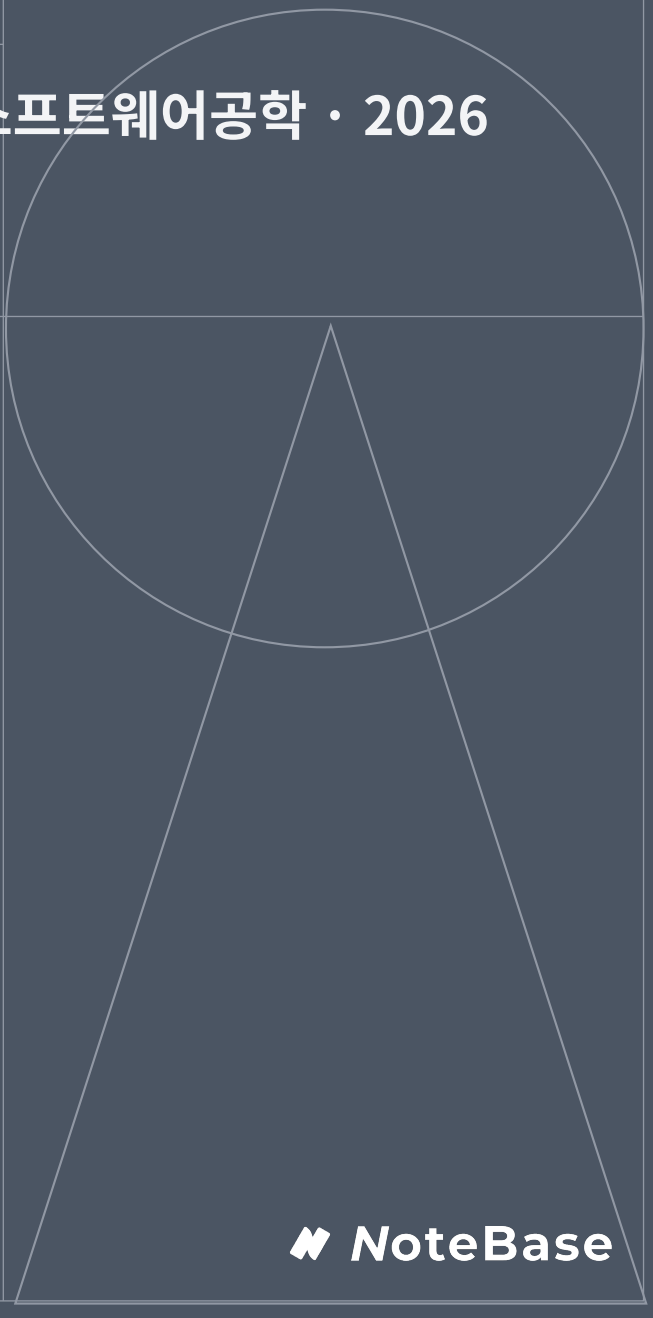
기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2026



NoteBase

기출OX 해설집

공무원/경찰/공채(경위) · 소프트웨어공학 · 2026

공무원/경찰/공채(경위) · 소프트웨어공학 · 2026

문 1. 소프트웨어의 특징으로 가장 적절하지 않은 것은?

① 닳아 없어지지 않는다.

② 제조하는 것이 아니라 개발한다.

③ 개발 완료 후 시험이 어렵고 수정할 수 없다.

④ 유형의 매체에 저장되지만 개념적이고 무형적이다.

정답 3번

간단한 해설

소프트웨어는 개발이 끝난 뒤에도 **시험과 수정이 가능**하다. 결함을 고치거나 환경·요구사항 변화에 대응하는 유지보수가 이루어지므로, 수정할 수 없다는 설명은 틀리다.

NoteBase

선지별 해설

소프트웨어는 프로그램과 관련 자료 등 논리적 구성으로 이루어진 무형의 산출물이다. 물리적 제품과 달리 사용에 따른 마모보다 설계·구현상의 결함, 환경 변화, 수정 과정에서의 오류가 품질에 영향을 준다.

① O: 소프트웨어 자체는 반복 사용한다고 기계 부품처럼 닳아 없어지지 않는다. 저장장치의 물리적 손상은 별개이며, 실행 중 드러나는 결함이나 환경 변화로 인한 장애를 물리적 마모와 혼동해서는 안 된다.

② O: 소프트웨어를 만드는 핵심은 요구분석·설계·구현·시험 등의 개발 활동이다. 완성된 프로그램의 복제는 물리적 제품을 한 개씩 제조하는 과정과 다르고, 주요 비용과 복잡성도 원본 소프트웨어를 개발·유지하는 데 발생한다.

③ X: 소프트웨어는 개발 완료 후에도 시험을 수행하고 결함을 수정할 수 있다. 유지보수에는 오류 수정, 새로운 실행환경에 대한 적응, 기능 개선 등이 포함된다. 복잡한 소프트웨어의 완전한 시험이나 변경 영향 파악이 어렵고 수정비용이 클 수는 있지만, 이것이 수정 불가능을 뜻하지는 않는다.

④ O: 프로그램은 디스크나 반도체 메모리 같은 유형의 매체에 저장되지만, 소프트웨어 자체는 명령·논리·데이터 등 개념적인 내용이다. 저장매체라는 물리적 대상과 그 안에 표현된 무형의 소프트웨어를 구별해야 한다.

NoteBase

문 2. 소프트웨어공학의 정의로 가장 적절하지 않은 것은?

① 유일한 제품이나 서비스를 만들기 위해 수행되어야 할 일시적인 활동 과정이다.

② 소프트웨어의 개발과 운용, 유지보수에 대하여 체계적이며, 훈련된 계량화할 수 있는 접근 방식을 적용한 것이다.

③ 프로그램을 설계하고 개발하며, 운용, 유지보수에 관련된 문서를 작성하는 데 필요한 과학적인 지식을 실용화한 것이다.

④ 효율적으로 작동하는 신뢰성 있는 소프트웨어를 경제적으로 획득하기 위해 적절한 공학적 원리를 수립하여 활용하는 것이다.

정답 1번

간단한 해설

고유한 제품·서비스를 만들기 위한 일시적인 활동은 **프로젝트**를 뜻한다. 소프트웨어공학은 소프트웨어의 개발·운용·유지보수에 체계적인 공학적 접근을 적용하는 분야이므로 두 개념은 구별된다.

선지별 해설

① X: 고유한 제품이나 서비스를 만들기 위한 일시적인 활동은 **프로젝트**의 정의이다. 프로젝트는 정해진 목표를 달성하면 종료되는 한시적 활동이며, 소프트웨어 개발도 하나의 프로젝트로 수행할 수 있다. 그러나 소프트웨어공학 자체는 특정 프로젝트가 아니라 소프트웨어의 개발·운용·유지보수에 공학적 원리와 방법을 적용하는 분야이다.

② O: 소프트웨어공학은 개발·운용·유지보수에 **체계적이고 규율 있으며 정량화 가능한 접근**을 적용한다. 절차와 기준에 따라 활동을 조직하고, 비용·일정·결함 등 측정 가능한 자료를 활용하여 품질과 과정을 관리한다는 의미이다.

③ O: 프로그램의 설계·개발뿐 아니라 운용·유지보수와 관련 문서 작성에도 과학적 지식을 실제로 적용한다는 설명이다. 소프트웨어공학의 대상은 코드 작성에만 한정되지 않으며, **문서와 생명주기 전반의 활동**을 포함한다.

④ O: 적절한 공학적 원리를 활용하여 신뢰성 있고 효율적인 소프트웨어를 경제적으로 확보하려는 설명이다. 기능 구현뿐 아니라 품질, 자원 사용, 개발·유지 비용을 함께 고려하므로 **기술적 품질과 경제성**을 동시에 추구한다.

NoteBase

문 3. 다음 소프트웨어 개발 프로세스 모델에 대한 설명으로 적절한 것을 있는 대로 모두 고른 것은?

