CASO 1: Febbraio 2024 - Teoria

Domanda Tullio: "Illustrare il rapporto tra processo e progetto secondo ISO 12207"

Risposta Studente Valutata Positivamente (da correzione Cardin)

```
Secondo ISO 12207, un processo è un insieme di attività interrelate
che trasformano input in output. È una risorsa organizzativa permanente
che definisce il "come" si svolgono le attività, indipendentemente
dal contesto specifico.

Un progetto è uno sforzo temporaneo finalizzato alla creazione di un
prodotto unico, con vincoli di tempo, costo e scope definiti. Il
progetto istanzia uno o più processi per raggiungere i propri obiettivi.

La relazione è many-to-many: un processo può essere istanziato da
molti progetti (es. processo di testing usato in progetti diversi);
un progetto istanzia molti processi (es. sviluppo, verifica, gestione
configurazione).

Diversamente dal progetto che "muore" al termine, il processo persiste
come asset riutilizzabile. Confondere i due porta a perdita di
conoscenza organizzativa: ogni progetto ricrea da zero procedure
che potrebbero essere standardizzate.

Nel contesto del nostro capitolato, il progetto ha istanziato processi
ISO 12207 come Configuration Management (applicato a tutte le baseline)
e Quality Assurance (eseguito ad ogni milestone), mantenendo la stessa
struttura procedurale ma con parametri adattati alla fase.
```

Risposta Studente Insufficiente (da errori comuni annotati)

```
Il processo è il metodo con cui si svolge il lavoro, il progetto
è il lavoro da svolgere. I processi sono usati nel progetto per
completare le attività. Nel nostro progetto abbiamo usato vari
processi per arrivare al risultato finale.
```

Analisi comparativa:

Elemento	Risposta Positiva	Risposta Insufficiente
Ancoraggio normativo	"Secondo ISO 12207" esplicito	Assente

Elemento	Risposta Positiva	Risposta Insufficiente
Definizione processo	"insieme attività interrelate, input→output"	"metodo con cui si svolge lavoro" (circolare)
Definizione progetto	"sforzo temporaneo, vincoli tempo/costo/scope"	"lavoro da svolgere" (generico)
Relazione	"many-to-many, istanziazione"	"usati nel progetto" (vago)
Contrasto	"Diversamente da... permanente vs temporaneo"	Assente
Esempio concreto	"Configuration Management su baseline RTB/PB/CA"	"vari processi" (non specificati)
Implicazione	"Confondere porta a perdita conoscenza"	Assente

CASO 2: Luglio 2024 - Teoria

Domanda Tullio: "Spiegare il principio di Dependency Inversion con esempio"

Risposta Studente Valutata Positivamente

```
Il Dependency Inversion Principle (Robert Martin, SOLID) stabilisce
che moduli ad alto livello non devono dipendere da moduli a basso
livello; entrambi devono dipendere da abstrazioni. Le abstrazioni
non devono dipendere dai dettagli, ma i dettagli dalle abstrazioni.

Questo inverte la dipendenza rispetto al flusso di controllo: il
modulo chiamante definisce l'interfaccia di cui ha bisogno, il
modulo chiamato la implementa. Si ottiene mediante interfacce o
classi astratte che fungono da contratto.

Diversamente dall'approccio tradizionale dove business logic dipende
da infrastruttura concreta (es. classe DAO specifica), DIP rende
la business logic indipendente dai dettagli implementativi. Questo
rispetta anche Open/Closed: cambiare implementazione non richiede
modifica del client.

Nel nostro backend, il service layer dipende dall'interfaccia
IUserRepository, non dalla classe PostgreSQLUserRepository. Quando
abbiamo sostituito PostgreSQL con MongoDB per requisiti di scalabilità,
abbiamo creato MongoUserRepository implementando la stessa interfaccia,
senza toccare una riga del service layer.

Violare DIP in sistemi safety-critical è particolarmente rischioso:
se la logica di controllo insulina dipende direttamente da driver
```

```
hardware specifici, diventa impossibile testare la logica di dosaggio
senza hardware reale, aumentando il rischio di bug fatali non rilevati.
```

