Évfolyamzárthelyi 0523

Határidő Nincs megadva határidő **Pont** 45 **Kérdések** 45
Elérhető máj 23, 09:00-ig **Időkorlát** 45 perc

Instrukciók

Az előadás ismeretanyagából az aláírás megszerzésének feltételeként a félév végén zárthelyit kell írni. A zárthelyi értékelése kétfokozatú (megfelelt / nem felelt meg), a sikeres teljesítéshez a kérdések legalább 2/3-át kell helyesen megválaszolni.

Az elméleti zárthelyin 45 feleletválasztós kérdésre kell választ adni 45 perc alatt. A zárthelyi sikeres, amennyiben legalább 30 kérdés helyesen megválaszolásra kerül.

A zárthelyit a géptermi gépeken kell teljesíteni, és semmilyen segédeszköz nem használható.

Ezt a kvízt ekkor zárolták: máj 23, 09:00.

Próbálkozások naplója

	Próbálkozás	Idő	Eredmény
LEGUTOLSÓ	1. próbálkozás	31 perc	25 az összesen elérhető 45 pontból

⚠ A helyes válaszok elérhetőek lesznek ettől eddig: máj 24, 09:00 - máj 26, 23:59 .

Ezen kvíz eredménye: **25** az összesen elérhető 45 pontból
Beadva ekkor: máj 23, 08:38
Ez a próbálkozás ennyi időt vett igénybe: 31 perc

1. kérdés — 1 / 1 pont

Mi volt az ún. szoftverkrízis kiváltó oka az 1960-as és 70-es években?

- (●) A szoftverprojektek méretben, komplexitásban, időben és a résztvevő fejlesztők számában is növekedni kezdtek.
- ( ) A hidegháborús versengés miatt a tudástranszfer akadályozása.
- ( ) A szoftverfejlesztés alulfinanszírozottsága miatti pénzügyi okok.

- A magasabb szintű programozási nyelvek megjelenése.

2. kérdés

1 / 1 pont

A szoftverfejlesztési csapatnak számos tagja lehet, akik különböző szerepeket töltenek be. Az alábbi szerepek közül melyik van **helytelenül** meghatározva?

- fejlesztő (developer): szoftver implementációja
- programgazda (program management): fejlesztés ütemezése, feladatok elosztása és követése
- minőségbiztosító (quality assurance): tesztelés tervezése, magvalósítása, minőségi kritériumok ellenőrzése
- termékgazda (product management): szoftver magas szintű tervének elkészítése

3. kérdés

1 / 1 pont

Melyik **nem** funkciója a projektmenedzsment eszközöknek?

- Programverziók és változások áttekintése.
- UML diagramok elkészítése és elhelyezése a tervben (case tooling).
- Feladatok (issue) létrehozása, célszemélyhez (assignee) rendelése.
- Hibák bejelentése, kapcsolódó információk (pl. eseménynapló) feltöltése.

4. kérdés

1 / 1 pont

Melyik lépés **nem** része a szoftver specifikációnak?

- követelmény feltárás és elemzés
- követelmény validáció
- megvalósíthatósági elemzés
- rendszer szerkezetének meghatározása

5. kérdés

1 / 1 pont

Mely álltás hamis? A követelmények feltárását nehezítheti, hogy...

- a vevők nem rendelkeznek informatikai ismeretekkel.
- a vevők nem egyértelműen fejtik ki az elvárásokat.
- a vevők a szoftver közvetlen felhasználói.
- a vevők bizonytalanok az elvárásokban.

Helytelen

6. kérdés

0 / 1 pont

Melyik nem "nem funkcionális" követelmény?

- Termék követelmények
- Menedzselési követelmények
- Külső követelmények

Szolgáltatások, reakciók leírása

Helytelen

7. kérdés

0 / 1 pont

Az alábbi SOLID elvek közül melyik van helyesen megfogalmazva?

Dependency inversion principle (DIP): függőségeket csak az absztrakciók között állítunk fel, és nem a konkrét megvalósítások között

Open/closed principle (OCP): a programegységek nyitottak a módosításra, de zártak a kiterjesztésre

Interface segregation principle (ISP): sok kisebb speciális interfész helyett kevesebb, de általános interfészt használjunk

Liskov substitution principle (LSP): az objektumok felcserélhetőek őstípusaik tetszőleges példányára a program viselkedésének befolyásolása nélkül

8. kérdés

1 / 1 pont

A modell/nézet architektúrára vonatkozóan az alábbi állítások közül melyik van **rosszul** megfogalmazva?

a nézet tartalmazza a grafikus felhasználói felület megvalósítását, beleértve a vezérlőket és eseménykezelőket

a modell függ a nézettől, egy modellt mindig egy adott felülethez készítünk el

a modell tartalmazza a háttérben futó logikát, azaz a tevékenységek végrehajtását, az állapotkezelést, valamint az adatkezelést, ezt nevezzük alkalmazáslogikának, vagy üzleti logikának

a felhasználó a nézettel kommunikál, a modell és a nézet egymással

Helytelen

9. kérdés

0 / 1 pont

Mi a *Single responsibility principle (SRP)* elv célja?