가. 소프트웨어 개발 프로세스 모델은 프로젝트에 대한 전체적인 기본 골격을 세워준다.
나. 소프트웨어 개발 프로세스는 소프트웨어를 개발할 때 일을 수행하는 작은 단위인 작업의 집합을 이야기한다.
다. 소프트웨어 개발 프로세스 모델은 소프트웨어 개발 프로세스에서 소프트웨어를 어떻게 개발할 것인가에 대한 전체 흐름을 체계화한 개념으로 프로그램 설계라고 한다.
라. 소프트웨어 개발 프로세스 모델은 넓은 의미에서 절차나 과정뿐만 아니라 작업을 수행하는 데 필요한 방법과 도구를 포함하지만 인적 자원에 대해서는 포함하지 않는다.
마. 소프트웨어 개발 프로세스는 수많은 반복과 시행착오를 통해서 얻은 절차나 방법 등에 관한 지식을 같은 작업을 수행하는 다른 사람들에게 전달함으로써 시행착오를 줄이고 빠르게 적응할 수 있는 가이드 역할을 한다.

① 가, 나, 다
② 가, 나, 마
③ 나, 다, 라
④ 다, 라, 마

정답 2번

간단한 해설

개발 프로세스 모델은 **전체 개발 활동의 흐름과 관계**를 나타내며, 그 일부인 프로그램 설계와 다르다. 넓은 의미에서는 절차와 도구뿐 아니라 수행자의 역할과 인적 자원도 다룬다.

NoteBase

보기별 해설

가. O: 개발 프로세스 모델은 요구 분석, 설계, 구현, 시험 등의 활동을 어떤 순서와 관계로 수행할지 나타낸다. 프로젝트가 따라갈 **전체적인 활동의 골격**을 제시하므로 계획과 진행 관리의 기준이 된다.

나. O: 개발 프로세스는 소프트웨어를 만들기 위해 수행하는 활동과 세부 작업들의 집합으로 볼 수 있다. 다만 서로 무관한 할 일의 나열이 아니라 작업 사이의 순서·의존관계와 산출물 등이 연결된 과정이다.

다. X: 개발 전체의 흐름을 체계화한 개념은 **개발 프로세스 모델**이지 프로그램 설계가 아니다. 프로그램 설계는 구현할 소프트웨어의 구조, 모듈, 자료구조, 알고리즘 등을 구체화하는 개발 활동으로, 전체 개발 과정 중 일부에 해당한다.

라. X: 넓은 의미의 개발 프로세스에는 절차·방법·도구뿐 아니라 **작업을 담당할 인적 자원과 역할**도 포함된다. 누가 어떤 책임과 권한으로 작업하고 협력하는지가 정해져야 실제 개발 과정으로 실행할 수 있다.

마. O: 프로세스는 개발 경험에서 얻은 절차·방법·주의점을 공유할 수 있는 형태로 정리한다. 이를 통해 다른 수행자가 같은 시행착오를 반복할 가능성을 줄이고 작업 방식에 적응하도록 돕는 가이드가 된다.

적절한 보기는 가, 나, 마이므로 ②와 일치한다.

NoteBase

문 4. 다음 설명에 해당하는 소프트웨어 개발 프로세스 모델의 단계로 가장 적절한 것은?

Grady Booch, Ivar Jacobson, James Rumbaugh가 각자 제안한 객체지향 분석/설계 방법의 중요한 특징을 결합하여 제안하였다. 반복/점증적 특징을 갖는 프로세스로, 4개의 단계와 10개의 작업 흐름으로 구성된다. 단계는 시간의 경과에 따른 시스템의 진화를 묘사한다. 시스템 개발은 4개의 단계를 반복하며, 각 반복에서 여러 작업을 수행한다.

① 구현 → 완성 → 문제 식별

② 시작 → 구체화 → 구축 → 전이

③ 요구 분석 → 위험 분석 → 개발 → 평가

④ 요구 분석 → 구조 설계 → 모듈 설계 → 구현

정답 2번

간단한 해설

통합 프로세스는 개발의 범위와 방향을 정한 뒤 **핵심 아키텍처와 위험을 먼저 구체화**하고, 기능을 구축하여 사용자 환경으로 전이한다. 각 단계 안에서 분석·설계·구현·시험 등의 작업을 반복·점증적으로 수행한다.

올바른 순서

시작 → 구체화 → 구축 → 전이

시작(Inception): 부치·야콥슨·럼바우의 객체지향 방법을 통합하고 반복·점증 개발을 강조한 통합 프로세스의 첫 단계이다. 시스템의 목적, 범위, 핵심 요구와 사업성을 파악하여 개발을 추진할 방향을 정한다.

구체화(Elaboration): 주요 요구사항을 더 명확히 하고 핵심 아키텍처를 정립하며 큰 위험요인을 검증·완화한다. 본격적인 기능 구축에 앞서 구조와 위험을 안정시키는 단계이며, 필요하면 실제 구현을 통해 기술적 가능성을 확인한다.

구축(Construction): 정립된 아키텍처를 바탕으로 여러 반복을 거치며 기능을 구현·통합·시험하여 제품을 점진적으로 완성한다. 분석·설계·구현·시험 같은 작업은 각 단계에서 필요에 따라 반복되므로 네 단계를 폭포수식 작업 구분과 같게 보아서는 안 된다.

전이(Transition): 완성된 시스템을 사용자 환경에 배포하고 인수시험, 사용자 교육, 운영 전환과 피드백 반영을 수행한다. 개발된 기능을 실제 사용 가능한 제품으로 전달하고 안정적으로 활용하도록 하는 단계이다.

NoteBase

문 5. 애자일(agile) 모델에 대한 설명으로 가장 적절하지 않은 것은?

① 변경에 대한 대응보다 계획 준수가 더욱 중요하다.

② 문서화보다 동작하는 소프트웨어가 더욱 중요하다.

③ 프로세스나 도구보다 개인과 상호작용이 더욱 중요하다.

④ 계약 협상보다 고객과의 지속적인 협력이 더욱 중요하다.

정답 1번

간단한 해설

애자일은 최초 계획의 고수보다 **변화에 맞춘 대응**을 우선한다. 다만 계획 자체를 불필요하다고 보는 것은 아니다.

선지별 해설

① X: 애자일은 처음 정한 계획을 고수하는 것보다 **변경된 요구에 적응하는 것**에 더 높은 가치를 둔다. 고객의 피드백이나 환경 변화로 더 나은 방향이 확인되면 계획을 조정한다. 계획을 아예 세우지 않는다는 뜻은 아니다.

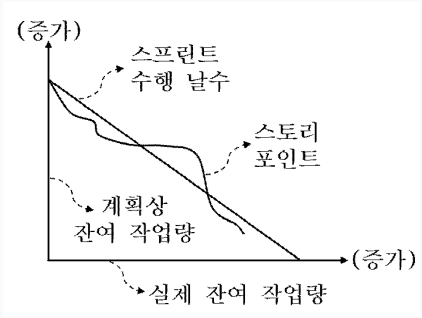
② O: 문서의 양이나 완성도 자체보다 사용자가 실제로 이용할 수 있는 기능을 구현하고 검증하는 것을 중시한다. 필요한 설계·운영 문서의 가치를 부정하는 것이 아니라, 문서 작성만으로 개발 성과를 대신하지 않는다는 의미이다.

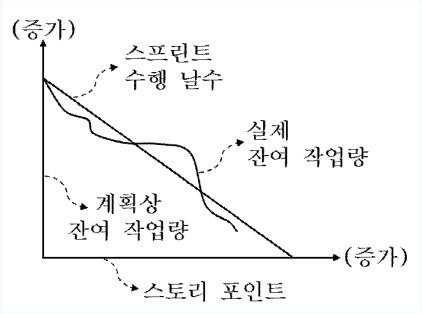
③ O: 정해진 절차나 도구의 사용 자체보다 구성원 사이의 의사소통과 협업을 우선한다. 도구와 절차는 팀의 문제 해결을 돕는 수단이며, 사람 간 상호작용을 대체할 수 없다는 취지이다.

