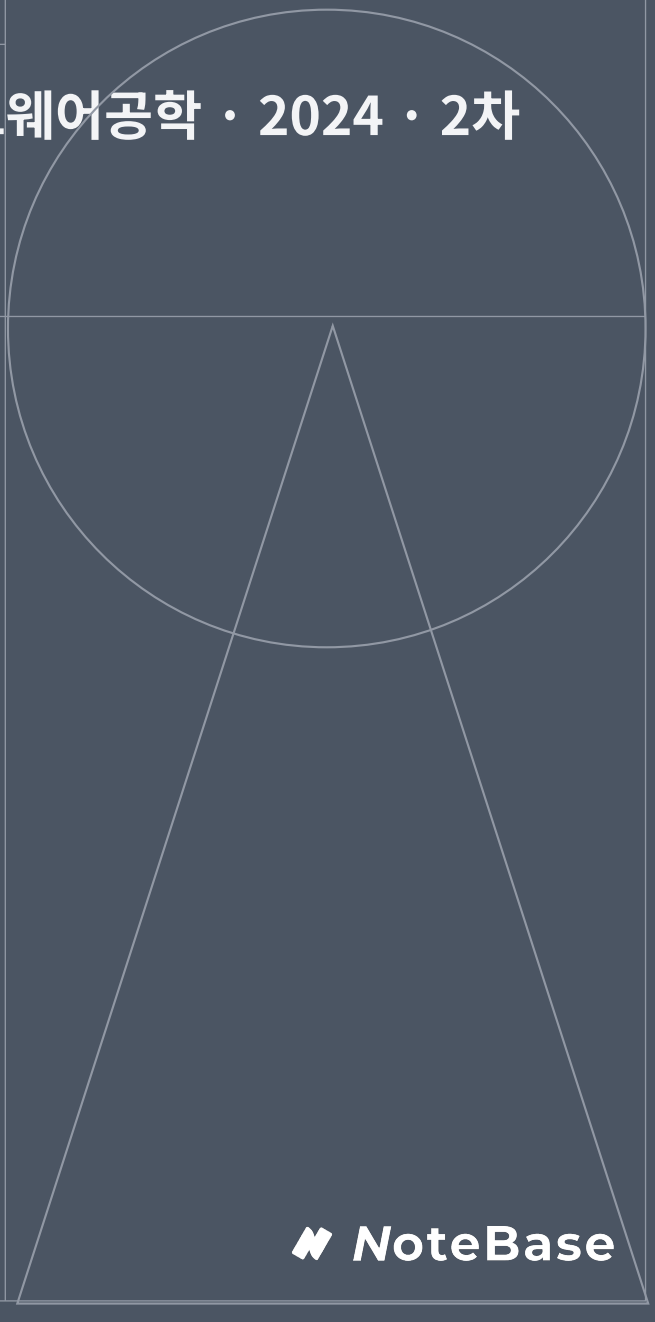
기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2024 · 2차



NoteBase

기출OX 해설집

공무원/국가직/7급 · 소프트웨어공학 · 2024 · 2차

공무원/국가직/7급 · 소프트웨어공학 · 2024 · 2차

문 1. 소프트웨어의 대표적 특징으로 옳지 않은 것은?

① 순응성

② 복잡성

③ 마모성

④ 비가시성

정답 3번

출제 포인트

이 문제는 소프트웨어의 대표적 특성 중 비마모성과 마모성을 구별할 수 있는지를 묻는 문제이다.

핵심 개념

순응성: 소프트웨어가 외부 업무 규칙, 제도, 인터페이스 요구에 맞추어져야 하는 성질이다.
복잡성: 구성 요소와 상호작용이 많아 구조가 쉽게 단순화되지 않는 성질이다.
비가시성: 물리적 형체가 없어 전체 구조를 눈으로 직관하기 어려운 성질이다.
비마모성: 소프트웨어는 기계 부품처럼 사용 자체로 닳거나 마모되지 않는 성질을 가진다.

NoteBase

선지별 해설

① O
순응성은 소프트웨어가 자연법칙보다 조직의 규칙, 사용자 요구, 외부 시스템 규약에 맞추어야 하는 성질을 뜻하므로 대표적 특성에 해당한다.

② O
복잡성은 기능과 논리 관계가 많아질수록 구조와 상호작용이 급격히 복잡해지는 성질이므로 대표적 특성에 해당한다.

③ X
틀린 부분은 '마모성'이다. 소프트웨어는 사용한다고 닳는 물리적 대상이 아니므로 올바르게는 '비마모성' 또는 '사용으로 마모되지 않음'이다. 소프트웨어의 품질 저하는 마모가 아니라 변경 누적이나 유지보수 과정의 오류에서 주로 발생한다.

④ O
비가시성은 소프트웨어가 눈에 보이는 입체적 형체를 갖지 않아 전체 구조를 한눈에 파악하기 어려운 성질을 말하므로 대표적 특성에 해당한다.

오답 포인트

③에서 하드웨어의 물리적 특성인 마모성을 소프트웨어에도 그대로 적용하면 틀리기 쉽다. 소프트웨어 결함 증가를 사용에 따른 마모로 오해하는 함정을 노린 문항이다.

다시 나오면 이렇게 판단

소프트웨어 특성 문제에서는 무형성에서 파생되는 성질인지 먼저 본다. 복잡성, 순응성, 비가시성은 해당하지만 사용으로 닳는다는 표현이 나오면 비마모성과 대비해 판단하면 된다.

암기 포인트

소프트웨어는 복잡하고 보이지 않으며 환경에 순응하지만 사용으로 닳지는 않는다.

NoteBase

문 2. 요구 분석에 필요한 작업의 단계를 순서대로 바르게 나열한 것은?

(가) 요구사항 분석
(나) 요구사항 검증
(다) 요구사항 추출
(라) 요구사항 명세

① (다)→(가)→(라)→(나)
② (다)→(라)→(나)→(가)
③ (라)→(가)→(다)→(나)
④ (라)→(다)→(가)→(나)

정답 1번

출제 포인트

이 문제는 요구 분석에서 요구사항을 어떤 흐름으로 다루는지, 즉 요구사항 추출 후 분석하고 그 결과를 명세한 뒤 최종적으로 검증하는 절차의 선후관계를 묻는 문제이다.

올바른 순서

1. 요구사항 추출 - 사용자와 이해관계자로부터 필요한 요구를 수집하는 단계이다.
2. 요구사항 분석 - 수집된 요구의 중복, 충돌, 모호성을 정리하고 구조화하는 단계이다.
3. 요구사항 명세 - 분석된 요구를 문서나 모델로 명확하게 기술하는 단계이다.
4. 요구사항 검증 - 명세된 요구가 완전성, 일관성, 타당성을 갖추었는지 확인하는 단계이다.

왜 이 순서: 분석은 추출된 요구가 있어야 가능하고, 명세는 분석 결과를 문서화하는 단계이며, 검증은 작성된 명세를 대상으로 수행되므로 추출 → 분석 → 명세 → 검증의 순서가 된다.

선지별 해설

① O - (다) 요구사항 추출 → (가) 요구사항 분석 → (라) 요구사항 명세 → (나) 요구사항 검증의 흐름이 표준 절차와 일치한다. 수집한 요구를 먼저 분석해 정리한 뒤 문서화하고, 마지막에 그 문서가 적절한지 검증한다.
② X - 틀린 부분은 요구사항 명세를 요구사항 분석보다 앞세운 점과 검증을 분석보다 먼저 둔 점이다. 명세는 분석이 끝난 요구를 기술하는 단계이고, 검증은 명세 결과를 확인하는 단계이므로 올바르게는 (다) → (가) → (라) → (나)이다.
③ X - 틀린 부분은 요구사항 명세를 가장 먼저 둔 점이다. 명세는 추출과 분석을 거친 뒤 작성하는 산출물이므로 출발 단계가 될 수 없다. 올바르게는 (다) → (가) → (라) → (나)이다.
④ X - 틀린 부분은 요구사항 명세를 요구사항 추출보다 먼저 둔 점이다. 무엇을 문서화할지 정해지기 전에 명세를 작성할 수 없으므로 요구사항 추출이 선행되어야 한다. 올바르게는 (다) → (가) → (라) → (나)이다.

NoteBase

오답 포인트

요구사항 분석과 요구사항 명세를 비슷한 단계로 보고 순서를 바꾸는 경우가 많다. 그러나 분석은 요구를 해석하고 정제하는 단계이고, 명세는 그 결과를 문서로 확정하는 단계이므로 분석이 먼저이다. 또한 검증은 완성된 명세를 대상으로 하므로 중간에 들어갈 수 없다.

다시 나오면 이렇게 판단

요구공학 절차 문제는 먼저 수집하는 단계가 무엇인지 찾고, 그다음 정제·해석 단계인지, 문서화 단계인지, 최종 확인 단계인지로 나누어 판단하면 된다. 수집된 입력이 있어야 분석이 가능하고, 분석 결과가 있어야 명세가 가능하며, 명세가 있어야 검증이 가능하다는 의존관계를 따라가면 된다.

암기 포인트

요구공학 기본 흐름은 추출 → 분석 → 명세 → 검증이다.

NoteBase

문 3. 소프트웨어 설계 시, 소프트웨어를 기능 단위로 분해하여 모듈 간의 결합도(coupling)를 최소화하고, 모듈의 독립성을 높이기 위한 설계 원칙은?

① 다형성

② 모듈화

③ 추상화

④ 연산자 오버로딩

정답 2번

출제 포인트

이 문제는 결합도를 낮추고 모듈의 독립성을 높이는 설계 원리가 무엇인지를 묻는 문제이다.

핵심 개념

모듈화는 소프트웨어를 기능 단위의 모듈로 나누어 모듈 간 의존을 줄이고 독립성을 높이는 설계 원리이다.
다형성은 같은 인터페이스나 메시지에 대해 객체마다 다른 동작이 가능하게 하는 객체지향 특성이다.
추상화는 복잡한 대상에서 핵심 속성만 뽑아 공통 개념으로 표현하는 원리이다.
연산자 오버로딩은 같은 연산자 기호를 자료형이나 문맥에 따라 다르게 동작하게 하는 기능이다.

선지별 해설

① X: 다형성은 객체지향에서 동일한 메시지에 대해 다른 구현이 수행되는 성질이므로 모듈 간 결합도 최소화와 직접 대응하는 설계 원리가 아니다. 틀린 부분은 '기능 단위 분해와 모듈 독립성 향상'을 다형성으로 본 점이다. 올바르게는 이러한 설명은 모듈화에 해당한다.
② O: 모듈화는 시스템을 기능별 모듈로 분해하여 각 모듈의 응집도를 높이고 모듈 간 결합도를 낮추는 원리이므로 문제의 설명과 정확히 일치한다.
③ X: 추상화는 핵심 개념만 남기고 세부를 감추는 원리이므로 모듈 간 결합도 최소화를 직접 설명하는 용어가 아니다. 틀린 부분은 '모듈 독립성 향상 설계 원리'를 추상화로 본 점이다. 올바르게는 추상화는 표현의 단순화와 본질의 강조에 관한 개념이다.
④ X: 연산자 오버로딩은 같은 연산자를 여러 방식으로 사용할 수 있게 하는 프로그래밍 기법이므로 시스템을 기능 단위로 분해하는 설계 원리와 무관하다. 틀린 부분은 '모듈 분해 원리'를 연산자 오버로딩으로 본 점이다. 올바르게는 연산자 오버로딩은 연산자 사용 방식의 확장이다.

오답 포인트

다형성, 추상화, 모듈화는 모두 소프트웨어 설계에서 자주 나오는 용어라서 헷갈리기 쉽다. 이 문항은 '기능 단위로 분해', '결합도 최소화', '독립성 향상'이라는 표현을 보고 모듈화를 바로 떠올릴 수 있는지를 묻는다.

NoteBase

다시 나오면 이렇게 판단

선지에 기능 분해, 결합도 감소, 응집도 향상, 모듈 독립성 같은 표현이 나오면 모듈화 개념을 먼저 연결하면 된다. 반대로 같은 메시지의 다른 구현은 다형성, 핵심만 남기는 것은 추상화로 구분된다.