Minden metódus csak egyféle típusú kivétel kezeletlenül hagyását teheti lehetővé.

Minden osztály reprezentációját egyetlen adattagban kell definiálni, egy komplex adatstruktúra létrehozásával.

Minden komponens, osztály, metódus csak egy felelősségi körrel rendelkezzen, ami megváltoztatásának oka lehet.

A felhasználói felület minden vezérlője csak egyetlen funkcióért felelhet, szoftverergonómiai megfontolásból.

Helytelen

10. kérdés

0 / 1 pont

Melyik állítás helyes?

Az UML szekvencia diagram (sequence diagram) célja, hogy az objektumok közötti interakció lefolyását az interakciók és az interakciókban felhasznált adatok szempontjából közelítse meg

Az UML tevékenység diagram (activity diagram) célja, hogy a végrehajtás lefolyását a tevékenységek és a tevékenységekben felhasznált adatok szempontjából közelítse meg

Az UML kommunikáció diagram (communications diagram) célja az objektumok közötti kommunikációban felhasznált adatok sorrendjének megállapítása

Az UML kommunikáció diagram (communications diagram) célja, hogy az objektumok közötti kommunikáció lefolyását a kommunikációk és a kommunikációkban felhasznált adatok szempontjából közelítse meg

11. kérdés

1 / 1 pont

Melyik diagram nem jó az objektumok és osztályok közötti interakciós folyamatok modellezésére?

- kommunikációs diagram
- szekvencia diagram
- tevékenység diagram
- állapot diagram

12. kérdés

1 / 1 pont

Melyik **nem** projektvezető szolgáltatás?

- GitLab
- Azure Devops
- GitHub
- Redmine

13. kérdés

1 / 1 pont

Git verziókezelő eszköz esetén mit értünk a *staging area* alatt?

- Azokat a változtatásokat a helyi munkakönyvtárban, amelyeket még nem vontunk verziókövetés alá.
- Azokat a változtatásokat, amelyeket már verziókezelés alá vontunk, de még nem mentettük el egy új verzióba (*commit*).
- Azokat a változtatásokat, amelyeket már egy új verzióban rögzítettünk (*commit*), de nem küldtük be a távoli tárolóra (*push*).
- Azokat a változtatásokat, amelyeket tesztelési célból egy külön fejlesztési ágra küldtünk be.

Helytelen

14. kérdés

0 / 1 pont

Melyik állítás **igaz** az alábbiak közül a Git verziókövető rendszerre?

- A Git a beküldés előtti egyesítés konkurenciakezelési módszert alkalmazza (*merge before commit*).

A Git szétválasztja a verziókezelési és a hálózati műveleteket.

A `.gitignore` fájlban azt adhatjuk meg, hogy mely állományokat nem szeretnénk a távoli tárolóról letölteni.

A Git elosztott verziókövető rendszer, ezért minden kliensnél csak a verziótörténet egy része található meg.

15. kérdés

1 / 1 pont

Az alábbiak közül a git mely parancsával szinkronizálhatjuk a távoli tárolóba a lokális tárolónkban létrehozott új verziót?

git pull

git synchronize

git commit

git push

16. kérdés

1 / 1 pont

Mi a Git LFS (Large File Storage) célja és működési elve?

A nagy méretű bináris állományokat Git helyett SVN verziókezelőrendszerbe helyezi, amely alkalmas a bináris állományok hatékony verziókezelésére.

A nagy méretű bináris állományoknak csak az utolsó állapotát őrzi meg, mert ezek verziókezelése jellemzően szükségtelen.

A nagy méretű bináris állományokat a GitHub szerverén tárolja, így az nem növeli a tároló (repository) méretét.

A nagy méretű bináris állományokat egy hivatkozással helyettesíti, és magukat a fájlokat külön tárolja.

17. kérdés

1 / 1 pont

Mi a build rendszerek elsődleges célja?