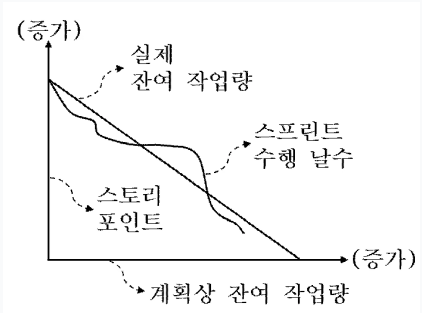
④ O: 계약 조건을 둘러싼 협상에만 의존하기보다 개발 과정에서 고객과 계속 소통하여 실제 요구와 피드백을 반영하는 것을 중시한다. 계약을 무시하자는 것이 아니라 고객과 함께 필요한 결과를 만들어 가는 협력에 더 높은 가치를 둔다는 뜻이다.

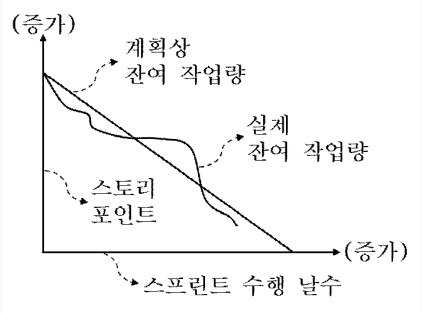
NoteBase

문 6. 스크럼(scrum) 방식에서 사용하는 소멸 차트(burndown chart)로 가장 적절한 것은?

① 

② 

③ 

④ 

정답 4번

간단한 해설

소멸 차트는 **시간에 따른 잔여 작업량**을 나타낸다. 수행 날수를 가로축, 스토리 포인트를 세로축에 두고 계획상 잔여량과 실제 잔여량을 비교한다.

NoteBase

도식 해설

소멸 차트는 시간이 흐르면서 남은 작업량이 어떻게 변하는지를 보여 준다. 이 그림에서는 가로축이 **스프린트 수행 날수**이고 세로축이 남은 작업량을 나타내는 **스토리 포인트**이다. 스토리 포인트는 작업 규모를 상대적으로 추정한 값이며, 완료한 작업량의 누적값을 나타내는 축이 아니다.

계획상 잔여 작업량은 시작 시점의 작업량에서 종료 시점의 0까지 일정하게 감소하는 직선이다. 실제 잔여 작업량은 작업 완료 상황에 따라 감소 속도가 달라지는 굽은 선이다. 실제 선이 같은 시점의 계획선보다 위에 있으면 계획보다 작업이 많이 남아 있고, 아래에 있으면 더 적게 남아 있다는 뜻이다. 실제 선의 수평 구간은 기록상 잔여 작업량이 줄지 않은 구간이다.

①은 가로축을 실제 잔여 작업량으로 두고 수행 날수를 직선에 붙였으므로 시간에 따른 잔여량을 나타내는 배치가 아니다.

②도 가로축을 스토리 포인트로 두고 수행 날수를 직선에 붙였다.

③은 가로축을 계획상 잔여 작업량으로 두고 실제 잔여량과 수행 날수의 선 표기도 바꾸었다.

④는 가로축에 수행 날수, 세로축에 스토리 포인트를 두고 직선과 굽은 선을 각각 계획상 잔여 작업량과 실제 잔여 작업량으로 표시하므로 소멸 차트의 의미에 맞는다.

NoteBase

문 7. DevOps 프로세스 단계와 사용 가능한 자동화 도구를 가장 적절하게 나열하지 않은 것은?

① 단계: 빌드, 도구: Puppet, Fortify

② 단계: 테스트, 도구: Junit, Selenium

③ 단계: 릴리즈, 도구: Microsoft Azure, AWS

④ 단계: 모니터링 및 피드백, 도구: Elastic, Splunk

정답 1번

간단한 해설

Puppet은 **인프라 구성 관리**, Fortify는 **애플리케이션 보안 검사**가 주된 역할이므로 둘을 빌드 도구로 묶은 연결이 가장 부적절하다. 빌드 과정에 연동할 수 있는 도구라고 해서 주된 기능까지 빌드인 것은 아니다.

선지별 해설

① X: 빌드는 소스 코드를 컴파일하고 의존성을 처리하여 실행·배포할 산출물을 만드는 단계이다. Puppet의 대표 기능은 **서버·인프라의 구성 관리와 자동화**이고, Fortify의 대표 기능은 **애플리케이션 보안 취약점 검사**이다. 둘을 빌드 단계의 대표 도구로 묶는 것은 적절하지 않다. 빌드에는 Maven·Gradle 같은 도구가 대표적으로 쓰인다. 보안 검사를 빌드 파이프라인에 연동할 수 있다는 사실은 Fortify의 주된 기능이 빌드라는 뜻이 아니다.

② O: 테스트 단계는 구현한 기능과 동작이 기대 결과를 만족하는지 확인한다. JUnit은 자바 계열 프로그램의 자동화된 단위 테스트 등에 사용하고, Selenium은 **웹 브라우저 동작을 자동화**하여 웹 애플리케이션의 기능·회귀 테스트 등에 사용한다.

③ O: 릴리즈 단계에서는 검증된 소프트웨어를 배포 가능한 형태로 제공하고 대상 환경에 전달하는 흐름을 관리한다. Microsoft Azure와 AWS는 단일 릴리즈 전용 도구라기보다 클라우드 플랫폼이지만, **릴리즈·배포 자동화를 지원하는 서비스**를 제공한다. Azure Pipelines의 릴리즈 파이프라인과 AWS CodeDeploy 등의 기능을 이 단계에 활용할 수 있다.

④ O: 모니터링 및 피드백 단계는 운영 중 발생한 오류·성능·사용 상태를 관찰하고 개선에 반영하는 단계이다. Elastic과 Splunk는 **로그와 운영 데이터의 수집·검색·분석** 등에 활용할 수 있어 장애 탐지, 원인 분석, 운영 결과의 피드백에 적합하다.

NoteBase

문 8. 다음 설명에 해당하는 SOLID 원칙으로 가장 적절한 것은?

◯ 하위 클래스를 만들 때, 상위 클래스의 기능을 교체하는 것이 아닌 기능을 유지하면서 확장할 수 있어야 한다.
◯ 자식 타입은 언제나 부모 타입을 대체할 수 있어야 한다. 이 원칙을 지키지 않으면 Refused Bequest라는 나쁜 냄새(bad smell)가 생긴다.
◯ 상위 클래스 객체를 하위 클래스 객체로 교체하여도 프로그램의 정확성, 수행하는 업무 등과 같은 프로그램 속성에 대한 변경이 없어야 한다.

① OCP(Open-Closed Principle)
② LSP(Liskov Substitution Principle)
③ ISP(Interface Segregation Principle)
④ DIP(Dependency Inversion Principle)

정답 2번

간단한 해설

부모 타입이 필요한 곳에 자식 타입을 넣어도 **기대한 계약과 올바른 동작이 유지**되어야 한다는 것은 리스코프 치환 원칙(LSP)이다. 단순히 기능을 추가하는지보다 하위 객체가 상위 객체를 안전하게 대체할 수 있는지가 핵심이다.

NoteBase

구성 요소별 해설

리스코프 치환 원칙(LSP)은 상위 타입을 사용하는 코드가 하위 타입의 객체로 바뀌어도 상위 타입에 대해 기대한 계약과 올바른 동작이 유지되어야 한다는 원칙이다. 제시된 세 설명은 모두 상속관계의 대체 가능성을 가리킨다.

첫 번째 설명: 상위 클래스의 기능을 유지한다는 것은 외부에서 기대하는 의미와 계약을 보존한다는 뜻이다. 하위 클래스는 기능을 확장하거나 메서드를 재정의할 수 있지만, 상위 타입을 사용하는 코드의 기대를 깨뜨려서는 안 된다. 구현 자체를 전혀 바꾸지 말라는 뜻은 아니다.

두 번째 설명: 부모 타입이 필요한 곳에 자식 타입을 넣어도 정상적으로 사용할 수 있어야 한다. 상속받은 기능을 하위 클래스가 거부하거나 지원하지 않아 호출 시 실패한다면, Refused Bequest라는 코드 냄새로 부적절한 상속관계를 의심할 수 있다. 명칭상의 상속만으로 행동의 대체 가능성이 확보되지는 않는다.

세 번째 설명: 하위 객체로 교체한 뒤에도 프로그램의 정확성과 원래 업무의 의미가 유지되어야 한다. 이를 위해 하위 타입은 상위 타입보다 더 강한 사전조건을 요구하거나 더 약한 사후조건만 제공하여 호출자의 계약을 깨뜨려서는 안 된다.

