# **Scrum Roles**

* Product Owner
* Scrum Master
* Team member

## Product Owner

Il Product Owner è responsabile di massimizzare il valore del prodotto coordinando il backlog, definendo priorità e requisiti. Collabora con gli stakeholder per comprendere le esigenze degli utenti finali, definisce la visione del prodotto e crea criteri di accettazione per le storie. Rappresenta il business e i clienti, assicurando che il team sviluppi ciò che è più prezioso per il successo del prodotto, mantenendo una comunicazione continua con il team e garantendo che le decisioni siano allineate alla strategia aziendale.

## Scrum Master

Lo Scrum Master è responsabile di facilitare e guidare un team nello sviluppo Agile secondo Scrum. Agisce come coach per aiutare il team a comprendere e adottare correttamente i principi Agile. Il suo obiettivo è creare un team auto-organizzato e autonomo, rimuovendo gli ostacoli che impediscono il progresso. Fornisce supporto costante, agisce come esperto di Scrum e facilitatore della comunicazione tra il team e gli stakeholder. Il ruolo include anche la promozione di una cultura Agile all'interno dell'organizzazione e la protezione del team da interferenze esterne.

## Team Member

Un membro del team Agile collabora all'interno del team Scrum per raggiungere gli obiettivi del progetto. Le sue responsabilità principali includono l'auto-organizzazione, la stima degli sforzi, l'utilizzo di strumenti e tecniche adeguate, e la collaborazione con gli altri membri del team. Mostra flessibilità nel lavorare fuori dalla propria zona di comfort quando necessario e contribuisce con una varietà di competenze. Il team Scrum include il Product Owner, lo Scrum Master e i membri del team.

# **Agile Contracts**

## Contract models

* **Fixed Price Fixed Scope FPFS:** riguarda un approccio alla gestione dei progetti in cui sia il prezzo che l'ambito delle funzionalità sono stabiliti e non possono essere modificati. Questo metodo è spesso preferito per la sua prevedibilità e facilità di gestione, ma presenta anche delle criticità, specialmente nel contesto dello sviluppo Agile.
* **Variable Price Variable Scope VPVS:** riguarda un approccio flessibile alla gestione del pricing di un progetto, particolarmente utile in ambienti di sviluppo software Agile. Questo metodo permette di stabilire un tetto massimo per il prezzo delle iterazioni del progetto, mantenendo al contempo una flessibilità sull'ambito delle funzionalità da sviluppare.
  + **Capped-Price**, **Variable-Scope**
  + **Capped-Price**, **Partial-Fixed-Scope**
  + **Fixed-Price**, **Variable-Scope**

## Pricing schemes

* **Lump Sum/Fixed Price**: Questo modello di pagamento prevede che il cliente paghi un prezzo stabilito per l'intero prodotto o progetto. Viene utilizzato quando l'ambito del progetto è fisso e ben definito in anticipo.
* **Time and Materials - TM**: Questo modello di pagamento prevede che il cliente paghi sia per il tempo effettivamente impiegato dalle persone che lavorano sul progetto, sia per i materiali e le risorse utilizzate.
* **Fixed price per iteration**: Questo modello di pagamento prevede che il cliente paghi un prezzo fisso per ciascuna iterazione del processo Agile. Ogni iterazione rappresenta un ciclo di sviluppo completo, alla fine del quale viene rilasciata una parte funzionante del prodotto.
* **Fixed price per Unit of Work**: Questo modello di pagamento prevede che il cliente paghi un prezzo fisso per ciascuna unità di lavoro completata. L'unità di lavoro è una quantità specifica di software misurata in un modo definito, come funzionalità completate, punti di storia, o moduli consegnati.
* **Hybrid (Shared pain/gain)**: Questo modello di pagamento combina elementi di prezzo fisso con incentivi basati sulle performance. Prevede un prezzo fisso entro un certo intervallo, con variazioni di pagamento in base alla tempestività della consegna del prodotto. Il fornitore riceve un bonus se consegna il prodotto prima del tempo stimato o una riduzione se impiega più tempo del previsto.
  + **Target cost**
  + **Multi-phase variable model**
  + **Profit sharing**
* **Pay per Use**: Questo modello di pagamento prevede che il cliente paghi in base all'uso effettivo del sistema o del servizio. Il cliente paga solo per il tempo in cui utilizza il sistema e può interrompere i pagamenti quando smette di usarlo.

