Ingegneria del Software

Filippo Tonietto

29 gennaio 2024

Indice

1 Quiz Teoria

1.1 Domanda 1

Il SEMAT associa 4 gradi progressivi di avanzamento (*milestone*) all'attività di codifica, denominati "*architecture selected*", "*demonstrable*", "*usable*", "*ready*". Nel nostro progetto didattico abbiamo usato il concetto di *baseline*, istanziandolo in TB e PB. Collega tali nozioni, proponendo una corrispondenza tra la TB e la PB del progetto didattico, singolarmente, e qualcuna delle *milestone* citate.

✓a. La TB, essendo associata a un PoC, corrisponde bene alla *milestone* "*demonstrable*"; la PB invece sta tra "*usable*" e "*ready*".

b. Nessuna associazione è possibile, perché non vi è alcuna corrispondenza tra *milestone* e *baseline*, che sono concetti tra loro indipendenti.

c. Più corrispondenze sono possibili, in ragione dell'interpretazione che vogliamo dare di TB e PB.

1.2 Domanda 2

Pensando alla funzione del *repository*, per contenuto e destinatari, all'interno di un progetto *software*, scegli una tra le seguenti affermazioni:

✓a. L'uso di un *repo* permette di dare ordine e stabilità al lavoro collaborativo. La natura dei prodotti di progetto ne determina l'organizzazione interna.

b. Il *repo* contiene esclusivamente artefatti *software* in sviluppo o in manutenzione, va centralizzato (per progetto), e serve **solo** ai programmatori. Nessun altro deve avervi accesso.

c. Il *repo* è risorsa personale e privata di ogni singolo sviluppatore, che ne determina anche i contenuti. La funzione primaria del *repo* è proteggere lo stato del lavoro di quella persona da intrusioni e malfunzionamenti.

1.3 Domanda 3

Uno dei compiti ricorrenti del responsabile di progetto (*project manager*) è la compilazione del preventivo a finire (PaF), al termine di ogni periodo di rendicontazione. Vogliamo sapere se tale atto implichi o meno la ripianificazione delle attività rimanenti nel tempo residuo.

a. Il piano di progetto è da modificare solo in conseguenza dell'emergere di gravi criticità: doverlo modificare, pertanto, è "cattiva notizia". Quanto più possibile, dunque, conviene preservare il piano iniziale, cosicché la compilazione dei PaF avvenga naturalmente per sottrazione del consuntivo dei consumi rilevati nel periodo di rendicontazione.

b. Certamente no. La compilazione dei PaF è un atto di natura strettamente contabile: tutto ciò che serve in esso è tenere conto dei consumi consuntivati nel periodo di rendicontazione, eventualmente modificando l'impiego dei ruoli, per assicurare che il costo totale del progetto non ecceda quanto pattuito contrattualmente.

✓c. La ripartizione del tempo di progetto in brevi periodi, con obiettivi e risorse contingentati, ognuno dei quali culminante in una retrospettiva, serve ad attenuare i rischi e a favorire il miglioramento continuo. Tale retrospettiva riguarda il rapporto tra l'avanzamento conseguito e il consumo riscontrato. Dalla sua analisi scaturiscono indicazioni utili alla rivisitazione del piano delle attività rimanenti, e da esso la determinazione dei PaF.

1.4 Domanda 4

Vostri colleghi di lavoro discutono se la copertura dei requisiti, come indicatore quantitativo, sia determinabile in un qualunque momento del progetto, oppure lo sia solo al momento dei *test* di sistema. Vedendovi freschi di studi, vi chiedono di selezionare la più plausibile tra le seguenti affermazioni al riguardo.

a. Per copertura dei requisiti intendiamo la percentuale di *test* funzionali superati con successo. In tale accezione, tale valore può essere determinato solo dopo tali *test*.

b. Per copertura dei requisiti intendiamo la proporzione di requisiti utente, espressi nel capitolato, presi in carico formalmente dal fornitore. In tale accezione, l'indicatore viene determinato in sede di revisione dei requisiti, alla stipula del contratto.

✓c. Per copertura dei requisiti intendiamo la percentuale di requisiti software associati a specifici artefatti/componenti/parti del prodotto come determinati dalla progettazione (*design*); la successiva campagna di *test* serve a confermare l'effettivo grado di copertura conseguito a valle della codifica.