OCP는 기존 코드를 크게 수정하지 않고 기능을 확장할 수 있게 하는 원칙, ISP는 사용하지 않는 인터페이스에 의존하지 않도록 인터페이스를 분리하는 원칙, DIP는 구체 구현보다 추상화에 의존하게 하는 원칙이다. 제시문의 직접적인 기준은 상위·하위 타입 간의 치환 가능성이므로 LSP인 ②에 해당한다.

NoteBase

문 9. 다음 설명에 해당하는 주요 활동으로 가장 적절하지 않은 것은?

위험(risk)이란 프로젝트에 심각한 영향을 미쳐 프로젝트 성패에 직접적인 영향을 미칠 수 있는 사건을 의미한다. 프로젝트를 성공적으로 완수하기 위해 미래에 발생할 수 있는 불확실한 위험 요소를 미리 파악하고, 발생 가능성을 분석하여, 이를 해결하기 위한 전략을 수립해야 한다.

① 위험 식별

② 위험 분석

③ 위험 제거

④ 위험 감시

정답 3번

간단한 해설

위험관리는 위험을 식별·분석하고 대응계획을 세워 감시하는 활동이다. **모든 위험을 제거하는 것이 필수 단계는 아니며**, 상황에 따라 완화·전가·수용 등의 전략을 선택한다.

선지별 해설

지문은 미래의 불확실한 사건을 파악하고 평가하여 대응하는 위험관리를 설명한다. 주요 활동은 위험 식별, 분석, 대응계획 수립, 감시 등이며 모든 위험의 완전한 제거를 전제하지 않는다.

① O: 위험 식별은 일정 지연, 핵심 인력 이탈, 기술적 난점 등 프로젝트에 영향을 줄 수 있는 사건과 원인을 찾아내는 활동이다. 지문의 **'위험 요소를 미리 파악'**하는 단계에 해당한다.

② O: 위험 분석은 식별한 위험의 발생 가능성과 발생 시 영향을 평가하여 중요도와 대응 우선순위를 정하는 활동이다. 단순히 위험 목록을 만드는 식별과 달리 **가능성과 영향의 크기**를 판단한다.

③ X: 위험 제거는 위험관리 전체에 공통으로 요구되는 독립적 주요 활동으로 보기 어렵다. 위험에 따라 회피·완화·전가·수용 등의 **대응 전략을 계획하고 실행**해야 하며, 비용이나 실행 가능성 때문에 일부 위험을 수용할 수도 있다. 회피 전략으로 특정 위험의 원인을 없애는 경우는 있지만, 모든 위험을 제거해야 관리가 성립하는 것은 아니다.

④ O: 위험 감시는 기존 위험의 변화, 새로운 위험의 출현, 대응조치의 실행 상황과 효과를 계속 추적하는 활동이다. 최초 분석 결과를 고정해 두지 않고 프로젝트 진행 중 위험 수준과 대응계획을 갱신하도록 한다.

NoteBase

문 10. ISO/IEC 25019 사용자 품질 모델에 대한 특성으로 가장 적절하지 않은 것은?

① ISO/IEC 25019는 소프트웨어 자체에 대한 품질평가보다 사용자가 소프트웨어를 사용하는 동안의 경험에 대한 품질평가를 위한 것이다.

② 효율성(efficiency)은 유용성, 신뢰성, 쾌적성, 안정성 기반으로 설정된 목표와 제약조건에 대한 효과적인 달성 여부를 평가한다.

③ 맥락 포괄성(context coverage)은 다른 상황이나 환경에서 얼마나 잘 적응하고 대응하는지, 유연하게 확장이나 변경할 수 있는지에 대하여 평가한다.

④ 위험 회피성(freedom from risk)은 소프트웨어가 예측 가능한지, 소프트웨어 사용으로 인한 사용자의 건강과 안전, 경제적, 환경적 위험을 완화하는가에 대해 평가한다.

정답 2번

간단한 해설

효율성은 목표 달성에 쓴 **시간·노력·비용 등의 자원**을 평가한다. ②의 유용성·신뢰성·쾌적성 등의 요소는 **만족도**에 관한 설명이며 목표를 얼마나 정확하고 완전하게 달성했는지는 효과성에 해당한다.

출제 포인트

사용 품질을 평가할 때 효율성, 효과성, 만족도를 구분하는 문제이다.

핵심 개념

사용 품질은 사용자가 특정 상황에서 소프트웨어를 사용하여 얻는 결과와 경험의 품질이다.
효과성은 목표를 얼마나 정확하고 완전하게 달성했는지, 효율성은 그 성과에 비해 시간·노력·비용을 얼마나 사용했는지, 만족도는 사용자의 요구와 경험이 얼마나 충족되었는지를 평가한다.
예를 들어 같은 업무를 똑같이 정확하게 끝냈더라도 더 적은 시간과 노력으로 수행했다면 효율성이 더 높다.
문항에 제시된 맥락 포괄성 등은 ISO/IEC 25010:2011의 기존 사용 품질 분류에 해당한다. ISO/IEC 25019:2023에서는 맥락 포괄성을 사용 품질 특성에서 제외했다. 이 문항의 선지 판단은 제시된 기존 개념 간 구분에 따른다.

NoteBase

선지별 해설

① O
사용 품질 모델은 소프트웨어 자체의 속성뿐 아니라 사용자가 실제로 사용하는 상황에서 얻는 성과, 경험과 영향을 평가한다. 따라서 사용 중의 경험을 평가한다는 취지가 적절하다.

② X
'유용성, 신뢰성, 쾌적성, 안정성 기반'이라는 설명은 효율성의 판단 기준이 아니다. 기존 사용 품질 모델의 만족도는 유용성, 신뢰, 즐거움, 편안함과 관련된다. 또한 목표를 정확하고 완전하게 달성했는지는 효과성의 기준이다. 효율성은 '목표 달성의 정확성과 완전성에 비해 투입한 시간·노력·비용 등의 자원을 평가한다'로 고친다.

③ O
이 문항에 제시된 기존 사용 품질 개념에서 맥락 포괄성은 정해진 사용 상황뿐 아니라 처음에 예상하지 못한 상황에서도 효과성·효율성·만족도와 위험 완화가 유지되는지를 뜻한다. 선지의 적응과 유연성은 다른 사용 상황까지 품질을 유지한다는 의미에서 적절하다. 단순히 코드를 확장하거나 변경하기 쉽다는 뜻으로만 이해하지 않는다.

④ O
위험 회피성은 사용으로 생길 수 있는 경제적 손실, 건강·안전상의 피해, 환경적 피해를 얼마나 완화하는지 평가한다. 위험 완화를 중심으로 한 선지의 취지가 적절하다. 다만 '예측 가능한지'만으로 위험 회피성을 정의하지는 않으며, 의도한 대로 동작할 것이라는 사용자의 믿음은 신뢰와 관련된다.

오답 포인트

②의 '효과적인 달성 여부'라는 표현에만 주목하면 효율성도 목표 달성을 평가하므로 맞다고 생각하기 쉽다. 효율성에서는 목표 달성에 투입한 자원을 함께 보며, 유용성·신뢰·즐거움·편안함은 만족도와 구분한다.

다시 나오면 이렇게 판단

목표 달성의 정확성과 완전성을 말하면 효과성, 달성한 성과에 비해 사용한 시간·노력·비용을 말하면 효율성으로 판단한다. 사용자가 느끼는 유용성·신뢰·즐거움·편안함은 만족도, 사용에 따른 피해의 완화는 위험 회피성에 연결한다.

암기 포인트

효과성은 달성의 정확성과 완전성, 효율성은 성과 대비 자원, 만족도는 사용자의 요구와 경험이다.

문 11. 소프트웨어 품질 측정 척도에 대한 설명으로 가장 적절하지 않은 것은?

① MTTR(Mean Time To Repair)가 높을수록 정확성이 좋다.

② MTTF(Mean Time To Failure)가 낮을수록 가용성이 좋다.

③ MTTC(Mean Time To Change)가 낮을수록 유지보수성이 좋다.