Risposta Studente Problematica (da annotazioni correzione)

```
Dependency Inversion vuol dire che le dipendenze devono essere
invertite usando interfacce. Invece di dipendere da classi concrete
si dipende da interfacce. Questo rende il codice più flessibile.

Nel nostro progetto abbiamo usato interfacce per disaccoppiare
i componenti e permettere di cambiare implementazione facilmente.
```

Analisi comparativa:

Elemento	Risposta Positiva	Risposta Problematica
Ancoraggio	"Robert Martin, SOLID" + definizione doppia	Solo "Dependency Inversion"
Meccanismo	"inversione rispetto a flusso controllo, contratto"	"usando interfacce" (come, non perché)
Relazione SOLID	Connessione esplicita a OCP	Assente
Contrasto	"Diversamente da approccio tradizionale DAO"	Assente
Esempio tecnico	"IUserRepository, PostgreSQL→MongoDB, service layer"	"interfacce per disaccoppiare" (generico)
Conseguenze	"Testing senza hardware in safety-critical"	"più flessibile" (vago)

CASO 3: Settembre 2023 - Teoria

Domanda Tullio: "Differenza tra Verifica e Validazione secondo IEEE"

Risposta Studente Valutata Positivamente

```
Secondo IEEE 1012, Verifica risponde a "Are we building the product
right?" - conferma che il prodotto è conforme alle specifiche.
Validazione risponde a "Are we building the right product?" -
conferma che il prodotto soddisfa i bisogni reali dell'utente.

Verifica è un processo tecnico e oggettivo: testing contro requisiti,
code review, analisi statica, ispezioni formali. Si basa su criteri
predefiniti e misurabili.
```

```
Validazione è empirica e soggettiva: acceptance testing con utenti
reali, usability study, feedback su prototipi. Richiede interazione
con stakeholder e può rivelare requisiti impliciti.

Diversamente dalla verifica che può essere automatizzata (unit test,
integration test), la validazione richiede giudizio umano. È possibile
verificare completamente un prodotto sbagliato; è impossibile validare
senza verifica perché il software non funzionante non può essere valutato.

Nel nostro progetto, il modulo di autenticazione OAuth2 ha superato
verifica con test coverage 98% e conformità RFC 6749. La validazione
con 5 utenti pilota ha rivelato che il flusso multi-step confondeva
utenti non tecnici, richiedendo redesign UX mantenendo logica già
verificata.

In ambito medicale, un dispositivo può superare tutti i test di verifica
(conformità IEC 62304) ma fallire validazione clinica FDA se il
protocollo non riflette condizioni d'uso reali (es. pulsossimetro
accurato in lab ma impreciso su pelle scura).
```

Risposta Studente Insufficiente

```
Verifica controlla che il software funzioni correttamente. Validazione
controlla che il software faccia quello che deve fare. La verifica
si fa con i test, la validazione con gli utenti.

Nel nostro progetto abbiamo fatto verifica testando i moduli e
validazione facendo provare il sistema ai committenti.
```

Analisi comparativa:

Elemento	Risposta Positiva	Risposta Insufficiente
Ancoraggio	"IEEE 1012" + domande canoniche	Assente
Natura verifica	"tecnico, oggettivo, criteri predefiniti"	"funzioni correttamente" (ambiguo)
Natura validazione	"empirica, soggettiva, requisiti impliciti"	"faccia quello che deve" (circolare)
Tecniche	Testing/review/analisi vs acceptance/usability	"test" vs "utenti" (troppo generico)
Contrasto	"automatizzabile vs giudizio umano"	Assente
Esempio tecnico	"OAuth2, RFC 6749, 98% coverage, flusso multi-step"	"testando moduli" (quale? come?)
Possibile errore	"verificare prodotto sbagliato"	Assente

Elemento	Risposta Positiva	Risposta Insufficiente
Safety-critical	"Pulsossimetro FDA, IEC 62304, bias pelle scura"	Assente