# Agile Contracts - Describe agile principles

1. **Individuals and interactions over processes and tools**: building projects around motivated individual trusting them to get the job done, conveying information to and within a development team through face-to-face conversation, having self-organizing teams that reflects on how to become more effective at regular intervals, valuing individuals and interactions over processes and tools.
2. **Working software over comprehensive documentation**: simplicity, technical excellence and good design, having working software as the primary measure of progress, valuing it more than comprehensive documentation.
3. **Customer collaboration over contract negotiation**: business people and developers must work together daily to the project, maintaining a constant pace of development sustainable for sponsors, developers and users.
4. **Responding to change over following a plan**: welcome changing requirements even late in development to meet customer’s changing needs and enhance its competitiveness, early and continuous delivery of valuable working software with a frequency varying from couples of weeks to couples of months.

# Technical Debt

Imperfections in a software system that were caused by lack of time or by intentional choices and that run the risk of causing higher future maintenance cost.

## Code smells

In the context of object-oriented programming, code smells are methods and classes that violate the principles of good object-oriented design: **clearly defined single responsibility**, **encapsulation**, **information hiding**, **few and clear interfaces**, **proper use of inheritance**.

Code smells are categorized in three types:

* **identity disharmonies**: Identity disharmonies occur when the structure of classes and their responsibilities are not well-balanced. One common example is the **God Class**.
* **collaboration disharmonies**: Collaboration disharmonies occur when the interactions between classes and methods are too tightly coupled, making the code hard to change and test. An example is **Intensive Coupling**. Intensive Coupling occurs when a method in a class calls too many methods from a few unrelated classes. This creates a web of dependencies that makes the system fragile.
* **classification disharmonies**: Classification disharmonies occur when classes do not conform to their intended roles or the expected structure of an inheritance hierarchy. An example is a **Tradition Braker**. A Tradition Braker is a subclass that provides services unrelated to those provided by its base class, not specializing in inherited services, and breaking the expected behavior of the hierarchy.

## TD types

* **Defect debt**: E.g., latent defects not yet fixed
* **Design/architecture debt**: E.g., bad organization of classes
* **Documentation debt**: E.g., outdated or incomplete documentation
* **Testing debt**: E.g., missing/not executed test cases/plans

# The 4 freedoms of Free Software

* **Freedom 0**: The freedom to run the program as you wish, for any purpose
* **Freedom 1**: The freedom to study how the program works, and change it so it does your computing as you wish
* **Freedom 2**: The freedom to redistribute copies so you can help your neighbor
* **Freedom 3**: The freedom to distribute copies of your modified versions to others. By doing this you can give the whole community a chance to benefit from your changes. (Access to the source code is a precondition for this)

## Free software

* A software is free software (not for free) if the user can enjoy all the four freedoms
* The 4 freedoms are made possible by using a copyright license

## Copyleft

Il termine copyleft si riferisce a una pratica nel campo del diritto d'autore che permette di utilizzare, modificare e distribuire liberamente un'opera, a condizione che tutte le copie e le versioni modificate siano anch'esse soggette alle stesse condizioni di libertà.

## GNU general public license (GPL, 1989)

* **4 freedoms + copyleft obligation**: "what is free, must remain free.”
* **Copyleft** applies also to modified free software

## Classification of licences

* **Non copy-left**: e.g., BSD, MIT, Apache
* **Strong copy-left**: e.g., GNU-GPL
* **Weak copy-left**: not all derivative works inherit the copyleft license (e.g., GNU-LGPL, MPL)
* **Network copy-left**: code available also for remote users (e.g., GNU-AGPL, EUPL)

# DevOps

**DevOps** è un insieme di pratiche che combina lo **sviluppo software** (Dev) e le **operazioni IT** (Ops) con l'obiettivo di migliorare la collaborazione tra i team di sviluppo e operazioni, automatizzare i processi di sviluppo e deployment, e aumentare la velocità e la qualità delle release del software. **DevOps** mira a creare un ciclo di sviluppo software più efficiente e reattivo, migliorando la comunicazione e l'integrazione tra i vari team coinvolti.