④ MTBF(Mean Time Between Failure)가 높을수록 신뢰성이 좋다.

정답 1, 2번

간단한 해설

MTTR은 평균 수리시간이므로 **낮을수록 복구와 가용성에 유리**하며 정확성 향상을 뜻하지 않는다. MTTF는 평균 고장 전 작동시간이므로, 수리시간이 같을 때 **높을수록 가용성이 좋다**.

선지별 해설

고장까지 정상적으로 작동하는 시간은 길수록 좋고, 수리나 변경에 드는 시간은 짧을수록 좋다. 신뢰성은 고장 없이 작동하는 정도, 가용성은 필요할 때 사용할 수 있는 정도, 유지보수성은 고치거나 변경하기 쉬운 정도로 구별한다.

① X: MTTR은 고장을 수리하는 데 드는 평균 시간이다. 다른 조건이 같다면 낮을수록 복구가 빨라 유지보수성과 가용성에 유리하다. 높을수록 정확성이 좋다는 설명은 지표의 의미와 바람직한 방향을 모두 잘못 연결했다.

② X: MTTF는 고장이 발생하기까지의 평균 정상 작동 시간이다. 같은 수리시간이라면 MTTF가 길수록 가용성이 좋아진다. 단순한 정상가동·수리 반복 모형에서 가용성은 $A = \frac{MTTF}{MTTF + MTTR}$이므로, MTTR을 고정하면 MTTF가 낮아질수록 가용성도 낮아진다.

③ O: MTTC는 변경을 완료하는 데 드는 평균 시간이다. 같은 범위와 품질의 변경을 더 짧은 시간에 수행할 수 있으면 수정·개선이 쉬운 것이므로 유지보수성이 좋다고 볼 수 있다.

④ O: MTBF는 고장과 고장 사이의 평균 간격을 나타내며 신뢰성을 평가하는 지표로 사용된다. 동일한 측정 정의와 운영조건 아래에서 고장 발생 간격이 길수록 고장이 드물다는 뜻이므로 신뢰성이 좋다고 평가한다.

NoteBase

문 12. 다음 설명의 괄호(㉠, ㉡)에 들어갈 용어를 가장 적절하게 나열한 것은?

(㉠)의 주요 활동 중 수립한 (㉡)은/는 정형화한 절차로만 변경할 수 있는 명세 또는 결과물이다. 이는 소프트웨어 개발 과정에서 변경 요청을 공식적으로 검토하고 승인한 시점 이후 개발의 기초로 이용한다.

① ㉠: 유지보수, ㉡: 기준선

② ㉠: 유지보수, ㉡: 변경 제어

③ ㉠: 형상관리, ㉡: 기준선

④ ㉠: 형상관리, ㉡: 변경 제어

정답 3번

간단한 해설

공식 검토·승인을 거쳐 이후 개발의 기초가 되는 산출물은 **기준선**이다. 기준선의 설정과 변경 통제는 **형상관리**에 속하며, 기준선은 정해진 절차로 변경할 수 있다.

빈칸별 해설

㉠ 형상관리: 소프트웨어 산출물의 구성과 버전을 식별하고, 변경을 통제하며, 상태와 이력을 기록하여 일관성을 유지하는 활동이다. 요구사항 명세서·설계서·코드 등의 **기준선을 설정하고 그 변경을 관리**하므로 지문의 상위 활동에 해당한다.

㉡ 기준선: 공식적으로 검토·승인되어 이후 개발의 기초로 삼는 명세나 산출물의 확정 상태이다. 기준선이 설정되면 누구나 임의로 내용을 바꾸는 것이 아니라 **정해진 변경 제어 절차를 통해서만** 변경한다. 변경할 수 없다는 뜻이 아니라 변경을 통제한다는 뜻이다.

유지보수와의 구분: 유지보수는 소프트웨어를 인도한 뒤 결함 수정, 환경 변화에 대한 적응, 기능·성능 개선 등을 수행하는 활동이다. 유지보수 과정에서도 형상관리를 적용하지만, 기준선 설정과 버전·변경 통제를 포괄하는 활동의 명칭은 형상관리이다.

변경 제어와의 구분: 변경 제어는 변경 요청의 영향과 필요성을 검토하고 승인·반려하며 승인된 변경을 반영하도록 관리하는 절차이다. 지문이 묻는 **변경 대상인 명세·결과물**은 기준선이고, 그 변경을 허용하는 절차가 변경 제어이다.

따라서 ㉠은 형상관리, ㉡은 기준선이며 이를 바르게 나열한 것은 ③이다.

NoteBase

문 13. 다음 의존 관계와 소요 기간을 기반으로 작업별 여유 시간(slack time) 계산 요소에 대한 설명으로 가장 적절하지 않은 것은?

작업명	시작	가	나	다	라	마	종료
선행작업	-	시작	시작	가, 나	다, 마	나	라
소요시간	2주	4주	1주	2주	1주	7주	1주

① '가' 작업의 EST는 '나' 작업의 LST와 같다.

② '다' 작업의 EFT는 '가' 작업의 LFT와 같다.

③ '라' 작업의 EST는 '마' 작업의 LST와 같다.

④ '마' 작업의 EFT는 '다' 작업의 LFT와 같다.

정답 3번

간단한 해설

라는 다와 마가 모두 끝나야 하므로 EST가 **10주**이다. 반면 마는 라의 시작을 늦추지 않으려면 늦어도 **3주**에 시작해야 하므로 두 값은 다르다.

NoteBase

선지별 해설

EST·EFT는 가장 빠른 시작·완료 시점, LST·LFT는 전체 완료를 늦추지 않는 가장 늦은 시작·완료 시점이다. 시점 0에서 시작하고 표의 '시작' 2주와 '종료' 1주도 작업기간에 포함한다. 순방향에서는 선행 작업의 EFT 중 최댓값을 EST로, 역방향에서는 후속 작업의 LST 중 최솟값을 LFT로 삼는다.

순방향 계산의 EST/EFT는 시작 0/2, 가 2/6, 나 2/3, 다 6/8, 마 3/10, 라 10/11, 종료 11/12주이다. 다는 가·나를, 라는 다·마를 모두 기다려야 하므로 각각 6주와 10주에 시작한다. 최소 완료시점 12주에서 역산한 LST/LFT는 종료 11/12, 라 10/11, 마 3/10, 다 8/10, 가 4/8, 나 2/3, 시작 0/2주이다. 나는 후속 작업 다와 마 중 더 이른 마의 LST인 3주까지 끝나야 한다. 여유 시간은 LST − EST이며, 가·다는 각각 2주, 나·마·라는 0주이다.

① O: 가의 EST는 시작 작업이 끝나는 2주이다. 나의 LST는 마의 최종 시작시점 3주에서 나의 기간 1주를 뺀 2주이므로 두 값이 같다.