암기 포인트

모듈화는 분해와 독립성, 다형성은 같은 호출의 다른 동작, 추상화는 핵심만 남김이다.

NoteBase

문 4. 요구사항의 특성에 대한 설명으로 옳지 않은 것은?

① 신뢰성, 성능, 안전성은 기능적 요구사항에 해당한다.

② 비기능적 요구사항인 MTBF는 (MTTF + MTTR)로 나타낸다.

③ Windows에서 C++ 언어로 개발하라는 것은 비기능적 요구사항으로 분류된다.

④ 비기능적 요구사항인 가용성을 높이기 위해서는, MTTF를 증가시키고 MTTR을 감소시켜야 한다.

정답 1번

출제 포인트

이 문제는 기능적 요구사항과 비기능적 요구사항의 구분, 그리고 신뢰도 지표인 MTBF, MTTF, MTTR의 의미를 함께 묻는 문제이다.

핵심 개념

기능적 요구사항은 시스템이 무엇을 해야 하는지를 정의한 요구사항이다.
비기능적 요구사항은 성능, 신뢰성, 가용성, 안전성처럼 시스템이 어느 수준의 품질로 동작해야 하는지를 정의한 요구사항이다.
MTTF는 고장 발생 전까지의 평균 동작 시간이고, MTTR은 고장 후 복구까지의 평균 시간이며, MTBF는 두 값을 합한 평균 고장 간격이다.

선지별 해설

① X
신뢰성, 성능, 안전성은 시스템의 품질 속성을 나타내므로 기능적 요구사항이 아니라 비기능적 요구사항이다.
틀린 표현은 "기능적 요구사항"이다. 올바르게는 "신뢰성, 성능, 안전성은 비기능적 요구사항에 해당한다"이다.

② O
수리 가능한 시스템에서 MTBF는 한 번 고장 수리 후 다음 고장까지의 평균 간격이므로 MTTF와 MTTR의 합으로 나타낸다.

③ O
Windows에서 C++ 언어로 개발하라는 내용은 구현 환경과 개발 제약을 정한 것이므로 기능 자체가 아니라 비기능적 요구사항으로 분류된다.

④ O
가용성은 보통 $\frac{MTTF}{MTTF + MTTR}$로 이해하므로 고장 없이 동작하는 시간인 MTTF는 커질수록 좋고, 복구 시간인 MTTR은 작아질수록 좋다.

NoteBase

오답 포인트

①은 요구사항의 대상이 아니라 요구사항의 종류를 뒤바꾼 선지이다. 수험생이 "성능"이나 "안전성"을 시스템 기능처럼 받아들이면 기능적 요구사항으로 착각하기 쉽다. 이 문항은 품질 속성은 비기능이라는 기본 분류를 흔드는 방식으로 함정을 만든 것이다.

다시 나오면 이렇게 판단

선지에 서비스, 처리 절차, 입력·출력처럼 시스템이 해야 할 일이 나오면 기능적 요구사항으로 본다. 반대로 성능, 신뢰성, 가용성, 보안, 안전성, 개발 환경 제약처럼 품질이나 제약 조건이 나오면 비기능적 요구사항으로 판단한다.

암기 포인트

무엇을 한다는 기능이고, 얼마나 잘해야 하는지는 비기능이다. MTBF는 MTTF와 MTTR의 합이다.

NoteBase

문 5. 소프트웨어 개발 프로세스 모델에 대한 용어와 설명을 바르게 연결한 것은?

(가) 나선형 모델 (spiral model)
(나) 통합 프로세스 모델 (unified process model)
(다) V 모델 (V model)
A. 테스트 단계가 분석 및 설계와 어떻게 관련되어 있는지 나타내고, 각 개발 단계를 검증하는 데 초점을 둔다.
B. 위험분석 단계가 존재해 처음부터 위험에 대해 고려하면서 개발을 진행한다.
C. 개발과정은 크게 4단계(도입, 정련, 구축, 전이)로 나뉘고 각 단계도 여러 개의 작은 단위로 나뉘며, 한 번 반복이 끝나면 실행 가능한 산출물이 도출된다.

① (가): A, (나): B, (다): C
② (가): A, (나): C, (다): B
③ (가): B, (나): C, (다): A
④ (가): C, (나): A, (다): B

정답 3번

출제 포인트

소프트웨어 개발 프로세스 모델의 대표 특징을 모델명과 연결하는 문제이다.

핵심 정리

나선형 모델: 위험분석을 중심에 두는 반복적 개발 모델이다. 개발 초기부터 위험을 식별하고 대응 방안을 검토하면서 진행한다.
통합 프로세스 모델: 도입, 정련, 구축, 전이의 4단계로 구성되는 반복·점증형 프로세스이다. 각 반복이 끝나면 실행 가능한 산출물이 나온다.
V 모델: 개발 단계와 이에 대응하는 테스트 단계를 V자 형태로 연결한 모델이다. 각 개발 단계의 검증과 확인을 강조한다.

NoteBase

선지별 해설

① X. (가)를 A로 연결한 점이 틀렸다. A는 테스트 단계와 분석·설계의 대응 관계를 나타내는 V 모델의 설명이다. (가) 나선형 모델은 올바르게는 B와 연결된다.
② X. (가)를 A로 연결한 점이 틀렸고, (다)를 B로 연결한 점도 틀렸다. B는 위험분석을 핵심으로 하는 나선형 모델의 설명이다. (다) V 모델은 올바르게는 A와 연결된다.
③ O. (가)=B, (나)=C, (다)=A로 모두 맞다. 나선형 모델은 위험분석 중심이고, 통합 프로세스 모델은 4단계 반복·점증 개발이며, V 모델은 개발 단계와 테스트 단계의 대응 관계가 핵심이다.
④ X. (가)를 C로 연결한 점이 틀렸다. C는 도입·정련·구축·전이의 4단계로 이루어지는 통합 프로세스 모델의 설명이다. (가) 나선형 모델은 올바르게는 B와 연결되고, (나) 통합 프로세스 모델은 올바르게는 C와 연결된다.

오답 포인트

나선형 모델과 통합 프로세스 모델은 모두 반복적으로 개발한다는 공통점이 있어 혼동하기 쉽다. 그러나 나선형 모델의 핵심 단어는 위험분석이고, 통합 프로세스 모델의 핵심 단어는 도입·정련·구축·전이의 4단계라는 점에서 구별된다.

다시 나오면 이렇게 판단

모델 이름을 보면 먼저 대표 키워드를 대응시켜 판단한다. 위험분석이 나오면 나선형 모델,
도입·정련·구축·전이가 나오면 통합 프로세스 모델, 개발 단계와 테스트 단계의 대응 관계가 나오면 V 모델로 연결하면 된다.

암기 포인트

나선형=위험분석, 통합 프로세스=도입·정련·구축·전이, V 모델=개발 단계와 테스트 단계 대응이다.

NoteBase

문 6. 특정 회사에서 공공 소프트웨어 개발 프로젝트를 수행하는 데 다음과 같은 조건이 적용될 때, 계약 기간 내에 프로젝트를 성공적으로 수행하기 위해서 추가로 투입될 최소 개발 인원은?

계약 기간: 10개월
추정 소스코드 라인 수: 100,000 LOC
개발자 1인당 월평균 생산성: 1,000 LOC/MM (Man Month)
현재 개발자 수: 7명

① 3명
② 4명
③ 5명
④ 6명

정답 1번

출제 포인트

이 문제는 총 개발량과 1인당 월평균 생산성을 이용해 총 투입 공수와 계약 기간 내 필요한 인원을 계산하는 문제이다.

조건 정리

주어진 값은 추정 소스코드 100,000 LOC, 개발자 1인당 생산성 1,000 LOC/MM, 계약 기간 10개월, 현재 개발자 수 7명이다. 구해야 할 값은 계약 기간 내 완료를 위한 추가 개발 인원이다.

사용할 공식

총 필요 공수는 총 LOC ÷ 1인당 생산성이다. 필요한 총 인원은 총 필요 공수 ÷ 계약 기간이다. 추가 인원은 필요한 총 인원 - 현재 인원이다.

계산 과정

100,000 LOC ÷ 1,000 LOC/MM = 100 MM이다. 100 MM ÷ 10개월 = 10명이다. 현재 7명이므로 10명 - 7명 = 3명을 추가로 투입해야 한다.

1번 선지

O. 계산 결과 계약 기간 10개월 안에 필요한 총 인원은 10명이고 현재 인원은 7명이므로 추가 인원은 3명이다.

NoteBase

2번 선지

X. 4명을 추가하면 총 11명이 되어 필요 총 인원 10명을 초과한다. 틀린 표현은 '추가 4명'이며, 올바르게는 '추가 3명'이다.

3번 선지

X. 5명을 추가하면 총 12명이 되어 계산 결과와 맞지 않는다. 틀린 표현은 '추가 5명'이며, 올바르게는 '추가 3명'이다.

4번 선지

X. 6명을 추가하면 총 13명이 되어 계약 기간 10개월 기준 필요 인원보다 과다하다. 틀린 표현은 '추가 6명'이며, 올바르게는 '추가 3명'이다.

오답 포인트

총 필요 공수 100 MM을 먼저 구하지 않고 곧바로 현재 인원을 더하거나 빼면 오답이 된다. 필요한 총 인원 10명과 추가 인원 3명을 구분하지 못하면 4명 이상을 고르기 쉽다.

다시 나오면 이렇게 판단

계산형 인력 산정 문제는 총 작업량을 MM으로 바꾸고, 이를 계약 기간으로 나누어 필요한 총 인원을 구한 뒤, 마지막에 현재 인원을 빼서 추가 인원을 계산한다.

암기 포인트

총 LOC ÷ 생산성 = 총 MM, 총 MM ÷ 기간 = 필요 인원, 필요 인원 - 현재 인원 = 추가 인원이다.

NoteBase

문 7. 다음 설명에 해당하는 용어는?

◯ 테스트 실행 결과를 검증하기 위한 메커니즘
◯ 테스트 수행 결과가 참인지 거짓인지를 판단하기 위해 미리 정의된 참값과 비교하는 테스트 기법

① 테스트 케이스 (test case)
② 테스트 오라클 (test oracle)
③ 테스트 하니스 (test harness)
④ 테스트 드라이버 (test driver)

정답 2번

출제 포인트

이 문제는 테스트 결과의 옳고 그름을 판정하는 장치인 테스트 오라클의 정의를 묻는 문제이다.

핵심 개념

테스트 케이스는 입력값, 수행 조건, 기대 결과를 미리 정리한 시험 항목이다.
테스트 오라클은 실제 실행 결과가 기대 결과와 일치하는지 판정하는 기준 또는 메커니즘이다.
테스트 하니스는 테스트 실행을 자동화하고 드라이버, 스텁, 로그 수집 등을 묶어 주는 실행 환경이다.
테스트 드라이버는 시험 대상 모듈을 호출하여 입력을 전달하는 호출 프로그램이다.

선지별 해설

① X: 테스트 케이스는 무엇을 시험할지 정리한 입력·조건·기대 결과의 묶음이지, 실행 결과를 판정하는 메커니즘은 아니다. 틀린 표현은 '결과를 검증하기 위한 메커니즘'이다. 올바르게는 '테스트할 데이터와 기대 결과를 정의한 항목'이다.
② O: 테스트 오라클은 테스트 결과가 맞는지 틀리는지 판단하는 기준이다. 실제 결과를 미리 정의한 기대값이나 참값과 비교해 합격·실패를 판정한다는 설명과 일치한다.
③ X: 테스트 하니스는 테스트를 수행하기 위한 지원 환경과 도구 집합이다. 틀린 표현은 '참인지 거짓인지 판단하기 위해 비교하는 테스트 기법'이다. 올바르게는 '테스트 실행을 제어하고 결과 수집을 돕는 환경'이다.
④ X: 테스트 드라이버는 하위 모듈이나 컴포넌트를 호출하는 임시 호출 프로그램이다. 틀린 표현은 '실행 결과를 검증하기 위한 메커니즘'이다. 올바르게는 '테스트 대상 모듈을 호출해 입력을 전달하는 프로그램'이다.