1.5 Domanda 5

In una discussione con un vostro collega senior, il tema vira sull'utilità dei *test* di unità. Il vostro collega è scettico al riguardo, soprattutto perché - dice - non vi è una definizione chiara e univoca di cosa sia una "unità" in un programma *software*. Il vostro interlocutore vi chiede di esprimervi sul tema, scegliendo una tra le seguenti risposte:

a. Vero. La nozione di "unità" *software* non è ben definita, e questo è fonte di confusione indesiderabile. Meglio concentrarsi sui *test* di sistema, dove non vi sono dubbi sulla natura dell'oggetto di *test*.

b. Dipende. Per certi linguaggi di programmazione e certi tipi di programmi, il *test* di unità o non serve o è troppo oneroso. Per altri, invece, vi è buona sinergia tra la tecnologia e la verifica e allora le cose "vanno da sole".

✓c. Falso. Cosa sia l'elemento unitario del programma *software* è una nozione architetturale sufficientemente chiara: una parte del tutto, piccola abbastanza da essere verificata individualmente (priva di catene di dipendenze significative), ma grande abbastanza da meritarsi verifica (cioè con correttezza non accertabile "a vista").

1.6 Domanda 6

I professionisti dello sviluppo *software* concordano, in generale, che i metodi agili siano da preferire. In un colloquio di lavoro, il vostro intervistatore vi chiede di provare a mettere in relazione lo sviluppo incrementale con quello agile, scegliendo una tra le seguenti risposte:

✓a. Entrambi guardano nella stessa direzione, ma lo sviluppo agile contempla il rischio di iterazioni distruttive, che invece lo sviluppo incrementale cerca in ogni modo di evitare.

b. Si tratta di visioni radicalmente diverse e tra loro incompatibili. Lo sviluppo agile è flessibile, e adattivo. Quello incrementale è pieno di vincoli.

c. Perfetta identità. Ogni metodo di sviluppo agile è intrinsecamente incrementale.

1.7 Domanda 7

La metrica "efficacia" misura il grado di conformità del risultato di una attività con le attese, mentre la metrica "efficienza" è funzione inversa della quantità di risorse impiegate per raggiungere l'obiettivo prefissato, oltre il minimo necessario. Ritieni vi sia un ordine di precedenza, in pianificazione, nel fissare gli obiettivi quantitativi di tali metriche?

✓a. Pianificando, prima decido fino a che livello voglio soddisfare le attese, e poi calcolo il costo corrispondente, iterando fino a quando i costi preventivati siano entro gli eventuali vincoli contrattuali.

b. Prima di tutto decido quanto posso e voglio spendere, e poi, con quello, vedo fin dove posso andare.

c. Nessun ordine di precedenza: si tratta di metriche completamente indipendenti tra loro.

1.8 Domanda 8

A lezione abbiamo visto che, secondo il SEMAT, il gruppo viene incaricato (*team*) di svolgere un lavoro (*work*) applica un *way of working* per organizzare e condurre le attività necessarie. Vogliamo sapere cosa venga fissato prima tra *team*, *work* e *way of working*.

✓a. Il *way of working*: prima di ogni altra cosa, si istituisce una prassi (le norme che guidano il lavoro); il resto segue.

b. Il lavoro (*work*) da svolgere: prima occorre determinare quale sia il lavoro da svolgere; il resto segue.

c. Il *team*: prima di tutto, si decide chi dovrà svolgere il lavoro; il resto segue.

2 Quiz Pratica

2.1 Domanda 1

Si selezionino tra le seguenti affermazioni corrette relative alle architetture a monolite e a microservizi.

✓a. Formando un sistema fortemente distribuito, in un'architettura a microservizi si dovrebbe preferire uno stile di comunicazione asincrona ad uno stile sincrono.

✓b. Secondo la Legge di Conway, un'architettura a microservizi dovrebbe essere utilizzata soprattutto all'interno di organizzazioni che hanno gruppi di sviluppo per lo più indipendenti.

c. Un monolite è uno stile architetturale sorpassato, completamente rimpiazzato dai microservizi in ogni tipo di applicazione.