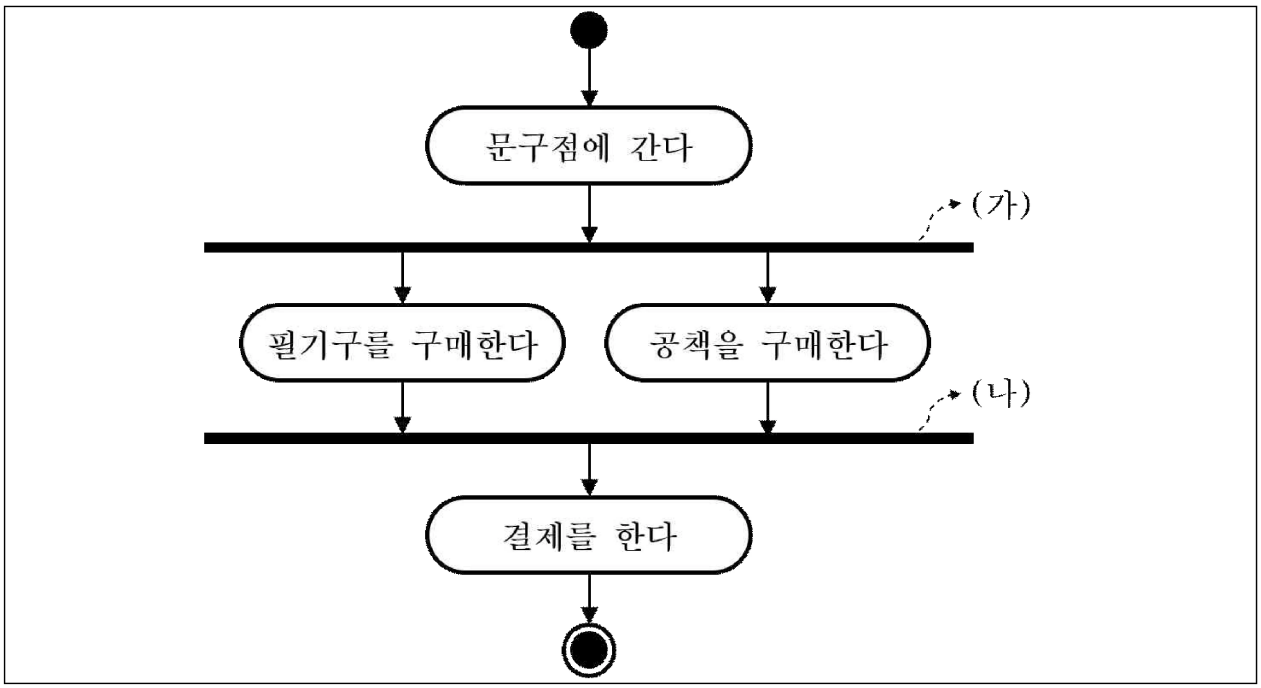
② O: 다의 EFT는 선행 작업이 모두 끝나는 6주에 기간 2주를 더한 8주이다. 가의 LFT는 후속 작업 다의 LST인 8주이므로 두 값이 같다.

③ X: 라의 EST는 다의 EFT 8주와 마의 EFT 10주 중 늦은 **10주**이다. 마의 LST는 라의 LST 10주에서 기간 7주를 뺀 **3주**이므로 서로 같지 않다.

④ O: 마의 EFT는 나가 끝나는 3주에 기간 7주를 더한 10주이다. 다의 LFT는 후속 작업 라의 LST인 10주이므로 두 값이 같다.

NoteBase

문 14. 다음 활동(activity) 다이어그램에서 (가)와 (나)에 해당하는 노드를 가장 적절하게 나열한 것은?



① (가): 조인, (나): 병합

② (가): 조인, (나): 포크

③ (가): 포크, (나): 병합

④ (가): 포크, (나): 조인

정답 4번

간단한 해설

한 흐름을 병렬 활동으로 나누는 (가)는 **포크**, 두 활동의 완료를 기다려 다시 합치는 (나)는 **조인**이다.

NoteBase

도식 해설

(가)는 포크(fork) 노드이다. '문구점에 간다'에서 들어오는 한 흐름이 굵은 막대를 지나 '필기구를 구매한다'와 '공책을 구매한다'의 두 흐름으로 나뉜다. 포크는 한 제어 흐름을 **병렬로 진행 가능한 여러 흐름**으로 분기한다. 두 활동 중 하나만 선택하는 조건 분기와 다르다.

(나)는 조인(join) 노드이다. 두 구매 활동에서 오는 흐름이 굵은 막대로 모인 다음 '결제를 한다'로 이어진다. 별도의 조인 조건이 없는 이 도식에서 조인은 두 구매 활동의 완료를 기다려 **병렬 흐름을 동기화**한 뒤 다음 활동을 진행하게 한다.

병합(merge) 노드는 대체적인 경로들을 다시 한 흐름으로 모으지만, 여러 병렬 활동이 모두 끝나기를 기다리는 동기화 기능은 없다. 일반적으로 마름모로 표시하므로 그림의 (나)에 해당하지 않는다.

따라서 (가)는 포크, (나)는 조인으로 연결하는 ④에 해당한다.

NoteBase

문 15. 소프트웨어 개발 비용 계산을 위해 사용하는 델파이 (Delphi) 기법에 대한 설명으로 가장 적절하지 않은 것은?

① 과거 유사 프로젝트 수행 경험을 바탕으로 개발 비용을 추정한다.

② 전문가의 주관적인 판단으로 실제 개발 비용과 편차가 클 수 있다.

③ 전문가 의견을 조정자가 수집하고 이를 조정하는 상향식 방법이다.

④ 전문가들의 편견이나 분위기에 끌려가지 않도록 조정자가 필요하다.

정답 3번

간단한 해설

델파이 기법은 전문가들의 경험과 판단을 반복적으로 수렴하는 **하향식 비용 추정 방법**이다. 세부 작업별 비용을 구해 합산하는 상향식 방법과 구별된다.

선지별 해설

① O: 델파이 기법은 여러 전문가의 경험과 판단을 이용하는 비용 추정 방법이다. 전문가들은 과거 유사 프로젝트의 규모·난이도·인력 투입 경험 등을 현재 프로젝트의 조건과 비교하여 비용을 예상한다. 과거 경험을 활용하되 단순히 과거 비용을 그대로 복사하는 것은 아니다.

② O: 전문가의 경험을 활용하더라도 추정에는 **주관적 판단**이 포함된다. 전문가가 가진 정보나 경험이 부족하거나 새로운 프로젝트의 특성을 잘못 평가하면 실제 개발 비용과 큰 차이가 생길 수 있다. 반복적인 의견 수렴은 이런 위험을 줄이지만 정확한 비용을 보장하지는 않는다.

③ X: 전문가의 추정 의견을 조정자가 수집·정리하고 피드백한다는 설명은 맞지만, 비용 산정 기법의 분류에서 델파이 기법은 **하향식 방법**이다. 프로젝트 전체에 대한 전문가의 판단으로 총비용을 추정하는 방식이며, 개별 작업이나 모듈의 비용을 각각 산정한 뒤 합산하는 상향식 방법과 구별된다. 여러 전문가의 의견을 모은다는 사실을 세부 작업별 비용의 합산과 혼동해서는 안 된다.

④ O: 조정자는 전문가들이 독립적으로 제시한 추정치를 수집하고 요약한 결과를 다시 제공하여 재추정을 돕는다. 익명성과 통제된 피드백을 유지하면 특정 전문가의 권위나 집단 분위기에 끌리는 영향을 줄일 수 있다. 조정자는 자신의 비용 추정치를 강요하는 사람이 아니라 **의견 수렴과 피드백을 관리하는 사람**이다.

NoteBase

문 16. 다음에 대한 MM(Man Month)으로 가장 적절한 것은?

◯ 프로젝트 복잡도 계수는 1.3이다.
◯ 유지보수 인력은 전체의 20%를 추가한다.
◯ 개발자 1인당 평균 생산성은 800 LOC/MM이다.
◯ 소프트웨어 예상 LOC(Line Of Code)는 120,000이다.

① 39
② 162
③ 195
④ 234

정답 4번

간단한 해설

기본 공수 150MM에 복잡도 계수 1.3을 적용하면 195MM이다. 여기에 **유지보수 공수 20%**인 39MM을 더하면 총 234MM이다.

계산 과정

MM은 한 사람이 한 달 동안 투입하는 작업량을 단위로 표현한 공수이다. 예상 코드량을 1MM당 생산성으로 나누어 기본 공수를 구한 뒤, 복잡도와 추가 유지보수 인력을 반영한다.

기본 개발 공수는 $\frac{120{,}000LOC}{800LOC/MM} = 150MM$이다. 코드량의 단위 LOC가 약분되어 공수 단위 MM이 남는다.