오답 포인트

①과 ③, ④는 모두 테스트에 필요한 구성요소여서 기능이 비슷해 보이지만, 판정 기능이 있는지는 별개이다. 이 문항의 핵심 함정은 '테스트를 수행하는 도구'와 '결과의 정오를 판단하는 기준'을 섞어 놓은 데 있다.

NoteBase

다시 나오면 이렇게 판단

선지에 '기대 결과와 비교한다', '정답 여부를 판정한다', '참값으로 검증한다'라는 표현이 나오면 오라클을 먼저 떠올리면 된다. 반대로 실행 환경이면 하니스, 호출 프로그램이면 드라이버, 시험 항목 정의면 테스트 케이스로 구분된다.

암기 포인트

케이스는 시험 항목, 오라클은 판정 기준, 하니스는 실행 환경, 드라이버는 호출 프로그램이다.

NoteBase

문 8. 익스트림 프로그래밍(XP, eXtreme Programming) 특징에 대한 설명으로 옳지 않은 것은?

① 테스트 우선(test-first) 개발방법이다.

② 개발자는 짝(pair)으로 작업하지 않는다.

③ 빈번한 배포(release)를 통해 결과물을 평가받는다.

④ 요구사항은 사용자 스토리(user story)로 표현된다.

정답 2번

출제 포인트

이 문제는 익스트림 프로그래밍(XP)의 대표 실천 방법과 애자일 요구사항 표현 방식을 구별할 수 있는지를 묻는다.

핵심 개념

XP는 변화 대응과 빠른 피드백을 중시하는 애자일 개발 방법이다.
핵심 실천에는 테스트 우선 개발, 페어 프로그래밍, 지속적 통합, 작은 릴리스가 포함된다.
요구사항은 길고 고정된 명세서보다 사용자 스토리 형태로 가볍게 표현하는 특징이 있다.

선지별 해설

1번 O: XP의 대표 실천 중 하나가 테스트 우선 개발이다. 테스트를 먼저 작성하고 그 테스트를 통과하도록 코드를 구현하는 방식이므로 옳다.
2번 X: 틀린 표현은 '짝(pair)으로 작업하지 않는다'이다. XP는 오히려 두 개발자가 한 작업 단위를 함께 수행하는 페어 프로그래밍을 핵심 실천으로 둔다. 올바르게는 '개발자는 짝(pair)으로 작업한다'이다.
3번 O: XP는 작은 단위의 기능을 자주 릴리스하여 빠르게 피드백을 받고 결과물을 평가받는 방식을 중시한다. 빈번한 배포는 XP의 전형적 특징이다.
4번 O: 애자일 계열인 XP에서는 요구사항을 사용자 관점의 짧은 설명인 사용자 스토리로 표현한다. 따라서 이 설명도 옳다.

오답 포인트

XP의 실천 항목을 반대로 바꿔 놓는 함정이 핵심이다. 특히 '페어 프로그래밍'을 알고 있으면서도 발문이 '옳지 않은 것'인지 놓치면 2번을 지나치기 쉽다.

다시 나오면 이렇게 판단

애자일 방법론 문제에서는 방법론 이름보다 대표 실천 키워드를 먼저 대조하면 판단이 빨라진다. XP는 테스트 우선, 페어 프로그래밍, 작은 릴리스, 사용자 스토리의 묶음으로 기억하면 된다.

NoteBase

암기 포인트

XP는 테스트 먼저, 둘이 함께, 자주 배포, 요구사항은 사용자 스토리이다.

NoteBase

문 9. 다음에서 설명하는 테스트 케이스 도출 기법은?

입력값 조합의 경우의 수를 줄이기 위해 두 입력값의 조합을 통해 테스트 케이스를 도출하는 기법으로, 모든 경우의 수를 테스트하지 않고 2개 요소의 모든 조합을 확인할 수 있도록 테스트 케이스를 생성한다.

① 결정 테이블(decision table)
② 동등분할(equivalence partitioning)
③ 경곗값 분석(boundary value analysis)
④ 페어와이즈 조합(pairwise combination)

정답 4번

출제 포인트

이 문제는 테스트 케이스 도출 기법 중 페어와이즈 조합의 정의와, 이를 결정 테이블·동등분할·경곗값 분석과 구별하는 기준을 묻는 문제이다.

핵심 개념

결정 테이블은 조건과 조치의 조합을 규칙 형태로 정리해 누락이나 충돌을 점검하는 기법이다. 키워드는 조건-행동 규칙이다.
동등분할은 입력값을 동일하게 처리되는 동치 클래스들로 나눈 뒤 대표값을 선택하는 기법이다. 키워드는 동치 클래스 대표값이다.
경곗값 분석은 오류가 자주 발생하는 경계의 안팎 값을 선택해 시험하는 기법이다. 키워드는 경계 전후 값이다.
페어와이즈 조합은 전체 조합을 모두 시험하지 않고, 두 입력 요소의 모든 쌍이 한 번 이상 나타나도록 테스트 케이스를 구성하는 기법이다. 키워드는 2개 요소의 모든 조합이다.

NoteBase

선지별 해설

① X. 설명문의 핵심은 두 입력값의 모든 쌍을 확인하는 것이다. 결정 테이블은 조건과 결과의 조합을 표 형태의 규칙으로 정리하는 기법이지, 2개 요소 조합만을 만족시키도록 테스트 수를 줄이는 기법은 아니다. 틀린 표현은 '두 입력값의 조합을 통해 테스트 케이스를 도출'이다. 올바르게는 '조건과 조치의 조합을 규칙표로 정리해 각 경우를 점검'이다.
② X. 동등분할은 입력값 전체를 몇 개의 동치 클래스로 나누고 각 클래스의 대표값만 뽑아 시험하는 방식이다. 문제처럼 두 요소의 모든 조합을 보장하는 설명과는 다르다. 틀린 표현은 '2개 요소의 모든 조합을 확인'이다. 올바르게는 '동일하게 처리되는 입력 구간을 대표값으로 나누어 시험'이다.
③ X. 경곗값 분석은 최소값, 최대값, 경계 바로 안팎처럼 오류가 잘 나는 경계 지점을 집중적으로 시험하는 기법이다. 문제의 설명에는 경계값이나 한계점에 대한 언급이 없고, 두 요소 조합 보장이 핵심이다. 틀린 표현은 '입력값 조합의 경우의 수를 줄이기 위해'이다. 올바르게는 '경계 부근 값을 선택해 오류 가능성이 큰 지점을 확인'이다.
④ O. 페어와이즈 조합은 모든 조합을 전부 만들지 않고도 각 두 요소의 조합이 한 번 이상 나타나도록 테스트 케이스를 설계하는 방법이다. 설명문의 '모든 경우의 수를 테스트하지 않고 2개 요소의 모든 조합을 확인'이 바로 페어와이즈 조합의 정의이다.

오답 포인트

출제자는 '경우의 수를 줄인다'는 표현 때문에 동등분할과 혼동하게 만들고, '조합'이라는 단어 때문에 결정 테이블로도 흔들리게 만든다. 하지만 결정 단서는 '2개 요소의 모든 조합'이며, 이 문구가 나오면 페어와이즈 조합으로 바로 연결된다.

다시 나오면 이렇게 판단

선지나 지문에서 전체 조합을 줄이면서도 두 인자 간 모든 쌍을 한 번씩은 확인한다고 설명하면 페어와이즈 조합으로 판단한다. 대표값을 뽑는다는 말이 중심이면 동등분할이고, 경계 부근 값을 본다면 경곗값 분석이며, 조건과 결과를 규칙표로 정리하면 결정 테이블이다.

암기 포인트

페어와이즈 조합은 '전수조합 대신 모든 2쌍 보장'으로 기억한다.

NoteBase

문 10. 소프트웨어 형상 관리에 대한 설명으로 옳지 않은 것은?

① 소프트웨어 제품의 변경된 상태에 대한 정보를 제공해준다.

② 좋은 품질의 소프트웨어를 생산하고 유지보수도 용이하게 해준다.

③ 형상 식별은 변경 요청, 변경 심사, 변경 실시, 변경 확인 등으로 세분화된다.

④ 개발 중 변해 가는 소프트웨어 형상을 체계적으로 관리하고 유지하기 위한 기법이다.

정답 3번

출제 포인트

소프트웨어 형상 관리의 목적과 하위 활동인 형상 식별, 형상 통제, 형상 상태 보고의 차이를 구분하는 문제이다.

핵심 개념

형상 관리는 개발 과정에서 변경되는 소프트웨어 산출물을 체계적으로 식별·통제·기록하여 일관성을 유지하는 관리 활동이다.
형상 식별은 무엇을 관리 대상으로 삼을지 정하고 각 형상 항목의 이름, 구조, 버전을 정의하는 활동이다.
형상 통제는 변경 요청 접수, 변경 심사, 변경 승인, 변경 적용처럼 변경 자체를 심의하고 반영하는 활동이다.
형상 상태 보고는 현재 형상과 변경 이력, 처리 상태를 기록하고 알려 주는 활동이다.

선지별 해설

① O
변경된 상태에 대한 정보를 제공하는 것은 형상 상태 보고의 기능에 해당하므로 옳다. 형상 관리는 각 버전과 변경 이력을 추적할 수 있게 한다.

② O
형상 관리는 변경을 통제하여 일관성을 유지하므로 품질 향상과 유지보수 용이성에 기여한다. 무질서한 수정으로 인한 오류를 줄이는 효과가 있다.

③ X
틀린 부분은 "형상 식별은 변경 요청, 변경 심사, 변경 실시, 변경 확인 등으로 세분화된다"이다.
실제로 이는 형상 식별이 아니라 형상 통제 또는 변경 관리의 절차이다. 형상 식별은 형상 항목 선정, 명명, 버전 부여, 기준선 설정처럼 관리 대상을 정의하는 활동이다.

④ O
개발 중 변해 가는 소프트웨어 형상을 체계적으로 관리하고 유지하기 위한 기법이라는 설명은 소프트웨어 형상 관리의 일반적 정의에 해당하므로 옳다.

NoteBase

오답 포인트

형상 식별과 형상 통제를 이름만 보고 한 범주로 묶어 버리면 ③에 속기 쉽다. 이 문항의 함정은 변경 요청·심사·실시·확인이라는 절차 용어를 일부러 넣어 두고 이를 형상 식별로 바꿔치기한 점이다.

다시 나오면 이렇게 판단

선지에 변경 요청, 심사, 승인, 반영, 확인 같은 절차어가 보이면 먼저 형상 통제인지 확인한다. 반대로 항목 선정, 명칭, 버전, 기준선처럼 관리 대상을 정하는 말이 나오면 형상 식별로 판단할 수 있다.

암기 포인트

형상 식별은 대상을 정하는 것, 형상 통제는 변경을 다루는 것, 형상 상태 보고는 결과를 기록하는 것이다.

NoteBase

문 11. 다음 Python 소스코드에 대한 순환 복잡도는?

```
temp = []
i = 0
d = 2
arr = [1, 2, 3, 4, 5]
while (i < d):
    temp.append(arr[i])
    i = i + 1
i = 0
while (d < len(arr)):
    arr[i] = arr[d]
    i = i + 1
    d = d + 1
arr[:] = arr[: i] + temp
```

① 2
② 3
③ 4
④ 5

정답 2번

출제 포인트

이 문제는 Python 코드의 맥케이브 순환복잡도를 $V(G) = P + 1$로 계산하는 문제이다.

조건 정리

주어진 값: 조건식이 있는 반복문은 while ($i < d$), while ($d < len(arr)$) 두 개이다.
구해야 할 값: 이 코드의 순환복잡도이다.
주의할 조건: 대입문, append, 슬라이싱은 분기를 만들지 않고 조건식이 있는 제어문만 센다.

사용할 공식

$V(G) = P + 1$이다. 여기서 P는 조건식이 있는 의사결정 노드 수이다.
이 공식을 쓰는 이유: 순환복잡도는 코드 안의 독립 경로 수를 보는 지표이므로 if, while, for처럼 분기를 만드는 제어문 개수를 기준으로 계산한다.