---

CASO 4: Giugno 2023 - Teoria

Domanda Tullio: "Spiegare PoC vs MVP nel contesto di sviluppo software"

Risposta Studente Valutata Positivamente (da soluzioni pubblicate)

```
Proof of Concept (PoC) è un artefatto throwaway che dimostra fattibilità
tecnica di un approccio incerto. È codice usa-e-getta, non evolutivo,
focalizzato su "può essere fatto?" Non ha requisiti di qualità produzione.

Minimum Viable Product (MVP) è la versione più semplice del prodotto
rilasciabile che fornisce valore agli utenti. È codice production-ready,
evolutivo, focalizzato su "quale valore minimo convince gli early adopters?"
Deve rispettare standard di qualità, sicurezza, manutenibilità.

Diversamente dal PoC che viene scartato dopo validazione tecnica, MVP
entra in produzione e costituisce la base per incrementi futuri. PoC
risponde a rischi tecnici, MVP a rischi di mercato/prodotto.

Nel nostro progetto, abbiamo sviluppato PoC per validare integrazione
con API esterna mal documentata: 200 righe di script Python per confermare
autenticazione e parsing response. Una volta confermata fattibilità,
lo script è stato eliminato e abbiamo implementato integrazione in
TypeScript nel backend production con error handling, retry logic, logging.

Confondere i due è pericoloso: portare PoC in produzione introduce
debito tecnico critico (security vulnerabilities, no error handling).
Trattare MVP come PoC significa sprecare effort su codice di qualità
che verrà scartato.
```

Risposta Studente Problematica

```
PoC serve a provare che una cosa funziona, MVP è la prima versione
del prodotto. Il PoC è una prova, l'MVP è un prodotto vero.

Nel nostro progetto abbiamo fatto un PoC per vedere se l'idea
funzionava, poi abbiamo sviluppato l'MVP per il cliente.
```

Analisi comparativa:

Elemento	Risposta Positiva	Risposta Problematica
Definizione PoC	"throwaway, fattibilità tecnica, non evolutivo"	"provare che funziona" (generico)
Definizione MVP	"versione minima rilasciabile, valore utente"	"prima versione prodotto" (impreciso)
Qualità	"PoC no standard produzione, MVP production-ready"	Non menzionato
Domande chiave	"Può essere fatto?" vs "Quale valore minimo?"	Assente
Contrasto	"scartato vs evolutivo, rischio tecnico vs mercato"	"prova vs prodotto" (superficiale)
Esempio tecnico	"200 righe Python API, eliminato, reimplementato TypeScript con error handling"	"idea funzionava" (quale idea? come?)
Rischi	"PoC in produzione = vulnerabilities, MVP usa-e-getta = spreco"	Assente

---

CASO 5: Febbraio 2024 - Teoria

Domanda Tullio: "Cosa significa 'way of working' e perché è rilevante nella gestione progetto?"

Risposta Studente Valutata Positivamente

```
Way of Working (WoW) secondo ISO/IEC/IEEE 24774 è l'insieme di
pratiche, convenzioni, strumenti e processi che un team adotta
per svolgere il lavoro. Include metodologia (Scrum, Kanban),
toolchain (Git, CI/CD), standard di codifica, cerimonie, e metriche.

È rilevante perché determina velocità, qualità e predicibilità del
team. Un WoW esplicito e condiviso riduce ambiguità, accelera
onboarding, facilita retrospettive basate su evidenze.

Diversamente dalla semplice scelta metodologica (Agile vs Waterfall),
WoW è la contestualizzazione operativa: due team Scrum possono avere
WoW drasticamente diversi (sprint duration, definition of done,
review frequency).

Nel nostro progetto, il WoW includeva: sprint bisettimanali, daily
standup asincroni su Slack, pull request obbligatorie con 2 approval,
CI con SonarQube (>80% coverage requirement), retrospettive con metrica
velocity trend. Questo ha permesso di identificare collo di bottiglia
```

```
nel code review (3-day median time) e risolverlo con pair programming.

Cambiare WoW mid-project senza consenso esplicito genera caos: nel
secondo sprint un membro ha bypassato review "per velocità", introducendo
bug che ha richiesto 2 giorni di fix, nullificando il presunto risparmio
temporale e danneggiando fiducia nel team.
```