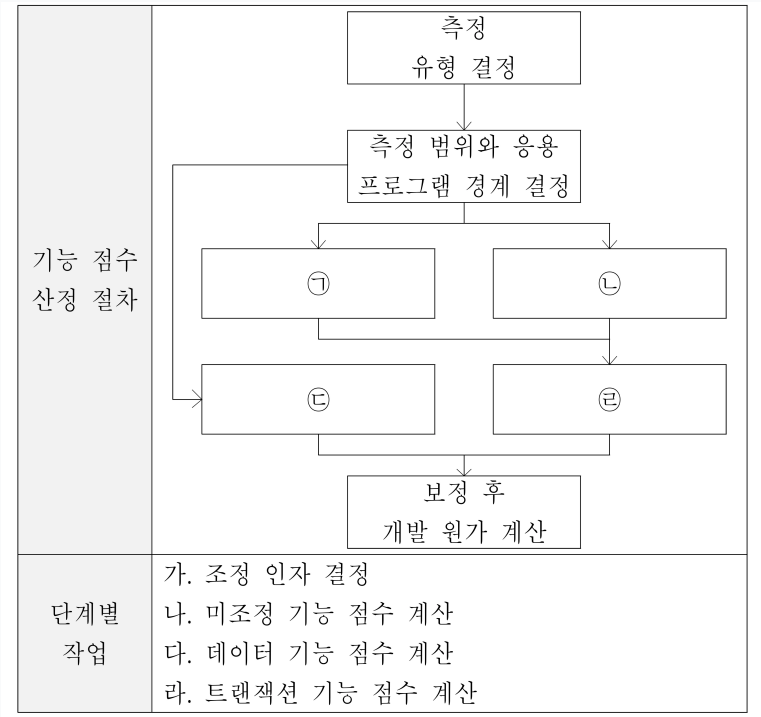
복잡도 계수 1.3을 적용하면 개발 공수는 $150 \times 1.3 = 195MM$이다. 이 계수는 기본 작업량을 30% 늘리는 역할을 한다.

유지보수 인력 20% 추가는 산정된 개발 공수에 20%를 더하는 것으로 적용한다. 추가 공수는 $195 \times 0.20 = 39MM$이고, 총공수는 $195 + 39 = 234MM$이다.

따라서 총 234MM으로 ④에 해당한다. 이는 총투입 공수이며, 반드시 234명이 필요하거나 실제 개발기간이 234개월이라는 뜻은 아니다.

NoteBase

문 17. 다음 기능 점수 산정 절차에 들어갈 단계를 가장 적절하게 나열한 것은?



① ㉠: 나, ㉡: 가, ㉢: 라, ㉣: 다

② ㉠: 나, ㉡: 라, ㉢: 가, ㉣: 다

③ ㉠: 다, ㉡: 라, ㉢: 가, ㉣: 나

④ ㉠: 다, ㉡: 가, ㉢: 라, ㉣: 나

정답 3번

간단한 해설

데이터 기능과 트랜잭션 기능의 점수를 합해야 **미조정 기능 점수**가 된다. 조정 인자는 별도로 정하여 마지막 계산에 반영하므로, 그림의 두 기능 점수 산정 가지와 조정 인자 가지를 구분해야 한다.

NoteBase

올바른 순서

측정 유형 결정 → 측정 범위·응용 프로그램 경계 결정 → (다·라 → 나, 가는 별도 병행) → 보정 후 개발 원가 계산

측정 유형 결정: 신규 개발, 개선 개발, 기존 응용 프로그램 등 어떤 대상을 측정하는지 정한다. 측정 목적과 유형을 먼저 확정해야 이후에 포함할 기능을 일관되게 판단할 수 있다.

측정 범위·응용 프로그램 경계 결정: 측정에 포함할 영역과 응용 프로그램의 안팎을 구분한다. 이 **경계가 데이터와 처리 기능의 분류 기준**이 되므로 기능 점수 계산보다 먼저 정한다.

다. 데이터 기능 점수 계산: 왼쪽 위 ㉠에 해당한다. 응용 프로그램이 유지하는 내부 논리 파일과 외부에서 유지되지만 참조하는 외부 인터페이스 파일 등 데이터 기능을 식별하고 복잡도에 따른 점수를 계산한다.

라. 트랜잭션 기능 점수 계산: 오른쪽 위 ㉡에 해당한다. 외부 입력·외부 출력·외부 조회 등 사용자가 인식하는 처리 기능을 식별하고 복잡도에 따른 점수를 계산한다. 그림의 ㉠과 ㉡은 **서로를 거치는 직렬 단계가 아니라** 두 종류의 기능을 나누어 산정하는 가지이다.

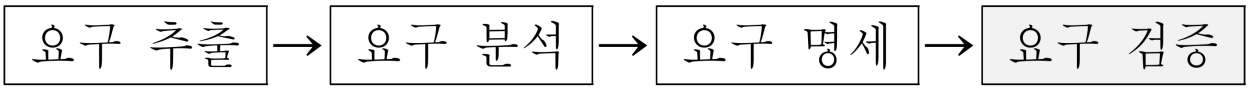
나. 미조정 기능 점수 계산: 오른쪽 아래 ㉣에 해당한다. 데이터 기능과 트랜잭션 기능의 점수를 합하여 조정 전 기능 규모를 구한다. 그림에서 두 위쪽 상자의 연결선이 합쳐지는 이유는 **두 종류의 점수가 모두 필요**하기 때문이다.

가. 조정 인자 결정: 왼쪽 아래 ㉢에 해당한다. 기능 점수의 합계와는 별도로 개발 대상의 특성을 평가하여 보정에 필요한 인자를 정한다. 그림에서는 측정 범위·경계 결정에서 이 단계로 직접 이어지므로, 미조정 기능 점수가 나온 뒤에만 시작하는 직렬 단계로 해석하지 않는다.

보정 후 개발 원가 계산: 미조정 기능 점수와 조정 인자의 결과를 결합하고, 적용하는 원가 산정 기준에 따라 개발 비용을 계산한다. 따라서 기능 규모를 구한 결과와 보정에 필요한 특성이 이 마지막 단계의 입력이 된다.

NoteBase

문 18. 다음 요구 공학 프로세스의 요구 검증 단계에서 검증하는 항목에 대한 설명으로 가장 적절하지 않은 것은?



① 검증 가능성(verifiability)은 모든 요구사항이 구현 가능한지 검증해야 한다.

② 일관성(consistency)은 모든 요구사항은 모순되지 않고 하나의 해석만 가능해야 한다.

③ 완전성(completeness)은 사용자의 모든 요구사항을 포함하고 있는지를 확인해야 한다.

④ 추적 가능성(traceability)은 설계문서 및 기능과 요구사항의 관계가 추적 가능해야 한다.

정답 1번

간단한 해설

검증 가능성은 구현된 시스템이 요구사항을 **충족하는지 시험·분석 등으로 확인할 수 있는 성질**이다. 요구사항을 구현할 수 있는지를 따지는 것은 **실현 가능성**에 해당한다.

선지별 해설

① X: 검증 가능성은 요구사항의 충족 여부를 시험·분석·검사 등으로 확인할 수 있어야 한다는 뜻이다. 예를 들어 '빠르게 응답해야 한다'보다 '응답시간이 2초 이내여야 한다'가 충족 여부를 확인하기 쉽다. 선지의 '구현 가능한지'는 기술·비용·일정 등의 제약 아래 실제로 구현할 수 있는지를 따지는 **실현 가능성**의 설명이다.

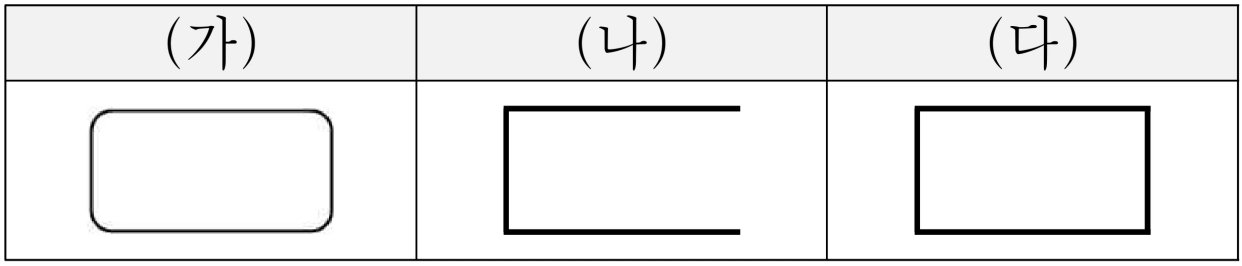
② O: 요구사항은 서로 모순되지 않아야 하며, 그 의미도 명확하게 해석될 수 있어야 한다. 다만 일관성의 핵심은 **요구사항 사이에 모순이 없는 것**이고, 선지의 '하나의 해석만 가능'은 엄밀히는 **비모호성**의 설명이다. 이 선지는 두 품질 조건을 함께 서술하고 있으므로, 하나의 해석만 가능하다는 것을 일관성만의 정의로 외우지 않도록 구분해야 한다.