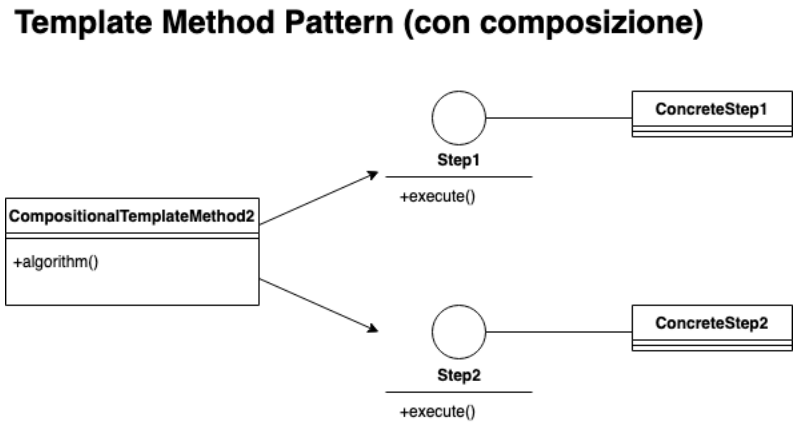
✓d. Un monolite di per sé non induce forzatamente all'accoppiamento tra le componenti che lo compongono.

e. E' possibile aggiornare solamente una piccola componente di un'architettura a monolite, senza dover effettuare il deploy dell'intera applicazione.

f. I microservizi possono condividere il medesimo database, leggendo e scrivendo dalle medesime tabelle, collezioni, etc. In questo modo, è possibile condividere le informazioni fra i microservizi, mantenendo però la completa segregazione del codice sorgente.

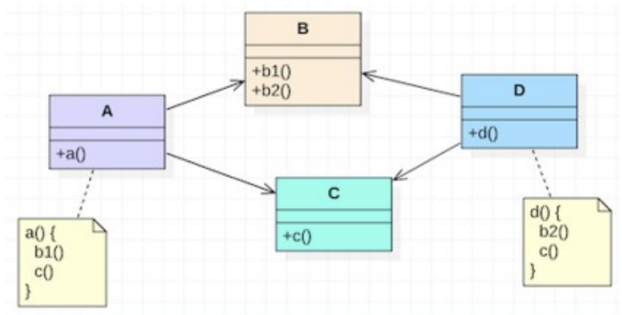
2.2 Domanda 2

Si fornisca il diagramma delle classi del *Template Method Pattern*. Si richiede, però, di non utilizzare l'ereditarietà, ma esclusivamente la composizione.



2.3 Domanda 3

Sia dato il seguente diagramma delle classi.



Si prenda in considerazione un sistema costituito unicamente dai tipi riportati nel diagramma. E' possibile affermare che il sistema abbia il massimo grado di coesione possibile?

O Vero

✓ Falso

2.4 Domanda 4

Si selezionino le affermazioni corrette fra le seguenti relative ai *SOLID principles*.

a. E' possibile che un gruppo di classi sia *Open-Close* anche se si utilizzano delle variabili globali, nei modi e termini permessi dal linguaggio di programmazione di riferimento.

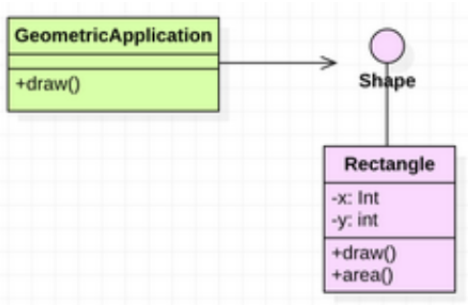
✓b. Usando il *Liskov Substitution Principle* è associabile al concetto di *Design by Contract*.

c. Un insieme di classi può essere definito *Open-Close* rispetto a qualsiasi tipo di modifica richiesta.

✓d. Il *Single Responsibility Principle* è anche noto con il nome di coesione.

2.5 Domanda 5

Sia data la seguente struttura, utilizzata da un'applicazione che visualizza figure geometrichce su uno schermo.



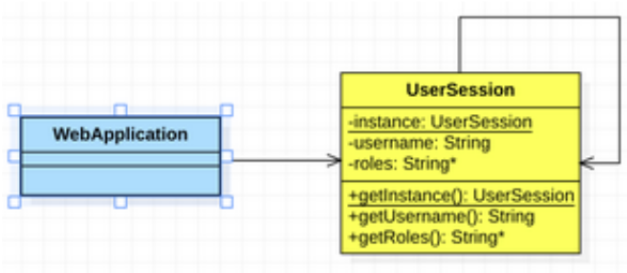
La classe Rectangle soddisfa il *Single Responsibility Principle*.