NoteBase

계산 과정

1. 첫 번째 while ($i < d$)는 조건 분기 1개이므로 P에 1을 더한다.
2. 두 번째 while ($d < len(arr)$)도 조건 분기 1개이므로 P에 1을 더한다.
3. 따라서 $P = 2$이고, $V(G) = P + 1 = 2 + 1 = 3$이다.

선지별 해설

① X: 2는 while문 개수만 세고 마지막 +1을 빠뜨린 값이다. 틀린 판단은 '순환복잡도는 분기 수와 같다'이고, 올바르게는 '순환복잡도는 분기 수에 1을 더한 값이다'이다.
② O: 조건식이 있는 제어문이 2개이므로 $V(G) = 2 + 1 = 3$이다. 따라서 3을 제시한 2번이 맞다.
③ X: 4는 분기 수를 과대계산한 값이다. 대입문이나 append까지 경로 분기로 세면 안 된다. 올바르게는 조건식이 있는 제어문 2개만 세어 3이 된다.
④ X: 5는 시간복잡도나 실행 횟수를 순환복잡도로 오해한 값이다. 순환복잡도는 반복이 몇 번 도는지가 아니라 분기 구조가 몇 개인지를 본다. 올바르게는 3이다.

오답 포인트

while문이 두 개이므로 2라고 바로 답하거나, 반복문의 수행 횟수를 떠올려 시간복잡도처럼 크게 잡는 실수를 유도한 문항이다. 순환복잡도는 실행 시간과 다른 개념이라는 점이 함정이다.

다시 나오면 이렇게 판단

순환복잡도 문제는 조건식이 있는 제어문 수를 먼저 세고 마지막에 항상 1을 더해 판단한다. 대입문, 함수 호출, 리스트 조작은 별도 분기로 세지 않는다.

암기 포인트

순환복잡도는 '분기 수 + 1'이며, if·while·for 하나당 분기 1개로 계산한다.

NoteBase

문 12. 소프트웨어 유지보수 시 비용이 가장 많이 드는 것은?

① 적응 유지보수(adaptive maintenance)

② 완전 유지보수(perfective maintenance)

③ 수정 유지보수(corrective maintenance)

④ 예방 유지보수(preventive maintenance)

정답 2번

출제 포인트

이 문제는 소프트웨어 유지보수 유형의 정의와 비용 비중의 차이를 묻는 문제이다.

핵심 개념

적응 유지보수는 운영체제, 하드웨어, 업무 절차처럼 외부 환경이 바뀌었을 때 그 변화에 맞추는 작업이다. 키워드는 환경 변화 대응이다.
완전 유지보수는 사용자 요구를 반영하여 기능, 성능, 사용성을 개선하는 작업이다. 키워드는 기능·성능 개선이다.
수정 유지보수는 운영 중 발견된 오류를 찾아 고치는 작업이다. 키워드는 결함 수정이다.
예방 유지보수는 미래 장애 가능성을 줄이기 위해 구조를 개선하고 품질을 높이는 작업이다. 키워드는 미래 결함 예방이다.

선지별 해설

① X: 적응 유지보수는 환경 변화에 맞추는 유지보수이다. 운영체제 변경, 하드웨어 교체, 제도 변화 대응이 여기에 해당한다. 비용이 가장 큰 유형은 아니다. 틀린 표현은 "가장 많이 드는"이다. 올바르게는 "환경 변화에 대응하는 유지보수"이다.
② O: 완전 유지보수는 기능 추가, 성능 향상, 사용성 개선처럼 사용 중 제기되는 개선 요구를 반영하는 유지보수이다. 실제 유지보수 비용은 이런 개선 작업에 가장 많이 투입되므로 옳다.
③ X: 수정 유지보수는 배포 후 발견된 결함을 수정하는 유지보수이다. 오류를 고치는 작업이므로 중요하지만 비용 비중이 가장 큰 유형은 아니다. 틀린 표현은 "가장 많이 드는"이다. 올바르게는 "결함을 수정하는 유지보수"이다.
④ X: 예방 유지보수는 앞으로 발생할 수 있는 장애를 줄이기 위해 구조 개선, 문서 보완, 리팩터링 등을 수행하는 유지보수이다. 미래 위험을 줄이는 목적의 유지보수이지만 비용 비중이 가장 큰 유형은 아니다. 틀린 표현은 "가장 많이 드는"이다. 올바르게는 "미래 장애를 예방하는 유지보수"이다.

NoteBase

오답 포인트

수험생은 유지보수라고 하면 오류 수정부터 떠올려 수정 유지보수를 고르기 쉽다. 그러나 시험에서는 운영 이후 가장 큰 비용이 사용자 요구 반영과 품질 향상에 들어간다고 보고 완전 유지보수를 정답으로 정리한다.

다시 나오면 이렇게 판단

유지보수 유형 문제는 변화의 원인을 먼저 구분하면 된다. 환경 변화면 적응, 이미 생긴 오류면 수정, 미래 오류를 줄이면 예방, 기능과 성능을 더 좋게 만들면 완전 유지보수이다. 비용 비중을 묻는 문항이면 완전 유지보수를 우선 판단 기준으로 둔다.

암기 포인트

수정은 고장 수리, 적응은 환경 대응, 예방은 미래 대비, 완전은 기능·성능 개선이며 비용 비중은 완전 유지보수가 가장 크다.

NoteBase

문 13. 다음 다이어그램에서 표현된 객체지향의 개념으로만 옳게 짝지은 것은?



① 다형성, 상속

② 캡슐화, 다형성

③ 정보은닉, 상속

④ 캡슐화, 정보은닉

정답 4번

출제 포인트

이 문제는 UML 클래스 다이어그램의 접근 제어 기호와 클래스 구성 요소를 보고 객체지향 개념을 판별하는 문제이다. 하나의 클래스 안에 속성(#color, -area)과 연산(+fill_color(), +erase(), +set_area())이 함께 묶여 있어 캡슐화가 나타나고, #와 - 기호로 내부 데이터를 외부에서 직접 다루지 못하게 하여 정보은닉이 나타난다.

NoteBase

선지별 해설

① X
다형성은 같은 메시지에 대해 객체가 서로 다르게 반응하는 성질이고, 상속은 부모-자식 클래스 관계가 드러나야 한다. 이 도식에는 단일 클래스만 있으며 상속 화살표나 재정의 관계가 보이지 않는다. 틀린 표현은 "다형성, 상속"이며, 올바르게는 "캡슐화, 정보은닉"이다.

② X
캡슐화는 맞지만 다형성은 아니다. 속성과 메서드가 한 클래스에 함께 정의되어 있는 점은 캡슐화를 보여 주지만, 동일 연산의 다중 구현이나 동적 바인딩을 나타내는 정보는 없다. 틀린 표현은 "다형성"이며, 올바르게는 "정보은닉"이다.

③ X
정보은닉은 맞지만 상속은 아니다. #color와 -area는 접근 범위를 제한하여 내부 상태를 숨기는 표시이지만, 클래스 간 일반화 관계는 제시되지 않았다. 틀린 표현은 "상속"이며, 올바르게는 "캡슐화"이다.

④ O
속성과 메서드를 Figure 클래스 안에 묶어 둔 점은 캡슐화이다. 또 #color는 protected, -area는 private 접근 수준을 뜻하므로 내부 데이터를 외부에서 직접 접근하지 못하게 하는 정보은닉이 함께 표현되어 있다.

오답 포인트

접근 제어 기호가 보이면 정보은닉을 먼저 떠올려야 하는데, 객체지향의 대표 개념이라는 이유만으로 다형성이나 상속을 함께 고르기 쉽다. 이 문제에서는 클래스가 하나뿐이므로 상속과 다형성은 도식만으로 확인될 수 없다는 점이 함정이다.

다시 나오면 이렇게 판단

클래스 다이어그램에서는 먼저 클래스가 하나인지 여러 개인지 본다. 하나의 클래스 안에 데이터와 연산이 함께 있으면 캡슐화를 의심하고, +, #, - 같은 가시성 기호가 있으면 정보은닉 여부를 판단한다. 부모-자식 연결선이 없으면 상속이 아니고, 동일 메시지의 다른 구현 관계가 없으면 다형성도 아니다.

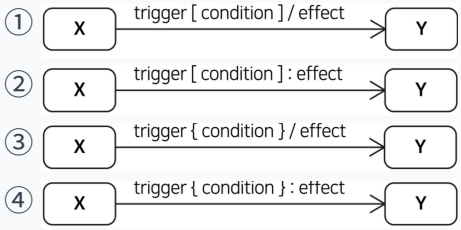
암기 포인트

한 클래스에 묶으면 캡슐화, 접근 제한 기호가 보이면 정보은닉이다.

NoteBase

문 14. 다음 제시문을 UML 상태머신(state machine) 다이어그램 표기법으로 바르게 나타낸 것은?

상태 X에서 trigger 이벤트가 발생했을 때, condition 조건이 참이면 effect 효과를 수행하고 상태 Y로 전이한다.



정답 1번

출제 포인트

이 문제는 UML 상태머신의 상태 전이 라벨을 `이벤트 [가드조건] / 효과` 형식으로 표기하는 규칙을 묻는 문제이다.

선지별 해설

① O
- UML 상태 전이에서 이벤트가 발생하고 가드 조건이 참이면 효과를 수행한 뒤 다음 상태로 이동할 때 `trigger [condition] / effect`로 적는다. 제시문과 정확히 일치한다.

② X
- `:`가 틀린 표현이다. 효과를 적는 구분자는 콜론이 아니라 `/`이다. 올바르게는 `trigger [condition] / effect`이다.

③ X
- `{condition}`이 틀린 표현이다. 상태 전이의 가드 조건은 중괄호가 아니라 대괄호 `[]`로 감싼다. 올바르게는 `trigger [condition] / effect`이다.

④ X
- 가드를 `{}`로 쓰고 효과 앞 구분자를 `:`로 쓴 부분이 모두 틀리다. 올바르게는 대괄호와 슬래시를 사용한 `trigger [condition] / effect`이다.

오답 포인트

이 문항은 가드 조건의 괄호 기호와 효과 구분 기호를 함께 묻는다. `[]`와 `/`를 동시에 기억하지 못하면 ②처럼 구분자만 바뀐 선지나 ③처럼 괄호만 바뀐 선지에 흔들리기 쉽다.

NoteBase

다시 나오면 이렇게 판단

상태 전이 라벨을 보면 `이벤트 [조건] / 동작` 순서인지 먼저 본다. 가드는 대괄호인지, 효과 앞 기호는 슬래시인지 확인하면 유사 선지를 빠르게 걸러낼 수 있다.

암기 포인트

UML 상태 전이는 `trigger [guard] / effect`이며, 가드는 `[]`, 효과는 `/`이다.

NoteBase

문 15. UML 다이어그램에 대한 용어와 설명을 바르게 연결한 것은?

(가) 커뮤니케이션 다이어그램
(나) 액티비티 다이어그램
(다) 시퀀스 다이어그램
A. 객체와 액터의 관계에 초점을 두고, 사건의 흐름을 네트워크 형태로 압축하여 나타낸다.
B. 사용 사례를 이루는 객체들이 어떻게 메시지를 보내 협력하는지를 시간의 흐름에 따라 표시한다.
C. 복잡한 시스템 작업의 병행 처리를 위해 포크(fork), 조인(join)을 이용한다.

① (가): A, (나): B, (다): C
② (가): A, (나): C, (다): B
③ (가): B, (나): C, (다): A
④ (가): C, (나): A, (다): B

정답 2번

출제 포인트

이 문제는 UML의 커뮤니케이션 다이어그램, 액티비티 다이어그램, 시퀀스 다이어그램의 핵심 표현 대상을 정확히 구별하는지를 묻는 문제이다.

핵심 개념

커뮤니케이션 다이어그램은 객체들 사이의 연결 관계와 메시지 교환을 네트워크 형태로 표현하는 상호작용 다이어그램이다.
액티비티 다이어그램은 작업 흐름과 제어 흐름을 표현하며 병행 처리에서는 fork와 join을 사용한다.
시퀀스 다이어그램은 시간의 흐름에 따라 객체들이 어떤 순서로 메시지를 주고받는지를 표현한다.

