# Laboratorio di Sviluppo delle Applicazioni Software

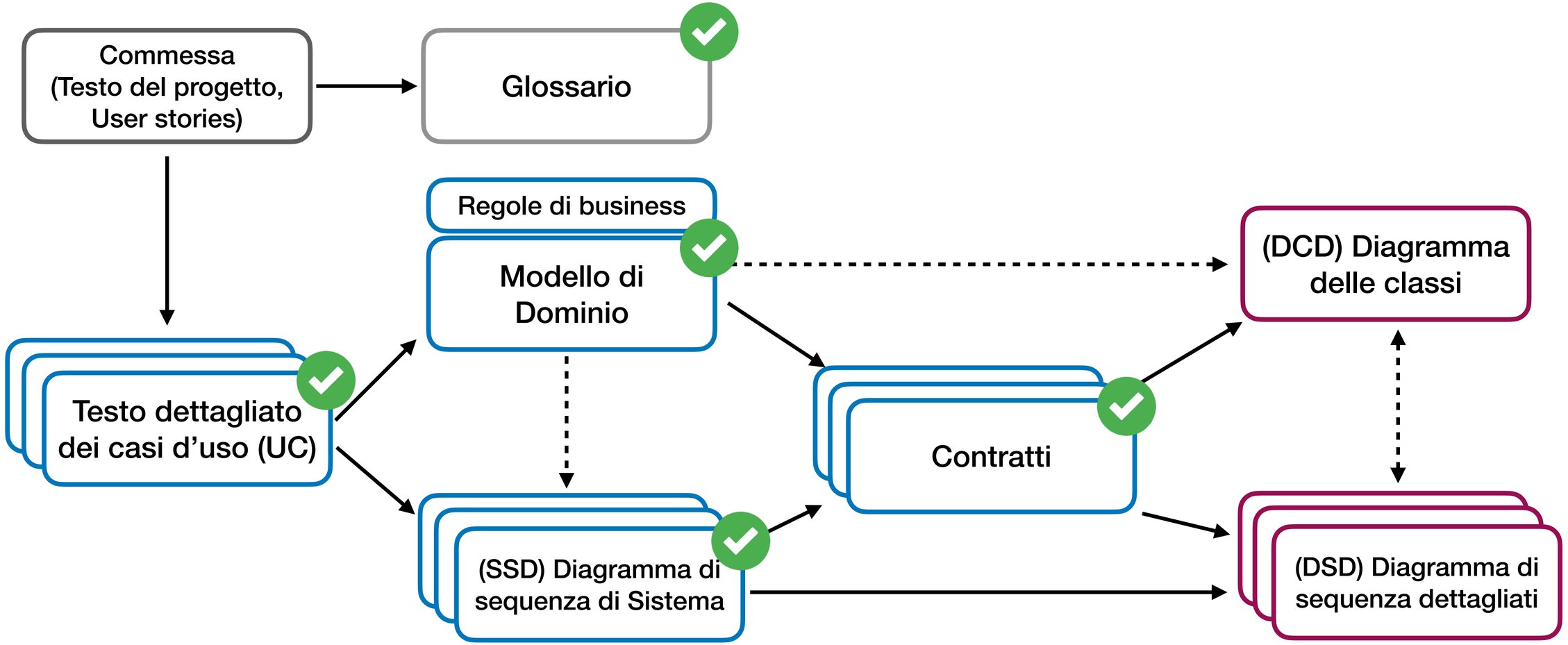
## Testing

# Punto della situazione

## Da dove partiamo

# Documentazione (codice escluso)

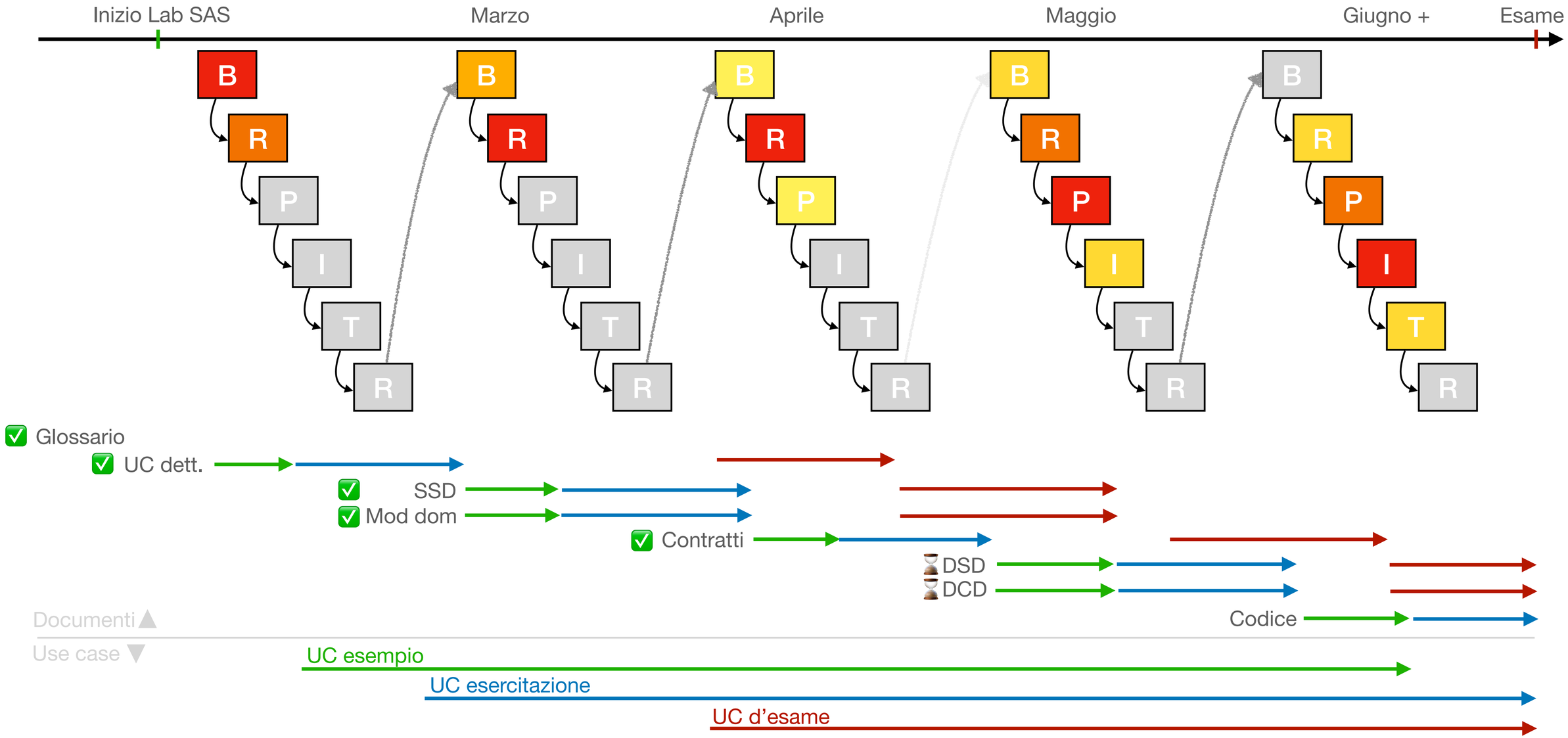
## Artefatti della metodologia UP




Requisiti

Progettazione

# Evoluzione degli artefatti di UP



# Testing

## Una veloce introduzione

# Come valutare la correttezza del software?

## Metodi formali

- **Analisi statica (Static analysis)**
  - Analisi basata sul testo del programma
  - Evidenzia errori, più difficile dimostrare la correttezza (in termini di esecuzione) del programma

- **Model checking**
  - Analizza il comportamento del programma, tramite un modello e proprietà desiderate
  - Limitato all’esplosione (in termini di numero) degli stati o modelli infiniti

- **Theroem Proving**
  - Approccio logico-deduttivo
  - Può richiedere elevate competenze

# Come valutare la correttezza del software?

## Testing

### Scopo

Assicurare che il **software soddisfa i requisiti**, ovvero faccia ciò che è supposto fare. Verificare che **non siano presenti errori, difetti o discrepanze** fra il comportamento del sistema e i requisiti.

### Approccio

**Eseguire il sistema implementato in esperimenti controllati** per validare i risultati prodotti (e anche le performances, aspetti di sicurezza, …).

È possibile testare singole componenti, la loro interazione e integrazione, o il sistema completo. Può essere fatto manualmente o – più realisticamente – **automatizzato** tramite tecnologie dedicate (e.g. JUnit per Java).

JUnit 5

# Tipi di testing

**Unit testing**

Verifica il corretto funzionamento di singole unità di codice, come funzioni, metodi, classi…. In isolamento. Esempio: può evidenziare errori funzionali.

**Integration testing**

Verifica la corretta interazione di singole unità quando vengono integrate insieme. Esempio: può evidenziare errori nelle comunicazione e flusso di dati.

**System testing**

Verifica l'intero sistema in un ambiente analogo a quello in produzione. Può testare sia aspetti funzionali che non funzionali, per esempio prestazioni.

**Acceptance testing**

Verifica la conformità alle specifiche richieste e l'adeguatezza per il rilascio. Può includere l'alpha e beta testing in ambienti reali prima del rilascio e essere usato come validazione contrattuale prima del pagamento.