✓ Vero

O Falso

2.6 Domanda 5

Dato il seguente diagramma delle classi, si selezioni quali delle affermazioni sono corrette.



a. Le implementazioni del Singleton sono sempre *thread-safe*.

✓b. In Java, l'implementazione del Singleton accettata dalla comunità prevede l'uso di una enumerazione.

✓c. L'utilizzo del pattern architettura *Dependency Injection* permette di utilizzare una versione più evoluta del pattern Singleton, che ha meno controindicazioni.

✓d. L'utilizzo del pattern Singleton della GoF può interagire con la corretta stesura degli Unit Test.

e. Il diagramma delle classi per la classe UserSession implementa il design pattern Singleton classico.

f. E' corretto usare un Singleton per modellare una session utente di un'applicazione multiutente.

2.7 Domanda 6

Data la classe Singleton che partecipa al design pattern *singleton*, si fornisca una sua implementazione in linguaggio Java, che sia *thread safe* rispetto alla costruzione della singola istanza istance della classe.

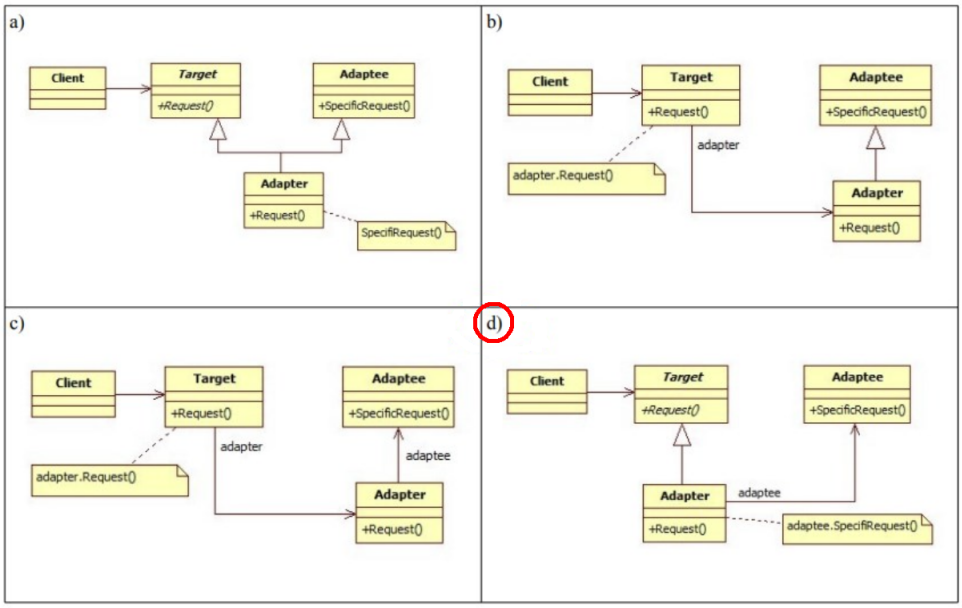
```
public enum EnumSingleton {
    INSTANCE("Info");
    private String info;

    private EnumSingleton(String info) {
        this.info = info;
    }

    public EnumSingleton getInstance() {
        return INSTANCE;
    }
}
```