A forráskód fordíthatóvá tétele continuous integration (CI) környezetben

A forráskód fordítása konzolos eszközökkel, ha integrált fejlesztőkörnyezet (IDE) nem áll rendelkezésre.

A forráskód felosztása fordítási egységekre.

A forráskód fordításának a definiált szabályok szerinti automatizálása.

18. kérdés

1 / 1 pont

Mi a .NET Standard?

A .NET Framework standard library-je.

Olyan API specifikáció, amelynek az összes .NET platform megfeleltethető.

A .NET Framework új-generációs, cross-platform futtatókörnyezete.

A .NET Framework implementációja Linux operációs rendszerre, korábbi nevén Mono Framework.

19. kérdés

1 / 1 pont

Mely feladatot **nem** látja el egy build rendszer?

- Program lefordítása
- A megváltozott projekt fájlok automatikus feltöltése a verziókezelőbe.
- Automatizált tesztek végrehajtása
- Függőségek kezelése

20. kérdés

1 / 1 pont

Mi a tesztelés helyes sorrendje?

- egységteszt, integrációs teszt, rendszerteszt, kiadásteszt, felhasználói teszt
- integrációs teszt, felhasználói teszt, kiadásteszt, egységteszt, rendszerteszt
- kiadásteszt, egységteszt, felhasználói teszt, rendszerteszt, integrációs teszt
- egységteszt, integrációs teszt, felhasználói teszt, rendszerteszt, kiadásteszt

21. kérdés

1 / 1 pont

Az alábbiak közül melyik egy Lehman törvény?

- egy szoftvert folyamatosan használni kell, hogy folyamatosan nőjön a használhatósága és minősége
- egy szoftvernek változnia kell, hogy folyamatosan csökkenjen a használhatósága és minősége
- **egy szoftvernek változnia kell, vagy különben folyamatosan csökken a használhatósága és minősége**
- egy szoftvert folyamatosan használni kell, vagy különben folyamatosan csökken a használhatósága és minősége

22. kérdés

1 / 1 pont

Mely állítás igaz?

- A kiadás tesztet a fejlesztő csapat végzi.
- **A felhasználói teszt jellemzően fekete doboz tesztekből áll.**
- A fejlesztői teszt jellemzően fekete doboz tesztekből áll.
- A füst tesztet a tápegységből felszálló füst mennyiségének mérésével végzik.

23. kérdés

1 / 1 pont

Melyik állítás **nem** igaz a *container framework*-ökre (pl. Docker)?

A containerekben futó alkalmazások belső hálózati kapcsolaton kommunikálhatnak egymással.

Minden container osztozik a gazda számítógép hardveres erőforrásain.

A containerek saját, elkülönített, virtualizált környezetben futnak.

A containerek futó alkalmazások annak saját virtuális operációs rendszerén (pl. Docker OS) futnak.

24. kérdés

1 / 1 pont

Mi a célja a folyamatos integrációs (continuous integration, CI) gyakorlati módszernek?

Objektum orientált programozási nyelvre való átállást segíti elő.

A lehetséges hibák, integrációs problémák azonnali, automatizált kiszűrése, visszajelzés a fejlesztőnek.

Az elbukott integrációs tesztek automatikus újra futtatása, ameddig meg nem javulnak.

A manuális tesztelés teljes kiváltása.

Helytelen

25. kérdés

0 / 1 pont

Mi a folyamatos teljesítés (continuous delivery) célja?

A programkódok egy központi tárhelyre küldésre, verziókezelő rendszer segítségével, naponta többször.

Az önszerveződő, kis csapatok folytonos interakciójának biztosítása gyors visszajelzésekkel.

A folyamatos kiadások automatizálása.

A gyors alkalmazásfejlesztés megvalósítása, inkrementális alapon.

Helytelen

26. kérdés

0 / 1 pont

Milyen célt szolgál a GitLab CI cache konfigurációjának kulcsa (key)?

Használatával a cache tartalma artifact-ként letölthetővé tehető.

Használatával korlátozható, hogy mely jobok férhetnek hozzá a cache-hez.

Használatával különálló cache használható akár jobonként vagy fejlesztési áganként.

Használatával megadható a Cache Server elérhetősége, amennyiben nem az alapértelmezettet kívánjuk használni.

Helytelen

27. kérdés

0 / 1 pont

Mely tulajdonságok jellemzőek a Clean Code-ra?

Olvasható, tömör, öndokumentáló

- Olvasható, karbantartható, tesztelhető, elegáns
- Jól dokumentált, tesztelt, elegáns
- Könnyen olvasható, nem tartalmaz kódismétlést, tesztelhető

28. kérdés

1 / 1 pont

Melyik koncepció része a Clean Code-nak?