**Performance testing**

Verifica efficienza, scalabilità, …

**… altri tipi di testing**

# Esempio JUnit per Unit testing

```java
import static org.junit.Assert.*;
import org.junit.Test;

public class BankAccountTest {

    @Test
    public void testWithdrawSufficientFunds() {
        // Creazione di un oggetto BankAccount con saldo i
        BankAccount account = new BankAccount(1000);

        // Prelevamento di 500
        account.withdraw(500);

        // Verifica che il saldo sia corretto dopo il prel
        assertEquals(500, account.getBalance());
    }
```

- Test per il metodo **`withdraw(double amount)`** della classe **`BankAccount`**
- Tramite il costrutto **`assertEquals`** verifica che il saldo dopo il prelievo sia corretto (post-condizione)

# Esempio JUnit per Unit testing

```java
@Test
public void testWithdrawInsufficientFunds() {
    // Creazione di un oggetto BankAccount con saldo ini
    BankAccount account = new BankAccount(1000);

    // Tentativo di prelevare più del saldo disponibile
    account.withdraw(1500);

    // Verifica che il saldo non sia cambiato
    assertEquals(1000, account.getBalance());
}
```

- Verifica che il tentativo di prelevare più del saldo, **fallisca**: saldo invariato (post-condizione)
- I test sono eseguiti automaticamente, potenzialmente ad ogni modifica del codice.
- Forniscono un risultato tipo *pass/fail* specificando la condizione che non è soddisfatta.

# Come testare?

## Il testing può solamente riportare errori evidenziati dai test, non la loro assenza!

**I test dovrebbero caratterizzare il comportamento desiderato e dimostrare l'assenza di errori, almeno quelli più plausibili** – un'arte? Può richiedere una buona dose di esperienza.

- Testare i **requisiti**: esempio, farsi guidare da pre- e post-condizioni
- Testare i **casi difficili**: esempio, un balance vuoto o negativo, esempi positivi e negativi, …
- Testare **sorgenti di errori note:** esempio, una variabile indefinita, un array vuoto, un valore problematico (0 in una divisione), un file non presente …
- …
- **Sapere/capire quando fermarsi!** Non si vuole testare il compilatore o il sistema operativo!!!

# Come organizzare il testing

## Diversi approcci possibili

- **Team distinti e indipendenti** fra sviluppatori e tester.
- In alcuni approcci, **i test vengono sviluppati prima dell'implementazione** del codice.
- Importante è definire in dettaglio un *testing plan*. Alcuni tipi di testing possono essere integrati e continui nel processo di software engineering.
- **Integrazione**, **documentazione**, feedback e planning sono aspetti rilevanti del testing.
- Il testing, per esempio l'*Acceptance testing*, può avere implicazioni contrattuali.

# Testing nel Torneo SAS

## Come avvengono le sfide

# Scopo del gioco

## Idea principale

- Il Torneo SAS ha lo scopo principale di sviluppare un analisi critica e **auto-valutazione** del progetto d'esame
- Il tipo di testing atteso nel Torneo si SAS è *analogo al tipo di conoscenza che ci si aspetta dagli studenti all'esame*.
- Il Torneo SAS è un "serious game" per permettere agli studenti di **testare e affinare la propria preparazione sul progetto di esame**.

# Testing nel Torneo di SAS

## Cosa e come testare – regole del gioco (I)

- Ogni team prepara 8 domande scritte su aspetti rilevanti che riguardano i seguenti documenti del progetto di esame (2 domande ciascuno): **Casi d'Uso Dettagliati**, **Modello di dominio**, Diagrammi sequenza (**SSD**) e **Contratti**
- Nello spirito del testing, le domande sono volte a verificare la conformità dei documenti con i requisiti, e a cercare di validare l'assenza di potenziali errori.
- Le domande sono formulate in italiano in un linguaggio informale, ma preciso, per esempio:
  - *"Il contratto relativo a … garantisce che … nel caso in cui …?"*
  - *"Il diagramma di sequenza relativo a … in che modo copre il caso in cui … ?"*
  - *"Cosa succede se …. non è stato precedentemente …?"*

# Testing nel Torneo di SAS

## Cosa e come testare – regole del gioco (II)

- Il torneo procede per match fra due squadre.
- In un match, ogni team **pone** all'altro team **due domande** scelte fra la lista presentata (una domanda può essere usata solamente una volta), e **ogni membro dell'altro team risponde ad una e una sola domanda**.
- Le domande devono essere concise e il team ha a disposizione due minuti per rispondere, usando il proiettore come supporto (preparare i documenti necessari sul portatile).
- La classe e il docente (con *golden share*) vota quale team passa il turno.