NoteBase

선지별 해설

① X
(가)=A는 맞다. A의 '객체와 액터의 관계', '네트워크 형태'라는 표현은 커뮤니케이션 다이어그램의 특징에 해당한다.
(나)=B는 틀리다. B의 '시간의 흐름에 따라 메시지를 보내 협력'은 시퀀스 다이어그램 설명이다. 액티비티 다이어그램은 실제로는 C처럼 작업 흐름과 병행 처리를 표현한다.
(다)=C는 틀리다. C의 'fork, join'은 액티비티 다이어그램의 요소이다. 시퀀스 다이어그램은 실제로는 B처럼 시간 축을 따라 메시지 순서를 표현한다.

② O
(가)=A, (나)=C, (다)=B의 연결이 모두 맞다. 커뮤니케이션 다이어그램은 관계 중심의 네트워크 표현이고, 액티비티 다이어그램은 fork/join으로 병행 흐름을 표현하며, 시퀀스 다이어그램은 시간 순서에 따른 메시지 흐름을 표현한다.

③ X
(가)=B는 틀리다. B는 시퀀스 다이어그램 설명이다. 커뮤니케이션 다이어그램은 실제로는 A처럼 객체 간 연결 구조를 중심으로 본다.
(나)=C는 맞다. 액티비티 다이어그램은 병행 처리 표현을 위해 fork와 join을 사용한다.
(다)=A는 틀리다. A는 커뮤니케이션 다이어그램 설명이다. 시퀀스 다이어그램은 실제로는 B처럼 시간의 흐름에 따른 메시지 순서를 보여 준다.

④ X
(가)=C는 틀리다. C는 액티비티 다이어그램 설명이다. 커뮤니케이션 다이어그램은 실제로는 A처럼 객체 간 관계와 메시지 연결을 네트워크로 나타낸다.
(나)=A는 틀리다. A는 커뮤니케이션 다이어그램 설명이다. 액티비티 다이어그램은 실제로는 C처럼 흐름과 병행 제어를 표현한다.
(다)=B는 맞다. 시퀀스 다이어그램은 시간 흐름에 따른 객체 간 메시지 교환 순서를 나타낸다.

오답 포인트

커뮤니케이션 다이어그램과 시퀀스 다이어그램은 모두 객체 간 상호작용을 다루므로 혼동하기 쉽다. 이때 '네트워크 형태의 관계 중심'이면 커뮤니케이션 다이어그램이고, '시간 흐름에 따른 메시지 순서 중심'이면 시퀀스 다이어그램이다. 또한 fork와 join이 보이면 액티비티 다이어그램으로 바로 연결된다.

다시 나오면 이렇게 판단

UML 다이어그램 문제는 먼저 표현 축을 잡으면 빠르게 풀린다. 관계 구조를 그리면 커뮤니케이션, 시간 순서를 그리면 시퀀스, 업무 흐름과 병행 처리를 그리면 액티비티로 판단된다.

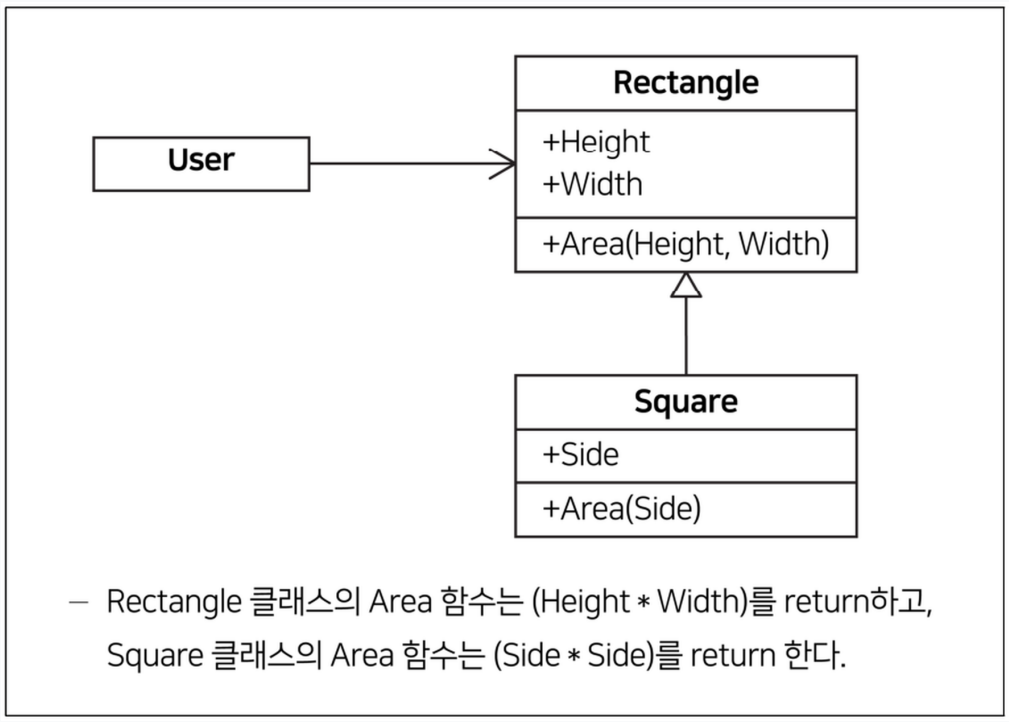
NoteBase

암기 포인트

커뮤니케이션은 관계망, 시퀀스는 시간축, 액티비티는 흐름과 fork/join이다.

NoteBase

문 16. 객체지향의 SOLID 설계 원리 중 다음 클래스 다이어그램이 위반한 설계 원리는?



① 개방-폐쇄 원리(open-closed principle)

② 단일 책임의 원리(single responsibility principle)

③ 리스코프 치환 원리(Liskov substitution principle)

④ 의존 관계 역전의 원리(dependency inversion principle)

정답 3번

출제 포인트

이 문제는 Rectangle을 상속한 Square가 상위 타입의 가정인 가로와 세로의 독립성을 유지하는지, 즉 하위 타입이 상위 타입을 그대로 대체할 수 있는지를 도식에서 판단하는 문제이다.

NoteBase

선지별 해설

1번: X. 개방-폐쇄 원리는 기존 코드를 크게 수정하지 않고 확장할 수 있어야 한다는 원리이다. 이 도식의 핵심 문제는 확장 가능성 자체보다 Rectangle의 자식인 Square가 같은 방식으로 사용될 수 없다는 점이다. 틀린 표현은 "이 도식이 개방-폐쇄 원리를 위반했다"이고, 올바르게는 "이 도식의 직접적인 문제는 리스코프 치환 원리 위반이다"이다.
2번: X. 단일 책임 원리는 클래스가 하나의 책임만 가져야 한다는 원리이다. 도식에는 Rectangle과 Square가 각각 도형의 상태와 넓이 계산이라는 자기 역할을 가지므로 책임이 뒤섞였다는 근거가 없다. 틀린 표현은 "책임이 여러 개라서 문제다"이고, 올바르게는 "문제는 책임 수가 아니라 상속 관계에서 행위 계약이 깨진 점이다"이다.
3번: O. 리스코프 치환 원리는 하위 타입이 상위 타입을 대체해도 프로그램 의미가 유지되어야 한다는 원리이다. Rectangle은 Height와 Width를 독립적으로 받아 Area(Height, Width)를 계산하는데, Square는 Side 하나로 Area(Side)를 계산하므로 Rectangle이 기대하는 사용 방식과 계약을 그대로 만족하지 못한다. 따라서 Square를 Rectangle의 하위 타입으로 두는 설계가 치환 가능성을 깨뜨린다.
4번: X. 의존 관계 역전 원리는 구체 클래스보다 추상화에 의존해야 한다는 원리이다. 그림에서 User가 Rectangle에 직접 의존하는 모습만 보면 추상화 부족을 떠올릴 수는 있지만, 이 문제의 핵심 위반은 User의 의존 방향이 아니라 Square가 Rectangle을 대체하지 못하는 상속 구조이다. 틀린 표현은 "이 그림의 본질적 위반이 의존 관계 역전이다"이고, 올바르게는 "의존 관계 역전 문제를 논하려면 Shape 같은 추상 타입 의존이 필요하지만, 이 문항의 핵심은 리스코프 치환 원리 위반이다"이다.

오답 포인트

User가 Rectangle에 직접 연결된 화살표 때문에 4번으로 흔들리기 쉽다. 그러나 출제자는 의존 방향보다 Rectangle과 Square의 상속 관계가 자연스러운지, 즉 정사각형이 직사각형을 언제나 그대로 대체할 수 있는지를 묻고 있다.

다시 나오면 이렇게 판단

상속 문제가 나오면 먼저 상위 클래스의 속성과 메서드 계약을 적고, 하위 클래스가 그 계약을 약화하거나 입력 구조를 바꾸는지 본다. 하위 타입이 상위 타입 자리에 들어가도 같은 방식으로 호출되고 같은 의미를 유지해야 하면 리스코프 치환 원리 문제이다.

암기 포인트

정사각형이 직사각형을 문법상 상속해도 가로와 세로의 독립 계약을 못 지키면 리스코프 치환 원리 위반이다.

NoteBase

문 17. 다음 요구사항 추출 방법에 대한 설명으로 (가), (나)에 들어갈 용어를 바르게 연결한 것은?

(가)은/는 여러 명으로부터 정보를 얻을 수 있는 효과적인 방법이다. 그룹으로 테이블에 앉아 아이디어를 추출할 목적으로 특정 토픽에 관하여 토론을 수행하며, 판단과 비판을 유보함으로써 창의성을 극대화할 수 있는 요구 추출 방법이다.
(나)은 사용자의 기초적인 요구를 최대한 반영한 소규모 모형을 개발하여 이를 기반으로 더욱 구체적인 요구 분석을 가능하게 하는 방법이다. 주요 목적은 소프트웨어 엔지니어의 아이디어에 대한 피드백을 조기에 받아 요구사항을 취합하는 것이다.

① (가): 인터뷰, (나): 관찰
② (가): 인터뷰, (나): 프로토타이핑
③ (가): 브레인스토밍, (나): 관찰
④ (가): 브레인스토밍, (나): 프로토타이핑

정답 4번

출제 포인트

요구사항 추출 기법의 핵심 특징을 읽고 브레인스토밍, 인터뷰, 관찰, 프로토타이핑을 구별하는 문제이다.

핵심 개념

브레인스토밍은 여러 사람이 자유롭게 아이디어를 내고 비판을 유보하여 요구를 끌어내는 집단 발상 기법이다.
인터뷰는 이해관계자에게 질문을 던져 요구를 직접 수집하는 대화 중심 기법이다.
관찰은 사용자의 실제 작업 환경과 업무 수행 모습을 지켜보며 암묵적 요구를 파악하는 기법이다.
프로토타이핑은 초기 요구를 반영한 시제품을 먼저 만들어 사용자 피드백으로 요구를 구체화하는 기법이다.

NoteBase

선지별 해설

① X
(가)는 여러 사람이 테이블에 앉아 특정 주제에 대해 토론하고 판단과 비판을 유보한다는 설명이므로 인터뷰가 아니라 브레인스토밍이다.
틀린 표현은 '(가)=인터뷰'이다. 올바르게는 '(가)=브레인스토밍'이다.
(나)는 사용자의 기초 요구를 반영한 소규모 모형을 개발해 조기 피드백을 받는 방식이므로 관찰이 아니라 프로토타이핑이다.
틀린 표현은 '(나)=관찰'이다. 올바르게는 '(나)=프로토타이핑'이다.
② X
(나)는 프로토타이핑 설명과 일치하지만, (가)가 인터뷰라는 연결이 틀렸다.
틀린 표현은 '(가)=인터뷰'이다. 올바르게는 '(가)=브레인스토밍'이다.
③ X
(가)는 브레인스토밍 설명과 일치하지만, (나)가 관찰이라는 연결이 틀렸다.
틀린 표현은 '(나)=관찰'이다. 올바르게는 '(나)=프로토타이핑'이다.
④ O
(가)는 집단 토론, 자유로운 아이디어 제시, 비판 유보라는 키워드가 모두 브레인스토밍의 특징과 맞는다.
(나)는 소규모 모형 개발, 조기 피드백, 요구 구체화라는 흐름이 프로토타이핑의 특징과 정확히 맞는다.