- Használjunk prefixeket az elnevezéseknél
- Ugyanazt a nevet ne használjuk különböző célra
- A break és continue utasításokat elővigyázatosan kell alkalmaznunk.
- Rövidítsük mindig a változó neveket

Helytelen

29. kérdés

0 / 1 pont

Mit mond ki a DRY elv?

- Az biztosan elmondható, hogy javulni fog a kódbázisunk minősége, ha mindig úgy hagyjuk ott az aktuális kódunkat, hogy az egy kicsit „jobb”, egy kicsit tisztább annál, mint ahogy megtaláltuk.
- A tökéletességet nem akkor lehet a legjobban megközelíteni, ha egy rendszerhez nem tudunk már semmit hozzáadni, hanem akkor, ha nem tudunk mit elvenni belőle.
-

Ne implementáljunk előre olyan kódot, ami „majd a jövőben kelleni fog”, mert szinte biztos, hogy sose lesz rá szükségünk.

A tudás minden darabkájának egyetlen, egyértelmű és megbízható reprezentációval kell rendelkeznie egy rendszeren belül.

Helytelen

30. kérdés

0 / 1 pont

Az alábbi, alkalmazások architektúrájára vonatkozó állítások közül melyik **hamis**?

A befecskendezésnek különböző módjai lehetnek (például: konstruktor, metódus).

Az egyes rétegek között függőségek alakulnak ki, mivel felhasználják egymás funkcionalitását.

A függőségeket úgy kell megvalósítani, hogy a konkrét megvalósítástól függjenek.

A függőség befecskendezés (*dependency injection*) jelentése, hogy a rétegek a függőségeknek csak az absztrakcióját látják, a konkrét megvalósítását külön adjuk át nekik.

31. kérdés

1 / 1 pont

Melyik tervezési minta megvalósításának része lehet az alábbi kódrészlet?

```
public MyPattern withName(string name) {
    this.name = name;
```

```
    return this;
}

public MyPattern withNumber(int number) {
    this.number = number;
    return this;
}
```

- Command (Parancs)
- Builder (Építő)
- Adapter (Illesztő)
- Singleton (Egyke)

Helytelen

32. kérdés

0 / 1 pont

Melyik tervezési mintát alkalmazhatjuk abban az esetben, ha konkrét osztály megadása nélkül szeretnénk kapcsolódó vagy egymástól függő objektumok családjának létrehozására felületet biztosítani?

- Factory method (Gyártó függvény)
- Builder (Építő)
- Adapter (Illesztő)
- Abstract Factory (Absztrakt gyár)

33. kérdés

1 / 1 pont

Melyik tervezési mintát alkalmazhatjuk abban az esetben, ha egy adott osztály példányosítását szeretnénk a hozzátartozó alosztályokra átruházni?

- Builder (Építő)

- Command (Parancs)
- Observer (Megfigyelő)
- Factory method (Gyártó függvény)

Helytelen

34. kérdés

0 / 1 pont

Mely tervminta tudja csökkenteni az objektumok közötti függőségeket?

- Híd (Bridge)
- Közvetítő (Mediator)
- Illesztő (Adapter)
- Gyártó művelet (Factory method)

35. kérdés

1 / 1 pont

Melyik tervezési minta nyújt megoldást arra a problémára, ha több objektumot szeretnénk értesíteni, amikor egy másik objektumnak megváltozik az állapota?

- Observer (Megfigyelő)
- Factory (Gyártó)
- Singleton (Egyke)
- Adapter (Illesztő)

Helytelen

36. kérdés

0 / 1 pont

Melyik tervezési minta alkalmazható a hosszú paraméterlistájú konstruktorok elkerülésére?

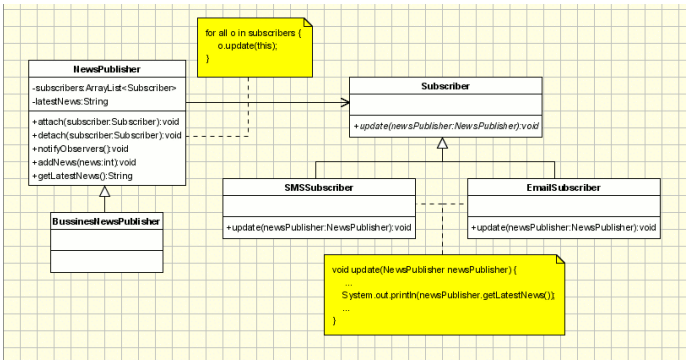
- Factory (Gyártó)
- Builder (Építő)
- Observer (Megfigyelő)
- Command (Parancs)

Helytelen

37. kérdés

0 / 1 pont

Melyik tervezési minta alkalmazása látszik a képen?