## Principi Fondamentali di DevOps

1. **Collaborazione e Comunicazione**:
   1. Migliorare la comunicazione tra i team di sviluppo e operazioni per allineare gli obiettivi e ridurre i silos.
   2. Promuovere una cultura di collaborazione e responsabilità condivisa per l'intero ciclo di vita del software.
2. **Automazione**:
   1. Automatizzare i processi ripetitivi e manuali, come il testing, il deployment e il monitoraggio.
   2. Utilizzare strumenti di automazione per creare pipeline di integrazione e distribuzione continua (CI/CD).
3. **Integrazione Continua (CI)**:
   1. Integrare frequentemente il codice sviluppato nei repository condivisi, eseguendo automaticamente i test per identificare e risolvere rapidamente i problemi.
4. **Distribuzione Continua (CD)**:
   1. Automatizzare il processo di deployment per consentire rilasci frequenti e affidabili del software in produzione.
   2. Minimizzare il rischio e l'intervento manuale durante il deployment.
5. **Monitoraggio e Feedback Continuo**:
   1. Monitorare costantemente l'applicazione e l'infrastruttura per rilevare problemi e migliorare le prestazioni.
   2. Utilizzare feedback continuo per informare lo sviluppo e apportare miglioramenti rapidi.
6. **Gestione della Configurazione**:
   1. Utilizzare strumenti per gestire e automatizzare la configurazione dell'infrastruttura, garantendo consistenza e riproducibilità.

## Benefici di DevOps

1. **Rilasci Più Veloci e Frequenti**:
   1. Ridurre i tempi di rilascio del software grazie all'automazione e all'integrazione continua.
   2. Aumentare la frequenza dei rilasci, migliorando la capacità di risposta ai cambiamenti e alle richieste degli utenti.
2. **Maggiore Affidabilità e Qualità**:
   1. Migliorare la qualità del software attraverso il testing automatizzato e il monitoraggio continuo.
   2. Ridurre il numero di errori e migliorare la stabilità delle release.
3. **Efficienza Operativa**:
   1. Automatizzare i processi manuali e ripetitivi, liberando tempo per attività a maggior valore aggiunto.
   2. Migliorare l'utilizzo delle risorse e la scalabilità dell'infrastruttura.
4. **Collaborazione Migliorata**:
   1. Favorire una cultura di collaborazione tra i team di sviluppo e operazioni, riducendo i conflitti e migliorando la comunicazione.
5. **Risposta Rapida ai Problemi**:
   1. Rilevare e risolvere rapidamente i problemi grazie al monitoraggio continuo e al feedback immediato.

## Componenti e Strumenti DevOps

1. **Continuous Integration (CI) e Continuous Delivery (CD)**:
   1. Strumenti come Jenkins, Travis CI, CircleCI per automatizzare il build, il testing e il deployment del software.
2. **Version Control**:
   1. Utilizzo di sistemi di controllo versione come Git per gestire il codice sorgente e le modifiche.
3. **Infrastructure as Code (IaC)**:
   1. Strumenti come Terraform, Ansible, Puppet, Chef per gestire e automatizzare la configurazione dell'infrastruttura.
4. **Containerizzazione e Orchestrazione**:
   1. Utilizzo di Docker per creare container che isolano le applicazioni e le loro dipendenze.
   2. Utilizzo di Kubernetes per orchestrare e gestire i container su larga scala.
5. **Monitoring e Logging**:
   1. Strumenti come Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana) per monitorare e analizzare le prestazioni delle applicazioni.
6. **Collaboration Tools**:
   1. Utilizzo di piattaforme come Slack, Microsoft Teams, Jira per migliorare la comunicazione e la gestione dei progetti tra i team.

### Il Ciclo di Vita DevOps

1. **Pianificazione**:
   1. Definire le caratteristiche e le funzionalità da sviluppare.
   2. Creare storie utente e backlog di prodotto.
2. **Sviluppo**:
   1. Scrivere e integrare il codice continuamente.
   2. Eseguire test automatici per assicurare la qualità.
3. **Build e Test**:
   1. Costruire automaticamente il software e eseguire test per rilevare errori il prima possibile.
4. **Rilascio**:
   1. Automatizzare il processo di deployment per rilasci frequenti e affidabili.
5. **Distribuzione**:
   1. Distribuire il software in ambienti di produzione utilizzando pipeline CI/CD automatizzate.
6. **Monitoraggio**:
   1. Monitorare le prestazioni e il comportamento del software in produzione.
7. **Feedback**:
   1. Raccogliere feedback dagli utenti e dal monitoraggio per migliorare continuamente il prodotto e il processo.