오답 포인트

(가)에서 '여러 명', '토론', '판단과 비판 유보'를 놓치면 인터뷰와 혼동하기 쉽다. 인터뷰는 질문과 응답이 중심이고, 브레인스토밍은 집단 발상과 비판 유보가 핵심이다. (나)에서도 '모형을 개발하여 피드백을 조기에 받는다'는 표현을 보지 못하면 관찰로 착각하기 쉽다. 관찰은 사용자 행동을 보는 것이고, 프로토타이핑은 시제품을 만들어 반응을 확인하는 것이다.

다시 나오면 이렇게 판단

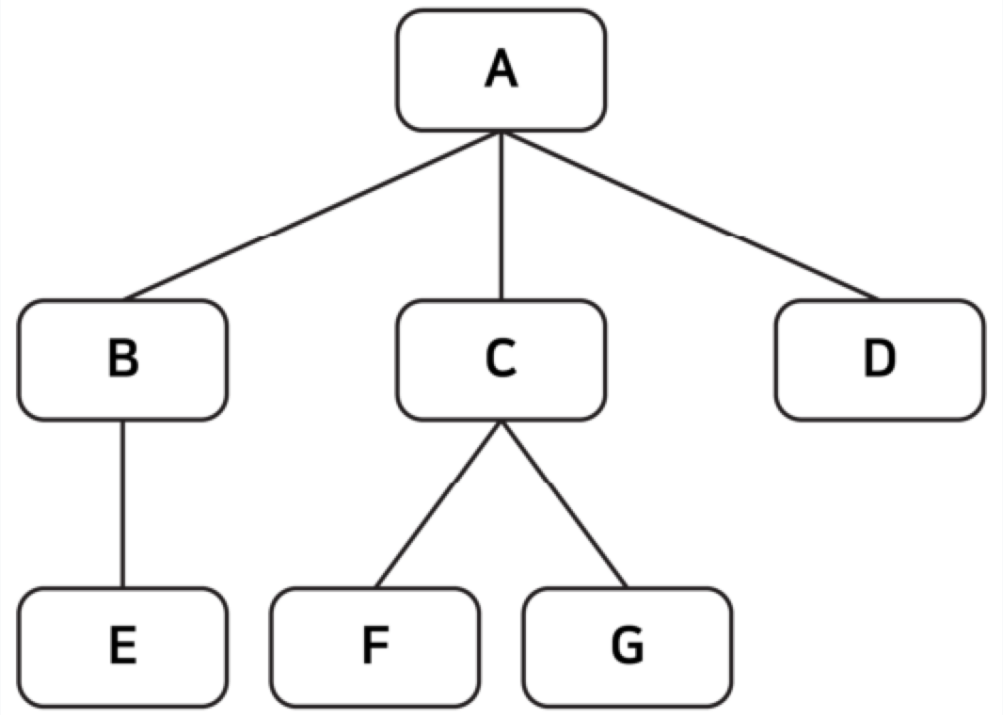
집단으로 아이디어를 쏟아내고 비판을 미루면 브레인스토밍으로 판단한다. 질문과 답변으로 요구를 모으면 인터뷰로 판단한다. 실제 업무 수행을 지켜보면 관찰로 판단한다. 시제품을 먼저 만들어 사용자 반응을 받으면 프로토타이핑으로 판단한다.

암기 포인트

브레인스토밍은 비판 유보형 집단 발상이고, 프로토타이핑은 시제품 피드백형 요구 구체화이다.

NoteBase

문 18. 다음과 같은 모듈 계층 구조를 가진 시스템의 테스팅을 수행하는 과정에 대한 설명으로 옳은 것만을 모두 고르면?



(가) 하향식 기법의 통합 테스팅은 모듈 A부터 시작한다.
(나) 빅뱅 테스팅을 수행하기 위해서는 A부터 G까지의 모든 모듈을 한꺼번에 연결하여 테스팅을 수행한다.
(다) 하향식 기법을 수행하기 위해서는 스텁(stub) 모듈이 필요하다.
(라) 상향식 기법을 수행하기 위해서는 드라이버(driver) 모듈이 필요하다.

① (가), (나)
② (다), (라)
③ (가), (나), (다)
④ (가), (나), (다), (라)

정답 4번

출제 포인트

이 문제는 모듈 계층도에서 상향식 통합, 하향식 통합, 빅뱅 테스트의 적용 방식과 스텁·드라이버의 쓰임을 정확히 구분하는 문제이다.

NoteBase

선지별 해설

① X
- ①의 "(가), (나)"만 옳다고 본 부분이 틀렸다. (가)는 하향식 통합 테스트가 최상위 모듈 A부터 시작된다는 점에서 옳고, (나)도 빅뱅 테스트가 모든 모듈을 한꺼번에 결합해 시험한다는 뜻이므로 옳다. 그러나 (다)와 (라)도 모두 옳으므로 올바르게는 "(가), (나), (다), (라)"가 모두 옳다.

② X
- ②의 "(다), (라)"만 옳다고 본 부분이 틀렸다. 하향식 통합에서는 아직 구현되지 않은 하위 모듈을 대신할 스텁이 필요하므로 (다)는 옳고, 상향식 통합에서는 상위 호출 모듈을 흉내 낼 드라이버가 필요하므로 (라)도 옳다. 다만 (가), (나) 역시 옳으므로 올바르게는 "(가), (나), (다), (라)"가 모두 옳다.

③ X
- ③의 "(가), (나), (다)"만 옳다고 본 부분이 틀렸다. (라)도 상향식 기법에서 필요한 보조 모듈이 드라이버라는 설명이므로 옳다. 올바르게는 "(가), (나), (다), (라)"가 모두 옳다.

④ O
- (가)는 하향식 통합 테스트가 제어 구조의 상위 모듈부터 내려가며 통합된다는 설명이므로 옳다.
- (나)는 빅뱅 테스트가 A부터 G까지 모든 모듈을 한 번에 연결해 통합 시험한다는 설명이므로 옳다.
- (다)는 하향식 기법에서 아직 없는 하위 모듈 대신 스텁을 사용하므로 옳다.
- (라)는 상향식 기법에서 아직 없는 상위 모듈 대신 드라이버를 사용하므로 옳다.

오답 포인트

스텁과 드라이버의 방향을 뒤집어서 외우면 (다)와 (라) 중 하나를 틀리기 쉽다. 또 빅뱅 테스트는 단계적으로 붙이는 방식이 아니라 전 모듈을 한꺼번에 결합하는 방식이라는 점을 놓치면 (나)를 흔들리기 쉽다.

다시 나오면 이렇게 판단

통합 방향을 먼저 본다. 위에서 아래로 내려가면 하향식이므로 스텁이 필요하고, 아래에서 위로 올리면 상향식이므로 드라이버가 필요하다. 전 모듈 일괄 결합이면 빅뱅 테스트로 판단한다.

암기 포인트

하향식은 스텁, 상향식은 드라이버, 빅뱅은 전부 한꺼번에 결합이다.

NoteBase

문 19. 다음 C 언어로 작성된 소스코드에서 나타나는 결합도로 옳은 것은?

```c
#include <stdio.h>
int sum_array(int *arr, int length) {
    int i, ret;
    ret = 0;
    for (i = 0; i < length; i++)
        ret += arr[i];
    return ret;
}

int main() {
    int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
    printf("%d", sum_array(a, 10));
    return 0;
}
```

① 제어 결합도

② 외부 결합도

③ 스탬프 결합도

④ 데이터 결합도

정답 3번

출제 포인트

이 문제는 결합도의 종류 중 데이터 결합도와 스탬프 결합도를 구별할 수 있는지를 묻는다.

핵심 개념

데이터 결합도는 모듈 사이에 필요한 단순 자료만 매개변수로 주고받는 형태이다.
스탬프 결합도는 배열, 레코드, 구조체처럼 묶인 자료구조 전체를 전달하는 형태이다.
제어 결합도는 처리 흐름을 바꾸는 제어 정보가 전달되는 경우이고, 외부 결합도는 외부 장치, 프로토콜, 파일 형식 같은 외부 환경을 함께 의존하는 경우이다.

NoteBase

선지별 해설

① X: 제어 결합도는 분기나 동작 모드를 결정하는 제어 플래그를 다른 모듈에 넘길 때 나타난다. 이 코드에서는 합계를 구할 배열과 길이만 전달하고 처리 방향을 지시하는 제어 정보는 없다. 틀린 표현은 '제어'이며, 올바르게는 제어 정보 전달이 아니라 자료구조 전달에 가깝다.
② X: 외부 결합도는 통신 프로토콜, 파일 포맷, 장치 인터페이스처럼 외부적으로 정해진 환경을 여러 모듈이 함께 사용할 때 나타난다. 이 코드는 단순 함수 호출일 뿐 외부 장치나 외부 형식 의존이 없다. 틀린 표현은 '외부'이며, 올바르게는 외부 환경 결합이 아니다.
③ O: `sum_array(a, 10)`은 배열 `a`를 함수에 전달한다. 배열은 여러 원소가 묶인 자료구조이므로 필요한 값 하나만 넘기는 데이터 결합도보다 스탬프 결합도의 특징에 해당한다.
④ X: 데이터 결합도는 정수 하나, 실수 하나처럼 필요한 단순 자료만 개별적으로 전달할 때의 결합이다. 이 코드에는 `length`라는 단순 자료도 있지만 핵심은 배열 전체가 함께 전달된다는 점이다. 틀린 표현은 '데이터 결합도'이며, 올바르게는 단순 자료만 주고받는 경우에 데이터 결합도라고 본다.

오답 포인트

배열이 C에서 포인터처럼 전달된다는 구현 세부만 보고 데이터 결합도로 착각하기 쉽다. 결합도 판정은 언어 문법보다 모듈 사이에 단순 값만 오가는지, 묶인 자료구조가 오가는지를 기준으로 본다.

다시 나오면 이렇게 판단

매개변수가 스칼라 값 몇 개면 데이터 결합도, 배열이나 구조체처럼 묶인 자료면 스탬프 결합도로 먼저 판단한다. 여기에 제어 플래그가 있으면 제어 결합도 여부를 추가로 본다.

암기 포인트

값 몇 개만 넘기면 데이터, 자료 묶음째 넘기면 스탬프이다.

NoteBase

문 20. 응집도를 강한 것부터 순서대로 나열할 때, (가) ~ (다)에 들어갈 용어로 적절하지 않은 것은?

기능적 응집 - (가) - (나) - (다) - 우연적 응집

① (가): 절차적 응집, (나): 시간적 응집, (다): 논리적 응집
② (가): 교환적 응집, (나): 절차적 응집, (다): 시간적 응집
③ (가): 순차적 응집, (나): 교환적 응집, (다): 논리적 응집
④ (가): 절차적 응집, (나): 순차적 응집, (다): 시간적 응집

정답 4번

출제 포인트

이 문제는 응집도의 강약 순서를 구별하는 문제이며, 특히 순차적 응집과 절차적 응집의 선후관계를 정확히 아는지가 핵심이다.

올바른 순서

응집도는 일반적으로 기능적 응집 → 순차적 응집 → 교환적 응집 → 절차적 응집 → 시간적 응집 → 논리적 응집 → 우연적 응집 순으로 약해진다.
기능적 응집은 하나의 잘 정의된 기능을 수행하므로 가장 강하다.
순차적 응집은 앞 단계의 출력이 뒤 단계의 입력이 되는 형태라서 절차적 응집보다 강하다.
교환적 응집은 동일한 데이터나 정보 집합을 중심으로 묶인다.
절차적 응집은 실행 순서만 공통일 뿐 데이터 의존성은 순차적 응집보다 약하다.
시간적 응집은 같은 시점에 수행된다는 이유로 묶이고, 논리적 응집은 유사 성격의 기능을 한 모듈에 모은 수준이며, 우연적 응집이 가장 약하다.