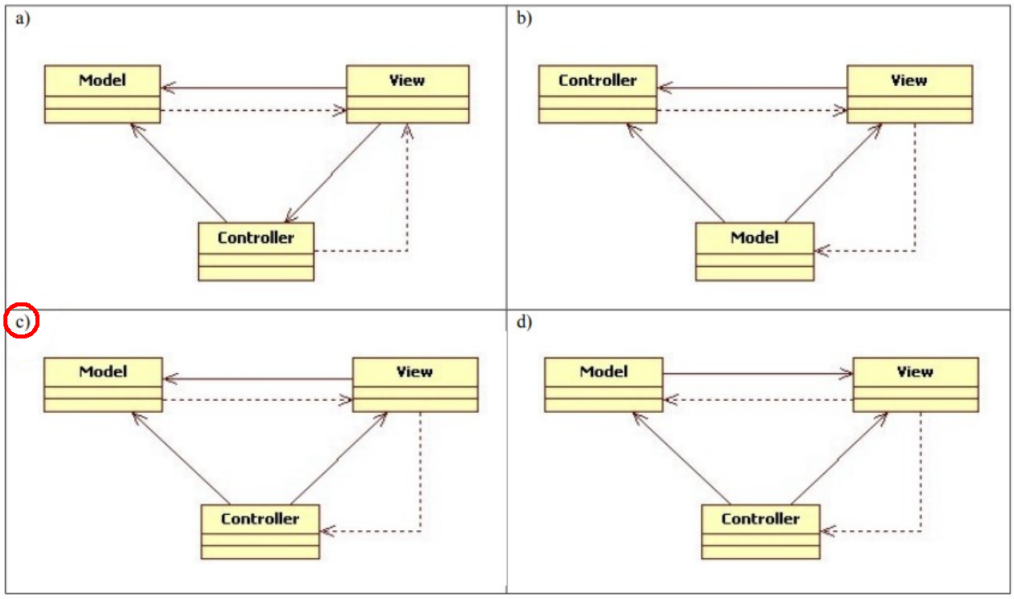
2.8 Domanda 7

Si indichi quale dei diagrammi delle classi sotto riportati modelli correttamente il *design pattern* Adapter nella sua variante *object adapter*.



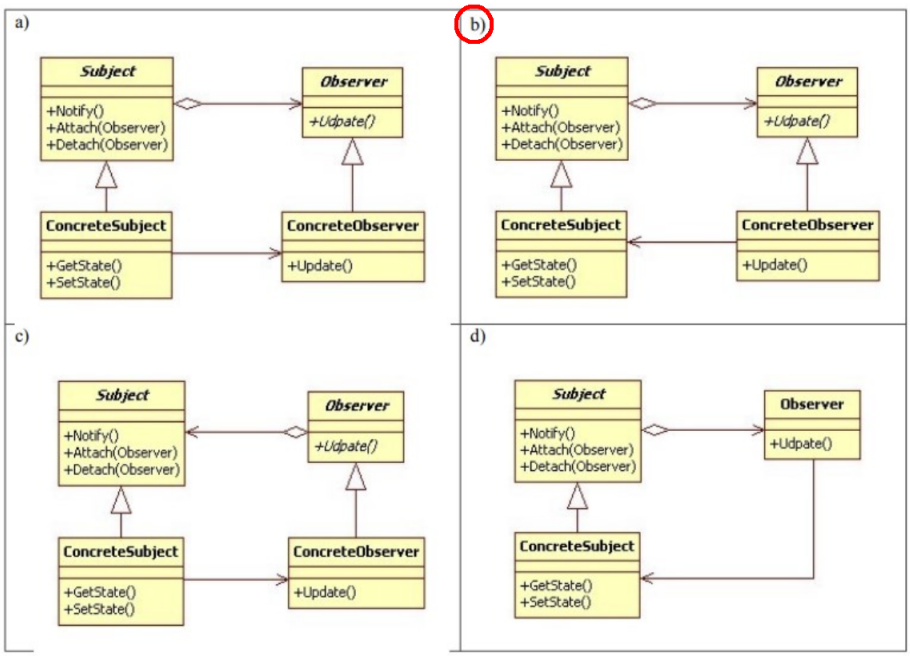
2.9 Domanda 8

Indicare con una X sulla corrispondente lettere identificativa il diagramma delle classi che, fra quelli sotto riportati, rappresenta correttamente il *design pattern* architetturale Model View Controller (MVC).



