same Way to Every Environment

Utilizzare lo stesso script per effettuare il rilascio in ambienti differenti. Lo script di rilascio sarà più solido perché verrà verificato maggiormente.

9.4.3 Smoke-Test your Deployments

Per verificare se il rilascio è andato bene prevedere l'esecuzione di smoke-test. Gli smoke-test devono anche verificareil corretto funzionamento dei sub-system esterni L'obbiettivo è sempre quello del fail fast, bisogna cercaregli errori in modo da far fallire la pipeline il più velocemente possibile.

9.4.4 Deploy into a Copy of Production

Prevede di avere a disposizione un ambiente con le stesse caratteristiche (o simili) dell'ambiente di produzione. Quindi è necessario eseguire i test e gli script di rilascio in ambienti il più possibile simili all'ambiente di produzione. La gestione dell'ambiente di rilascio è il Configuration Management. L'ambiente dovrà avere:

- La stessa configurazione di rete
- Lo stesso SO
- Lo stesso stack applicativo
- Gestione dei dati consistente

9.4.5 Each Change should Propagate through the Pipeline Instantly

Ogni modifica al codice sorgente deve avviare la pipeline. La pipeline impiegà molto tempo per eseguire l'intero processo e servono verifiche negli stages che la facciano fallire il prima possiible. La CI non eseguirà ad ogni commitè sarà più difficile identificare l'errore. Per questo si consiglia un server CI che possa eseguire il processo di CI apartire da uno specifico commit. In questo modo sarà più semplice effettuare attività di debug per identificare la modifica che ha fatto fallire il processo

9.4.6 If any Part of the Pipeline Fails, Stop the Line

Eseguire per primi i controlli veloci e meno esaustivi. In questo modo si può far fallire la pipeline e notificare il team del problema.

9.5 Benefici

- **Ridurre il rischio legato al deploy:** dal momento che si sta distribuendo piccole modifiche, è meno probabile corrompere il sistema ed è più facile risolvere l'errore in caso di problemi.
- **Velocizza il time to market:** questa pratica porta ad avere rilasci più frequenti
- **Maggiori feedback da parte degli utenti**
- **Progressi tangibili:** molti team diversi monitorano i progressi. Cioè la semplice conferma da parte degli sviluppatori che qualcosa è stato fatto è molto meno forte rispetto a "fatto, verificato e distribuito in un ambiente di produzione".
- **Minor costo:** diretta conseguenza dell'automazione
- **Prodotti migliori:** che soddisfano le aspettative degli utenti
- **Team meno stressati e più collaborativi**
- **Maggiore documentazione implicita**