선지별 해설

① O: 절차적 응집 → 시간적 응집 → 논리적 응집의 순서는 맞다. 절차적 응집은 처리 순서의 관련성이 있고, 시간적 응집은 같은 시점 수행이라는 느슨한 기준이며, 논리적 응집은 유사 기능을 임의 선택으로 묶은 수준이라 더 약하다.
② O: 교환적 응집 → 절차적 응집 → 시간적 응집의 순서는 맞다. 같은 데이터 집합을 중심으로 묶이는 교환적 응집이 단순 처리 순서 중심의 절차적 응집보다 강하고, 절차적 응집은 시간적 응집보다 강하다.
③ O: 순차적 응집 → 교환적 응집 → 논리적 응집의 순서는 맞다. 순차적 응집은 앞 작업의 결과가 뒤 작업의 입력이 되는 강한 의존관계가 있고, 교환적 응집은 그보다 약하지만 논리적 응집보다는 강하다.
④ X: 틀린 부분은 '(가) 절차적 응집, (나) 순차적 응집'의 배치이다. 순차적 응집은 단계 간 출력·입력 의존이 있으므로 절차적 응집보다 강하다. 올바르게는 기능적 응집 → 순차적 응집 → 교환적 응집 → 절차적 응집 → 시간적 응집 → 논리적 응집 → 우연적 응집이다.

NoteBase

오답 포인트

많이 헷갈리는 부분은 순차적 응집과 절차적 응집의 차이이다. 둘 다 순서가 있어 보이지만, 순차적 응집은 앞 단계 결과가 다음 단계 입력이 되는 데이터 의존이 있고, 절차적 응집은 단지 실행 순서만 공유한다는 점에서 더 약하다.

다시 나오면 이렇게 판단

응집도 순서 문제는 먼저 기능적 응집이 최강, 우연적 응집이 최약이라는 양 끝을 잡고, 그 사이에서는 데이터 의존이 있는 순차적 응집이 단순 절차 공유인 절차적 응집보다 앞선다고 판단하면 된다. 같은 시점 기준의 시간적 응집과 유사 기능 묶음인 논리적 응집은 그보다 더 약한 쪽에 둔다.

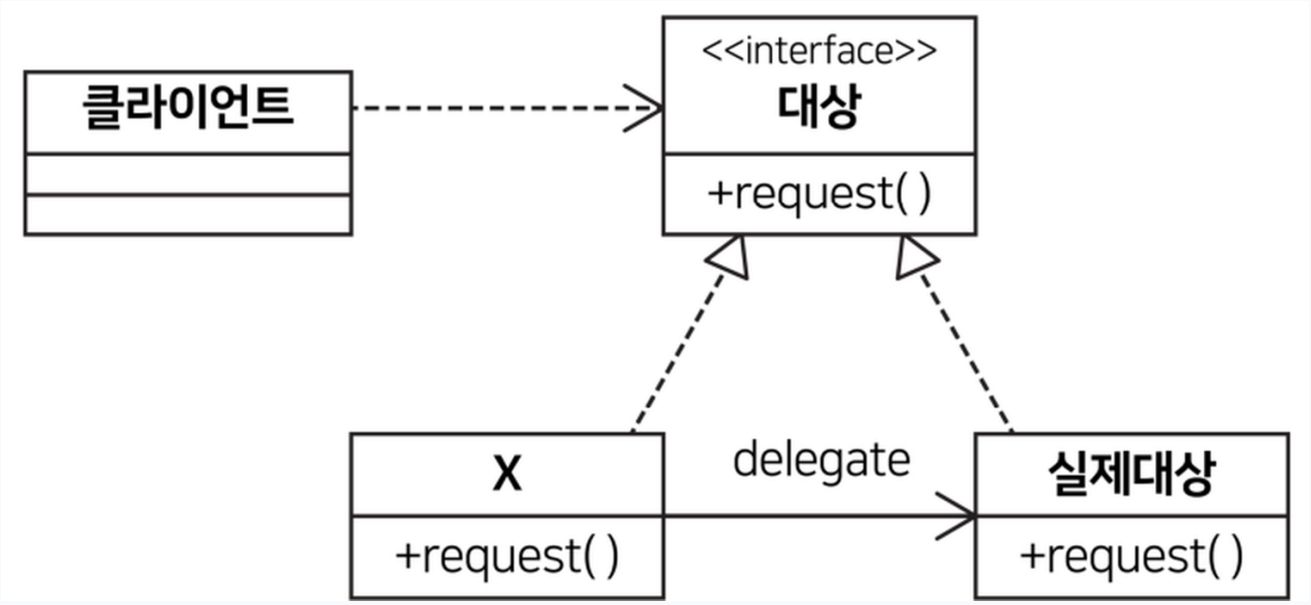
암기 포인트

응집도는 기능-순차-교환-절차-시간-논리-우연 순으로 약해진다.

NoteBase

문 21. 다음에서 설명하는 디자인 패턴은?

그림에서 '클라이언트'는 '대상' 인터페이스를 통해서 'request()'를 실행하는데, 어떤 객체를 통해서 'request()'가 처리되는지 알지 못하고 'X' 객체가 '실제대상' 객체를 참조하고 있으면서 '클라이언트'의 요청에 대응하여 '실제대상' 객체에게 실제 'request()' 실행을 요청하여 필요한 작업이 이루어진다.



① Proxy 패턴

② Bridge 패턴

③ Adapter 패턴

④ Chain of responsibility 패턴

정답 1번

출제 포인트

이 문제는 클라이언트가 대상 인터페이스만 알고, 중간 객체 X가 실제대상을 참조하여 같은 request()를 위임 호출하는 구조를 보고 Proxy 패턴을 식별하는 문제이다.

NoteBase

선지별 해설

① O: Proxy 패턴은 대리 객체가 실제대상과 동일한 인터페이스를 제공하고, 내부에서 실제대상을 참조하여 요청을 위임 처리하는 구조이다. 그림의 X가 대상 인터페이스를 구현하면서 실제대상에게 request()를 delegate하므로 Proxy 패턴에 해당한다.
② X: Bridge 패턴은 추상화와 구현을 분리하여 둘을 독립적으로 확장하는 구조이다. 이 그림은 추상화 계층과 구현 계층의 분리보다 대리 객체 X가 실제대상을 대신 호출하는 점이 핵심이다. 틀린 표현은 '구현 분리 구조'로 보는 점이며, 올바르게는 '실제대상에 대한 접근을 대리하는 구조'이다.
③ X: Adapter 패턴은 기존 인터페이스를 클라이언트가 기대하는 다른 인터페이스로 변환하는 구조이다. 이 그림에서는 인터페이스 변환이 아니라 대상 인터페이스 request()를 그대로 유지한 채 실제대상 호출만 위임한다. 틀린 표현은 '호환되지 않는 인터페이스를 맞춘다'는 점이며, 올바르게는 '동일 인터페이스로 실제대상 호출을 대리한다'이다.
④ X: Chain of responsibility 패턴은 여러 처리 객체가 사슬처럼 연결되어 요청을 넘겨 가며 처리할 대상을 찾는 구조이다. 이 그림은 여러 처리자에게 순차 전달하는 구조가 아니라 X 하나가 실제대상 하나를 참조하여 직접 위임한다. 틀린 표현은 '연쇄적 전달 구조'라는 점이며, 올바르게는 '단일 대리 객체의 위임 구조'이다.

오답 포인트

Adapter와 Proxy는 모두 중간 객체가 존재해 혼동되기 쉽다. 그러나 Adapter는 인터페이스를 바꾸는 것이 핵심이고 Proxy는 같은 인터페이스를 유지한 채 실제대상 접근을 대신하는 것이 핵심이다. Bridge와도 헷갈리기 쉬우나 Bridge는 추상화-구현 분리가 중심이고, 이 문항은 실제대상 참조와 위임 호출이 중심 단서이다.

다시 나오면 이렇게 판단

중간 객체가 실제대상을 내부에 참조하고 클라이언트와 같은 인터페이스로 요청을 대신 받아 위임하면 Proxy로 판단한다. 인터페이스를 바꾸면 Adapter, 추상화와 구현을 분리하면 Bridge, 여러 객체로 요청을 넘기면 Chain of responsibility로 판단된다.

암기 포인트

같은 인터페이스 유지 + 실제대상 참조 + 요청 위임이면 Proxy이다.

NoteBase

문 22. 다음 C 언어로 작성된 소스코드에서 두 개의 테스트 케이스, [Test case 1: (x=10, y=10, z=10), Test case 2: (x=8, y=7, z=5)]를 사용하여 테스트를 실행할 때, 이 두 개의 테스트 케이스로 달성되는 분기 커버리지와 조건 커버리지를 바르게 계산한 것은? (단, 단축 연산(short-circuit evaluation)은 수행하지 않고, 분기 및 조건에서 true 및 false 각각을 하나의 경우의 수로 계산한다)

```
if ((x > 5) || (y > 10) || (z < 8)) {
    z = x + y;
    z = z - 1;
}

if ((x > 8) && (z > 9)) {
    z = z + 1;
}
```

① 분기 커버리지(%): 50, 조건 커버리지(%): 70

② 분기 커버리지(%): 75, 조건 커버리지(%): 80

③ 분기 커버리지(%): 75, 조건 커버리지(%): 70

④ 분기 커버리지(%): 100, 조건 커버리지(%): 80

정답 3번

출제 포인트

이 문제는 두 개의 테스트 케이스를 실행했을 때 분기 커버리지와 조건 커버리지를 각각 몇 퍼센트까지 달성하는지를 계산하는 문제이다.

조건 정리

- 분기 수는 `if` 문 2개에서 true, false를 각각 세므로 총 4가지이다.
- 조건 수는 원자 조건 $x > 5$, $y > 10$, $z < 8$, $x > 8$, $z > 9$의 true, false를 각각 세므로 총 10가지이다.
- 단축 연산을 하지 않으므로 각 테스트 케이스에서 모든 원자 조건을 끝까지 평가한다.
- 첫 번째 `if`가 실행되면 z 값이 바뀌고, 두 번째 조건의 $z > 9$는 그 변경된 z로 판단한다.

NoteBase

계산 과정

Test case 1은 초깃값이 $(x, y, z) = (10, 10, 10)$이다.
- 첫 번째 `if`: $x > 5$=true, $y > 10$=false, $z < 8$=false이므로 분기 결과는 true이다.
- 블록 실행 후 $z = x + y = 20$, 다시 $z = z - 1 = 19$가 된다.
- 두 번째 `if`: $x > 8$=true, $z > 9$=true이므로 분기 결과는 true이다.

Test case 2는 초깃값이 $(x, y, z) = (8, 7, 5)$이다.
- 첫 번째 `if`: $x > 5$=true, $y > 10$=false, $z < 8$=true이므로 분기 결과는 true이다.
- 블록 실행 후 $z = x + y = 15$, 다시 $z = z - 1 = 14$가 된다.
- 두 번째 `if`: $x > 8$=false, $z > 9$=true이므로 분기 결과는 false이다.

분기 커버리지는 첫 번째 `if`에서 true만, 두 번째 `if`에서 true와 false를 모두 덮으므로 3/4 = 75%이다. 조건 커버리지는 $x > 5$는 true만, $y > 10$은 false만, $z < 8$은 true와 false 모두, $x > 8$은 true와 false 모두, $z > 9$는 true만 나온다. 따라서 덮인 경우의 수는 1 + 1 + 2 + 2 + 1 = 7개이므로 7/10 = 70%이다.

선지별 해설

① X. 조건 커버리지 70%는 맞지만 분기 커버리지가 50%라는 부분이 틀리다. 첫 번째 `if`의 false만 빠지고 나머지 세 분기는 모두 실행되므로 올바른 값은 75%이다.
② X. 분기 커버리지 75%는 맞지만 조건 커버리지가 80%라는 부분이 틀리다. 원자 조건 5개 중 true, false를 모두 만족한 것은 2개뿐이어서 올바른 값은 70%이다.
③ O. 분기 커버리지는 75%, 조건 커버리지는 70%로 계산되므로 보기의 수치가 모두 맞다.
④ X. 첫 번째 `if`의 false 분기가 한 번도 실행되지 않았으므로 분기 커버리지가 100%일 수 없다. 또한 $x > 5$, $y > 10$, $z > 9$는 true 또는 false 한쪽만 나왔으므로 조건 커버리지도 80%가 아니라 70%이다.