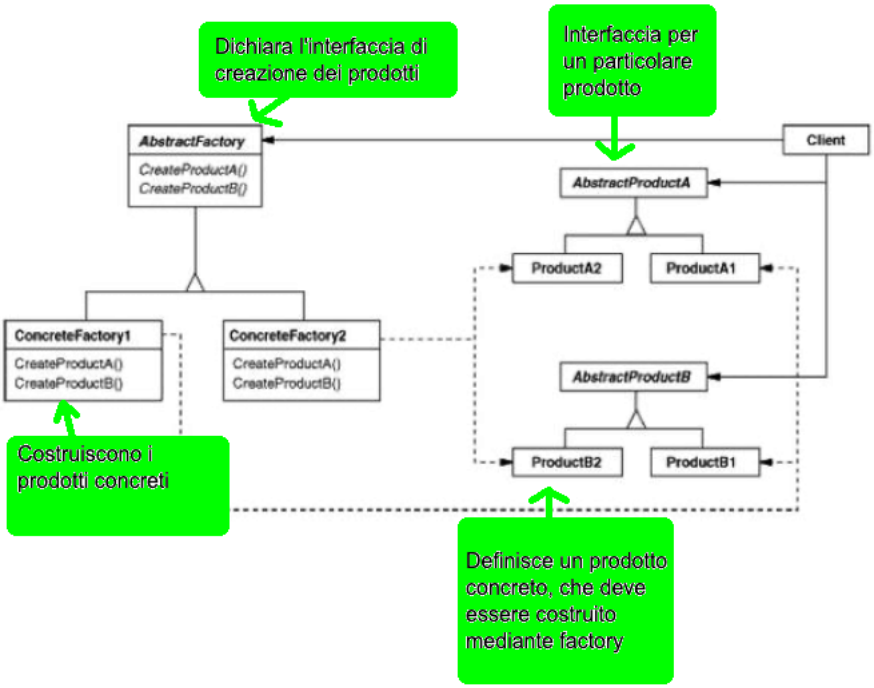
2.10 Domanda 9

Barrare con una X la lettera del diagramma delle classi che fra i seguenti rappresenta in modo corretto il *design pattern* comportamentale *Observer*.



2.11 Domanda 10

Si disegni il diagramma delle classi che modella le componenti e le relazioni che intercorrono tra di esse all'interno del design pattern *Abstract Factory* (GoF). Si ponga particolare attenzione alle operazioni definite all'interno di ogni classe.



2.12 Domanda 11

Si selezionino le affermazioni vere tra le seguenti.

a. Un'architettura a microservizi è sempre consigliata per lo sviluppo di qualsiasi tipo di applicazione.

✓b. Un microservizio può avere un'architettura logica di tipo *esagonale*.

c. E' buona norma condividere le informazioni fra microservizi utilizzando delle tabelle database comuni.

✓d. La comunicazione fra microservizi dovrebbe essere il più possibile sincrona.

✓e. Un microservizio può avere un'architettura logica di tipo *multi-layered*.

f. Le architetture a microservizi, se ben progettate, consentono uno *scaling* verticale delle componenti dell'applicazione.

2.13 Domanda 12

Si specifichi se la seguente affermazione è vera o falsa:

Utilizzando il *design pattern Abstract Factory* risulta immediato aggiungere un nuovo tipo di prodotto, mentre è complessa l'aggiunta di una nuova famiglia di prodotti.

O Vero

✓ Falso

2.14 Domanda 13

Si selezionino fra le seguenti le affermazioni vere.

✓a. Nell'architettura *multi-tier* la *business logic* è isolata dall'*application* e dalle *persistence logic*.

b. Un microservizio può essere anche un monolite.

c. Un microservizio non può essere implementato utilizzando un'architettura di tipo *multi-tier*.

✓d. Un microservizio rappresneta una unità di *deploy*.

e. Un monolite può essere implementato utilizzando un'architettura di tipo esagonale.

✓f. Esistono varie tipologie di architetture, che si differenziano sulla base di quali requisiti non funzionali intendono soddisfare (ad esempio, *deploy*, dipendenze fra le componenti, etc.)

2.15 Domanda 14

Una classe si dice coesa se ognuno dei suoi utilizzatori utilizza metodi differenti rispetto a quelli utilizzati dagli altri.

O Vero

✓ Falso

2.16 Domanda 15

Tra le attività in capo al(la) responsabile di progetto sta la redazione del "consuntivo di periodo". Tra le attività essenziali dello *sprint*, come inteso nei metodi di sviluppo agile, sta la "retrospettiva". La relazione tra tali due concetti risulta spesso oscura agli apprendisti informatici. L'elenco qui sotto riporta alcune tra le ipotesi più ricorrenti. Indicate quale tra esse sia per voi la più condivisibile.

✓a. Quando lo sviluppo è suddiviso in periodi successivi e la gestione di progetto segue prassi allo stato dell'arte, i due concetti si sovrappongono perfettamente.

b. La retrospettiva può essere prassi utile in situazioni critiche, a valle del verificarsi di qualche problema, oppure come innesco a un ciclo di auto-miglioramento. Il consuntivo di periodo, invece, è richiesto a ogni periodo. Ne segue che i due concetti sono disgiunti e non sovrapponibili.

c. Tra "consuntivo di periodo" e "retrospettiva" non vi è alcuna relazione. Il primo è un ragionamento esclusivamente contabile; il secondo è un modo per alimentare lo "spirito di gruppo".