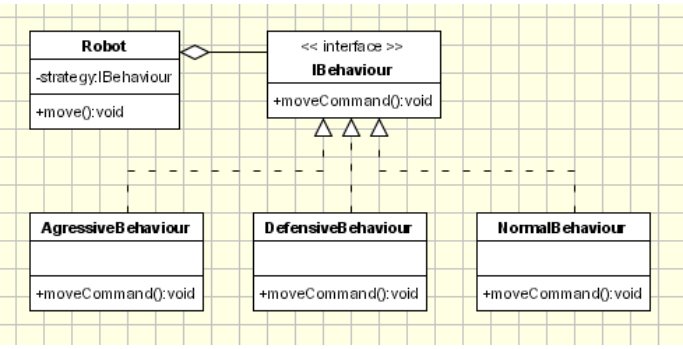
- Object Pool (Objektumkészlet)
- Abstract Factory (Elvont gyár)
- Observer (Megfigyelő)
- Facade (Homlokzat)

Helytelen

38. kérdés

0 / 1 pont

Melyik tervezési minta alkalmazása látszik a képen?



- [x] Decorator (Díszítő)
- [ ] Command (Parancs)
- [ ] Template method (Sablonfüggvény)
- [ ] Strategy (Stratégia)

Helytelen

39. kérdés

0 / 1 pont

Mi a különbség a folyamat (*process*) és a szál (*thread*) között?

- [ ] A folyamatoknak saját végrehajtási környezetük (pl. memóriaterület) van, a szálak osztozkodnak ezen.
- [ ] A folyamatokat Linux operációs rendszeren szálaknak hívjuk.
- [ ] Nincs különbség, a kettő egymás szinonimája.
- [x] Egy szál több folyamatot is tartalmazhat.

Helytelen

40. kérdés

0 / 1 pont

Mi a kiéheztetés (*starvation*)?

Olyan jogosultságkezelési hiba, amely során egy szál nem tudja a szükséges erőforrásokat (pl. fájlokat) megnyitni.

Olyan logikai hiba, amelynek során egy közös erőforráshoz több szál is egyszerre hozzáférhet, ezzel inkonzisztens állapotba juttatva a programot.

Olyan ütemezési hiba, amely során egy erőforrásra több szál is várakozik és egyikük sem jut soha hozzá, így "lefagy" a program.

Olyan prioritási hiba, amely esetén a kis prioritású vagy nagy erőforrás igényű folyamatok sosem képesek lefutni.

41. kérdés

1 / 1 pont

Melyik állítás igaz a kölcsönös kizárásra (*mutual exclusion*)?

Nincsen olyan többszálú program, amely kölcsönös kizárás nélkül helyesen tud működni,

A kölcsönös kizárás garantálja, hogy a közös erőforráshoz egyszerre csak egy szál férhessen hozzá, kizárva ezzel a versenyhelyzetet (*race condition*).

A kölcsönös kizárás célja, hogy a többszálú program egyszerre mindig csak egy szál futhasson.

A kölcsönös kizárás célja a szálak szinkronizációja: a kritikus szakasz mindig ugyanazon a szálon fusson le.

Helytelen

42. kérdés

0 / 1 pont

Melyik **nem** tartozik az előadáson felsorolt SCRUM folyamat elemek közé?

- bemutató
- visszatekintés
- futam
- bemutató tervezés

43. kérdés

1 / 1 pont

Mi a *planning poker*?

- projektmenedzsment becslési módszere
- szoftver tervezésének becslési módszere
- kártyajáték
- szerencsejáték

Helytelen

44. kérdés

0 / 1 pont

A három Scrum-termék (artifacts) a következő:

- termék kívánságlista (product backlog), futam teendőlista (sprint backlog), Scrum tábla (Scrum table)
-

termékvízió (product vision), termék kívánságlista (product backlog), felhasználói történet (user story)

termék kívánságlista (product backlog), Scrum tábla (Scrum table), haladási diagram (progress diagram)

termék kívánságlista (product backlog), futam teendőlista (sprint backlog), inkrementum (increment)

Helytelen

45. kérdés

0 / 1 pont

Melyik nem agilis szoftverfejlesztési módszertan szerinti modell az alábbiak közül?

- Rational Unified Process (RUP)
- Lean
- Scrum
- Kanban

Kvízeredmény: **25** az összesen elérhető 45 pontból