오답 포인트

가장 흔한 실수는 첫 번째 `if` 실행 뒤 바뀐 z를 놓치고 두 번째 조건의 $z > 9$를 초깃값으로 판단하는 것이다. 또 조건 커버리지를 계산할 때 원자 조건 5개 자체를 세지 않고, 각 조건의 true와 false를 따로 세어야 한다는 점을 빼먹기 쉽다.

다시 나오면 이렇게 판단

커버리지 계산 문제는 각 테스트 케이스를 코드 실행 순서대로 따라가며 변수 변경을 반영한 뒤, 분기는 `if` 결과의 true와 false, 조건은 각 원자 조건의 true와 false를 별도로 체크하는 방식으로 판단한다.

암기 포인트

분기 커버리지는 각 판단문의 true와 false를 세고, 조건 커버리지는 각 원자 조건의 true와 false를 센다는 점이 핵심이다.

NoteBase

문 23. CMMI(Capability Maturity Model Integration)는 소프트웨어 프로세스 개선 성숙도를 측정하기 위해 개발된 모델이다. 단계별 성숙도 측정 시 최적화(optimizing) 단계에 해당하는 프로세스 영역은?

① 제품 통합(product integration)

② 형상 관리(configuration management)

③ 협력 업체 관리(supplier agreement management)

④ 근본 원인 분석 및 해결(causal analysis & resolution)

정답 4번

출제 포인트

이 문제는 CMMI 단계별 성숙도에서 각 프로세스 영역이 어느 수준에 속하는지를 구분하는 문제이다.

핵심 개념

CMMI의 2단계는 프로젝트를 계획하고 통제하는 관리 기초를 갖추는 수준이고, 3단계는 조직 차원의 표준 프로세스를 정의하여 정착시키는 수준이며, 5단계는 결함의 원인을 제거하고 지속적으로 프로세스를 개선하는 최적화 수준이다.
제품 통합은 3단계의 엔지니어링 영역이고, 형상 관리는 2단계의 지원 영역이며, 협력 업체 관리는 2단계의 프로젝트 관리 영역이다. 근본 원인 분석 및 해결은 결함 원인을 분석해 재발을 막는 5단계의 대표 영역이다.

NoteBase

선지별 해설

① X
제품 통합은 여러 구성요소를 결합하여 완성된 제품을 만드는 3단계 프로세스 영역이다. 최적화 단계에 해당하지 않는다.
틀린 표현은 "최적화 단계"이다. 올바르게는 "제품 통합은 정의 단계의 프로세스 영역"이다.

② X
형상 관리는 소프트웨어 산출물의 버전과 변경을 통제하는 2단계 프로세스 영역이다. 반복 가능한 관리 체계를 만드는 단계에 속한다.
틀린 표현은 "최적화 단계"이다. 올바르게는 "형상 관리는 관리 단계의 프로세스 영역"이다.

③ X
협력 업체 관리는 외부 공급자와의 계약과 납품을 관리하는 2단계 프로세스 영역이다. 공급자 통제는 초기 관리 수준의 활동이지 최적화 수준의 활동이 아니다.
틀린 표현은 "최적화 단계"이다. 올바르게는 "협력 업체 관리는 관리 단계의 프로세스 영역"이다.

④ O
근본 원인 분석 및 해결은 결함이나 문제의 근본 원인을 찾아 재발을 방지하는 활동이므로, 지속적 개선을 수행하는 5단계 최적화 수준의 프로세스 영역이 맞다.

오답 포인트

제품 통합, 형상 관리, 협력 업체 관리처럼 자주 보이는 프로세스 영역 이름만 기억하고 단계까지 연결하지 못하면 오답이 된다. 이 문항은 "최적화"라는 단어를 보고 막연히 개선과 관련 있어 보이는 선지를 고르는 실수를 노린 문제이다.

다시 나오면 이렇게 판단

CMMI 문제는 먼저 단계의 성격을 잡으면 판단이 빨라진다. 2단계는 관리와 통제, 3단계는 표준화와 정의, 5단계는 원인 제거와 지속적 개선으로 구분하면 된다. 선지의 프로세스 영역이 결함 예방과 원인 분석에 직접 연결되면 5단계를 우선 떠올리면 된다.

암기 포인트

CMMI 5단계는 최적화이며, 대표 키워드는 근본 원인 분석과 지속적 개선이다.

NoteBase

문 24. 기능 점수 방법을 활용한 소프트웨어 개발 비용 산정의 수행 순서를 바르게 나열한 것은?

(가) 기능 점수 계산
(나) 규모 추정
(다) 노력 산정
(라) 기능 점수 보정
(마) 소요 기간 산정

① (가) → (라) → (나) → (다) → (마)
② (가) → (라) → (나) → (마) → (다)
③ (나) → (가) → (라) → (다) → (마)
④ (나) → (가) → (라) → (마) → (다)

정답 1번

출제 포인트

기능 점수 방법을 이용한 비용 산정에서 기능점수 산정, 기능점수 보정, 규모 추정, 노력 산정, 소요 기간 산정의 선후관계를 묻는 문제이다.

올바른 순서

1. 기능 점수 계산 - 입력, 출력, 조회, 파일 등의 기능을 세어 기본 기능점수를 계산한다.
2. 기능 점수 보정 - 일반 시스템 특성을 반영하여 기능점수를 보정한다.
3. 규모 추정 - 보정된 기능점수를 바탕으로 개발 규모를 추정한다.
4. 노력 산정 - 추정된 규모를 기준으로 필요한 인월을 산정한다.
5. 소요 기간 산정 - 산정된 노력과 인력 투입 계획을 바탕으로 일정을 계산한다.

왜 이 순서: 보정된 기능점수가 규모 추정의 입력값이고, 규모가 정해져야 노력과 기간을 산정할 수 있기 때문이다.

NoteBase

선지별 해설

① O - (가) → (라) → (나) → (다) → (마) 순서가 맞다. 기능점수를 먼저 계산하고 보정한 뒤 그 결과로 규모를 추정하고, 그 규모를 바탕으로 노력과 기간을 산정한다.
② X - 틀린 부분은 (마) → (다)이다. 올바르게는 (다) → (마)이다. 소요 기간 산정은 노력량이 먼저 계산된 다음에 가능하므로 기간이 노력보다 앞설 수 없다.
③ X - 틀린 부분은 (나) → (가)이다. 올바르게는 (가) → (라) → (나)이다. 규모 추정은 기능 점수 계산과 보정 결과를 입력으로 사용하므로 기능점수 산정보다 먼저 올 수 없다.
④ X - 틀린 부분은 (나)부터 시작하는 점과 (마) → (다) 순서이다. 올바르게는 기능 점수 계산 → 기능 점수 보정 → 규모 추정 → 노력 산정 → 소요 기간 산정이다. 규모 추정은 보정된 기능점수 뒤에 오고, 기간 산정은 노력 산정 뒤에 온다.

오답 포인트

규모 추정과 노력 산정, 소요 기간 산정을 서로 독립 단계로 보면 순서를 섞기 쉽다. 이 문항의 함정은 규모 추정이 기능점수 보정의 결과를 받아야 하고, 소요 기간 산정이 노력 산정 결과를 받아야 한다는 의존관계를 놓치게 만드는 데 있다.

다시 나오면 이렇게 판단

비용 산정 절차 문제는 앞 단계의 산출물이 다음 단계의 입력인지로 판단한다. 기능점수법에서는 기능점수 계산과 보정이 끝나야 규모가 나오고, 규모가 나와야 노력과 기간을 차례로 산정할 수 있다.

암기 포인트

기능점수법은 계산 → 보정 → 규모 → 노력 → 기간 순서이다.

NoteBase

문 25. 다음 소프트웨어 개발 일정에 대한 설명으로 옳지 않은 것은?

작업 이름	A	B	C	D	E	F	G	H	I	J	종료
선행 작업	–	A	A	C, E	B	E	F	D, G	H	I	J
수행 기간 (개월)	2	4	5	2	2	2	1	2	1	2	–

① D작업은 임계 경로에 포함되지 않는다.

② 개발 종료까지 최소 16개월이 소요된다.

③ 일정상 C작업의 여유 기간은 2개월이다.

④ H작업을 완료하기 위해서는 개발 시작 후 최소 11개월이 필요하다.

정답 4번

출제 포인트

이 문제는 작업의 선행 관계와 수행 기간으로 임계 경로, 전체 개발 기간, 각 작업의 여유 시간을 구하는 CPM 문항이다.

조건 정리

주요 선행 관계는 A→B, A→C, B→E, E→F, F→G, C와 E→D, D와 G→H, H→I, I→J, J→종료이다.
전진 계산으로 가장 이른 시작·완료 시각을 구하고, 후진 계산으로 가장 늦은 시작·완료 시각을 구하면 된다.

사용할 공식

어떤 작업의 가장 이른 시작 시각은 모든 선행 작업의 가장 이른 완료 시각 중 최댓값이다. 가장 이른 완료 시각은 가장 이른 시작 시각에 수행 기간을 더해 구한다.
여유 시간은 $LS - ES$ 또는 $LF - EF$로 구하며, 여유 시간이 0인 작업이 임계 경로에 포함된다.

계산 과정

전진 계산 결과는 A 0~2, B 2~6, C 2~7, E 6~8, D 8~10, F 8~10, G 10~11, H 11~13, I 13~14, J 14~16이다.
따라서 전체 개발 기간은 16개월이다.
임계 경로는 A-B-E-F-G-H-I-J이고 길이는 16개월이다.
후진 계산 결과 H의 최신 시작 시각은 11, D의 최신 완료 시각은 11이므로 D의 최신 시작 시각은 9이다. 따라서 D의 여유 시간은 1개월이고, C의 최신 시작 시각은 4이므로 C의 여유 시간은 2개월이다.

NoteBase

선지별 해설

① O
D작업의 가장 이른 시각은 8~10이고 최신 시각은 9~11이므로 여유 시간이 1개월이다. 여유 시간이 0이 아니므로 D작업은 임계 경로에 포함되지 않는다.

② O
종료까지 가장 긴 경로는 A-B-E-F-G-H-I-J이며 기간 합은 2+4+2+2+1+2+1+2=16이다. 따라서 개발 종료까지 최소 16개월이 소요된다.

③ O
C작업의 가장 이른 시작 시각은 2이고 가장 늦은 시작 시각은 4이다. 따라서 여유 시간은 2개월이다.

④ X
H작업의 선행 작업은 D와 G이므로 H는 둘 다 끝난 뒤에 시작할 수 있다. D는 10개월에, G는 11개월에 끝나므로 H의 가장 이른 시작 시각은 11개월이고 가장 이른 완료 시각은 13개월이다. 따라서 'H작업을 완료하기 위해서는 개발 시작 후 최소 11개월이 필요하다'는 틀렸고, 올바르게는 'H작업을 시작하기 위해서는 개발 시작 후 최소 11개월이 필요하다' 또는 'H작업을 완료하기 위해서는 최소 13개월이 필요하다'가 된다.

오답 포인트

4번은 H의 완료 시점을 묻는 문장을 H의 착수 가능 시점으로 바꿔 읽게 만드는 함정이다. 또한 둘 이상의 선행 작업이 있을 때는 선행 작업 완료 시각의 최댓값을 잡아야 하는데, 이 점을 놓치면 H와 D의 시각을 잘못 계산하게 된다.

다시 나오면 이렇게 판단

CPM 문항은 먼저 전진 계산으로 각 작업의 가장 이른 시작·완료 시각을 적고, 종료 시점에서 후진 계산으로 최신 시각을 구한 뒤, 여유 시간 0인 작업만 임계 경로로 묶어 선지와 대조하면 된다.

암기 포인트

전진 계산은 최조 시각, 후진 계산은 최신 시각, 여유 시간 0은 임계 경로라는 기준으로 정리된다.

NoteBase