2.17 Domanda 16

Tra aspiranti informatici sorge una discussione accesa sul rapporto tra ora di orologio e ora produttiva. L'elenco qui sotto riporta alcune tra le ipotesi più ricorrenti in tale discussione. Indicate quale tra esse sia per voi la più condivisibile.

a. Esiste solo l'ora di orologio, che è misura oggettiva e indipendente dall'individuo. L'ora produttiva è un concetto arbitrario, soggettivo e non misurabile.

b. I due concetti esistono, ma, dove il primo è oggettivo e indipendente dall'individuo, il secondo è fortemente variabile e di difficile uso. Pertanto, affidarsi al secondo è esercizio pieno di rischi, mentre usare come riferimento il primo fornisce certezze contabili.

✓c. La "taglia" di un progetto, per lunghezza temporale e costi economici, è determinata in sede di preventivo dal numero di ore produttive stimate necessarie per la realizzazione di quanto richiesto. Lo svolgimento di ogni singola attività di progetto consuma simultaneamente ore di orologio (nella misura del tempo personale) e ore produttive (in funzione del tasso di raggiungimento degli obiettivi di tale attività). La direzione del rapporto tra tali due quantità rispetto al punto di perfetto equilibrio dice la pressione sulla persona (se maggiore di 1) o il margine utile del fornite (se minore di 1).

2.18 Domanda 17

Un'altra fonte di frequente equivoco tra apprendisti informatici è l'interpretazione della relazione intercorrente tra le nozioni di progetto e di processi di ciclo di vita del *software*. L'elenco qui sotto riporta alcune tra le ipotesi più ricorrenti. Indicate quale tra esse sia per voi la più condivisibile.

✓a. Un progetto si compone di attività provenienti da vari processi, la cui specifica costituisce il *way of working* del fornitore.

b. Progetto e processi non hanno alcuna relazione tra loro: il primo (progetto) non ha bisogno dei secondi; similmente, i secondi (processi), se mai esistano, lo fanno al di fuori dei confini di un progetto.

c. L'unica relazione possibile è intorno al concetto di "sviluppo". Se esiste un processo "sviluppo", allora quel singolo processo ingloba un intero progetto, che - appunto - sviluppa un dato prodotto.

2.19 Domanda 18

Parliamo di temi e ambiti di SWE; ciò fissa il contesto della domanda e delle risposte proposte.
Tra apprendisti informatici, nascono sovente dubbi interpretativi sulla relazione intercorrente tra i concetti di *milestone* e *baseline* nel dominio IS. L'elenco qui sotto riporta alcune tra le ipotesi più ricorrenti. Indicate quale tra esse sia per voi la più condivisibile.

a. I due concetti sono del tutto disgiunti e indipendenti: la *milestone* attiene alla pianificazione di progetto, e sostanzialmente corrisponde a una data di calendario, alla quale corrispondono certe attese. La *baseline* descrive il contenuto di un *repository* soggetto a controllo di versione e di configurazione, a un certo istante temporale. In certe situazioni, il primo può riferire al secondo, ma non sempre e non necessariamente.

✓b. I due concetti sono distinti, ma strettamente correlati e complementari nel significato. La *milestone* fissa un particolare punto (data) nel calendario di progetto, al quale associa specifiche attese di avanzamento (progresso atteso). La *baseline* costituisce l'evidenza tecnica di uno specifico punto di avanzamento, costituita dalle parti che la compongono secondo determinate regole di composizione.

c. I due concetti sono sinonimi uno dell'altro. Entrambi denotano un particolare punto nel tempo di progetto, fissato dal contratto con il committente.

2.20 Domanda 19

Si selezionino fra le seguenti le affermazioni corrette relative al principio *Open-Close*.

✓a. Una componente che utilizza variabili con *scope* globale può essere chiusa al cambiamento.

b. La *Run Time Type Identification* (RTTI) è alla base del principio *Open-Close*.

✓c. Solitamente, la chiusura al cambiamento viene ottenuta attraverso l'utilizzo di astrazioni e polimorfismo.

d. Solitamente, una componente chiusa al cambiamento non è affetta da "cambiamenti a cascata".

e. Un insieme di classi può essere chiuso al 100% dei cambiamenti.

✓f. L'*encapsulation* abilita una componente ad essere chiusa al cambiamento.

2.21 Domanda 20

Una delle *best practice* nell'applicazione di un'architettura a microservizi è quella di utilizzare un database con delle tabelle condivise per lo scambio di informazioni fra diversi microservizi.

O Vero

✓ Falso