Risposta Studente Insufficiente

```
Way of Working è il modo in cui il team lavora insieme. È importante
perché permette di organizzarsi meglio e lavorare più efficientemente.

Nel nostro progetto abbiamo definito il Way of Working all'inizio
stabilendo come comunicare e quali strumenti usare.
```

Analisi comparativa:

Elemento	Risposta Positiva	Risposta Insufficiente
Ancoraggio	"ISO/IEC/IEEE 24774" + componenti specifici	Assente
Componenti	"metodologia, toolchain, standard, cerimonie, metriche"	"comunicare, strumenti" (generico)
Rilevanza	"velocità, qualità, predicibilità, onboarding"	"organizzarsi meglio" (vago)
Contrasto	"vs metodologia: contestualizzazione operativa"	Assente
Esempio tecnico	"Sprint duration, 2 approval PR, SonarQube >80%, velocity trend"	"stabilendo all'inizio" (cosa?)
Evidenza misurabile	"3-day median code review time"	Assente
Conseguenza violazione	"Bypass review → 2 giorni fix, danno fiducia team"	Assente

---

Pattern Linguistici Distintivi Tullio

Marker Positivi (presenti in risposte approvate)

1. **"Secondo [Standard]..."** → ISO 12207, IEEE 1012, Robert Martin
2. **"Diversamente da..."** → Sempre contrasto esplicito
3. **"Questo ha permesso/impedito..."** → Conseguenze causali
4. **Metriche concrete** → "98% coverage", "3-day median", "200 righe Python"

5. **"In sistemi safety-critical..."** → Escalation conseguenze
6. **"Confondere X con Y porta a..."** → Dimostrazione comprensione profonda
7. **Nomi tecnici specifici** → OAuth2, RFC 6749, SonarQube, PostgreSQL

Anti-Pattern (presenti in risposte insufficienti)

1. **Definizioni circolari** → "processo è il metodo", "PoC prova che funziona"
2. **"Più flessibile/efficiente/organizzato"** → Aggettivi senza quantificazione
3. **"Nel nostro progetto abbiamo usato..."** → Senza dettagli tecnici
4. **"Importante perché..."** → Senza meccanismo causale
5. **Assenza standard** → Mai citato ISO/IEEE/RFC
6. **Esempi generici** → "testando i moduli", "vari processi", "strumenti"

Formula Algoritmica Precisa

```
TULLIO_RESPONSE =
  ANCHOR(standard + definizione formale) +
  EXPAND(2-3 componenti/aspetti + relazioni) +
  CONTRAST("Diversamente da..." + alternativa + differenza critica) +
  EXEMPLIFY(nome_tecnico + azione_specifica + metrica/outcome) +
  IMPLICATE(conseguenza_violazione | safety_critical_context)

dove:
  len(ANCHOR) ≈ 2 righe
  len(EXPAND) ≈ 2-3 righe
  len(CONTRAST) ≈ 2 righe
  len(EXEMPLIFY) ≈ 3-4 righe con nomi propri
  len(IMPLICATE) ≈ 2 righe

  TOTAL ≈ 8-10 righe (mai oltre)
```

Checklist Pre-Consegna (5 secondi)

```
☑ Ho citato standard/autore nel primo paragrafo?
☑ Ho usato "Diversamente da..." o "A differenza di..."?
☑ L'esempio contiene NOMI TECNICI (non "moduli", ma "OAuth2")?
☑ Ho quantificato con numeri/metriche dove possibile?
☑ Ho spiegato conseguenze di errore/violazione?
☑ Lunghezza < 12 righe?
```

Conclusione operativa: Le risposte Tullio-approved condividono struttura invariante: ancoraggio normativo → espansione meccanicistica → contrasto differenziale → esempio nominale con metriche → implicazione safety/costi. Le insufficienti sono circolari, generiche, prive di standard, senza contrasti né conseguenze misurabili.