③ O: 완전성은 시스템에 필요한 사용자의 요구사항이 **빠짐없이 포함**되어 있는지를 뜻한다. 필요한 기능이나 성능 조건 등이 누락되어 있다면, 기록된 요구사항끼리 모순이 없더라도 완전하다고 할 수 없다.

④ O: 추적 가능성은 요구사항과 설계문서·구현 기능 등 관련 산출물의 **대응 관계를 따라갈 수 있는 것**이다. 어떤 요구사항이 어느 설계와 기능에 반영되었는지 확인할 수 있어야 하며, 이 관계는 요구사항 변경 시 영향을 받는 부분을 찾는 데도 쓰인다.

NoteBase

문 19. 다음 구조적 분석기법의 자료 흐름도에서 사용하는 기호에 대한 의미를 가장 적절하게 나열한 것은?



① (가): 프로세스, (나): 외부엔터티, (다): 데이터저장소

② (가): 프로세스, (나): 데이터저장소, (다): 외부엔터티

③ (가): 데이터저장소, (나): 외부엔터티, (다): 프로세스

④ (가): 외부엔터티, (나): 데이터저장소, (다): 프로세스

정답 2번

간단한 해설

둥근 모서리는 **데이터 처리**, 한쪽이 열린 사각형은 **데이터 보관**, 닫힌 사각형은 시스템 밖의 데이터 제공자·수신자를 나타낸다.

도식 해설

자료 흐름도는 데이터가 어디에서 들어와 어떤 처리를 거쳐 어디로 전달·저장되는지를 나타낸다. 그림의 기호는 모서리와 변의 닫힘 여부로 구별할 수 있다.

(가)의 모서리가 둥근 직사각형은 프로세스이다. 입력 데이터를 가공하거나 변환하여 출력 데이터를 만들어 내는 처리 기능을 나타낸다. 단순히 데이터를 보관하는 장소와 구별된다.

(나)의 한쪽 변이 열린 직사각형은 데이터저장소이다. 처리 과정에서 참조하거나 이후 사용하기 위해 데이터를 보관하는 곳을 나타낸다. 파일이나 데이터베이스처럼 저장된 데이터의 집합에 해당한다.

(다)의 닫힌 직사각형은 외부엔터티이다. 분석 대상 시스템의 경계 밖에서 데이터를 제공하거나 결과를 받는 사람·조직·다른 시스템을 나타낸다. 시스템 내부에서 데이터를 변환하는 프로세스가 아니다.

따라서 (가) 프로세스, (나) 데이터저장소, (다) 외부엔터티로 연결하는 ②에 해당한다.

NoteBase

문 20. 구조적 분석기법 요소에 대한 설명으로 가장 적절하지 않은 것은?

① 자료사전은 자료 흐름도에 존재하는 데이터에 대한 자세한 설명을 기록한 것이다.

② 배경도는 하나의 프로세스로 구성되며, 개발하게 될 시스템의 명칭을 부여한다.

③ 소단위 명세서는 자료 흐름도에서 프로세스의 실제 알고리즘 형태를 작성한다.

④ E-R 다이어그램은 사용사례에 대하여 도식화하여 액터에 대한 설명을 작성한다.

정답 4번

간단한 해설

E-R 다이어그램은 **개체·속성·개체 사이의 관계**를 나타내는 데이터 모델이다. 액터와 사용사례를 표현하는 것은 유스케이스 다이어그램이므로 서로 구별해야 한다.

선지별 해설

구조적 분석에서는 자료 흐름도로 처리와 데이터의 이동을 나타내고, 자료사전과 소단위 명세서로 그 의미와 처리 논리를 구체화한다. 데이터 구조를 표현하는 E-R 모델과 사용자 관점의 기능을 표현하는 유스케이스 모델은 목적이 다르다.

① O: 자료사전은 자료 흐름도에 나타나는 데이터의 이름, 의미, 구성, 자료형 등 상세한 내용을 정의한다. 데이터 흐름이나 저장소에 쓰인 같은 이름을 일관된 뜻으로 이해하도록 돕는다.

② O: 배경도는 시스템 전체를 하나의 프로세스로 표현하고, 외부 개체와 시스템 사이의 데이터 흐름을 보여 주는 최상위 도식이다. 이 단일 프로세스에는 개발 대상 시스템의 명칭을 붙이며, 내부의 세부 처리과정은 하위 수준에서 분해한다.

③ O: 소단위 명세서는 더 이상 분해하지 않는 기본 프로세스가 입력을 어떻게 처리하여 출력을 만드는지 구체적으로 기술한다. 구조적 문장, 의사코드, 의사결정표 등으로 처리 논리나 알고리즘을 표현한다. 반드시 실제 프로그래밍 언어로 구현한 코드를 뜻하는 것은 아니다.

④ X: E-R 다이어그램은 고객·주문 같은 개체, 개체의 속성, 개체 간 관계를 표현한다. 시스템과 상호작용하는 액터와 그 액터가 이용하는 기능인 사용사례를 나타내는 것은 유스케이스 다이어그램이다. 따라서 선지는 데이터 모델과 사용사례 모델을 혼동했다.

NoteBase

문 21. 다음 중 비기능적 요구사항만 있는 대로 모두 고른 것은?

가. 보안 요구사항을 만족해야 한다.
나. 가용성을 최소 99% 이상 보장한다.
다. 파일 열기는 3초 이내에 완수해야 한다.
라. 사용자 입력값은 1부터 99까지 만으로 제한한다.
마. 텍스트(txt) 형식으로 문서를 저장할 수 있어야 한다.
바. 시스템 장애로 인한 서비스 불가 시간이 연 10시간을 넘지 않아야 한다.

① 가, 나, 다, 라
② 가, 나, 다, 바
③ 나, 다, 마, 바
④ 다, 라, 마, 바

정답 2번

간단한 해설

비기능적 요구사항은 **보안·가용성·응답시간처럼 서비스의 품질 수준**을 정한다. 입력값의 허용 범위와 특정 형식으로 저장하는 기능은 시스템의 구체적인 동작을 정하는 기능적 요구사항이다.

NoteBase

보기별 해설

기능적 요구사항은 시스템이 수행할 기능이나 입력에 대한 처리 규칙을 정한다. 비기능적 요구사항은 기능을 제공할 때 충족해야 하는 품질 수준이나 운영상 제약을 정한다. 숫자나 제한 조건이 있다는 사실만으로 비기능적 요구사항이 되는 것은 아니다.

가. 비기능적 요구사항: 시스템이 갖추어야 할 **보안 품질**을 요구한다. 구체적인 로그인 처리 절차 같은 기능을 지정하는 문장이 아니라 보안 요구를 충족해야 한다는 품질 조건이다.

나. 비기능적 요구사항: 시스템을 사용할 수 있는 정도인 **가용성의 하한을 99%**로 정한다. 어떤 기능을 수행하는지가 아니라 서비스를 얼마나 안정적으로 이용할 수 있어야 하는지를 규정한다.

다. 비기능적 요구사항: 파일 열기라는 기능 자체보다 그 기능을 **3초 이내에 완료해야 한다는 응답시간**을 요구한다. 이는 성능에 관한 측정 가능한 품질 조건이다.

라. 기능적 요구사항: 사용자가 입력할 수 있는 값의 범위를 1부터 99까지로 정한 **입력값 검증 규칙**이다. 시스템이 어떤 입력을 유효하게 받아들이는지를 지정하므로 기능의 동작 명세에 해당한다.

마. 기능적 요구사항: 문서를 텍스트(txt) 형식으로 저장하는 **구체적인 기능**을 제공해야 한다는 요구이다. 저장 속도나 신뢰성 등의 품질 수준을 정한 것이 아니다.

바. 비기능적 요구사항: 장애 때문에 서비스를 이용하지 못하는 시간을 연 10시간 이하로 제한한다. 이는 **서비스 가용성과 신뢰성에 관한 운영 품질**을 정한 것이며, 구체적인 장애 처리 기능을 명세한 문장이 아니다.

비기능적 요구사항은 가, 나, 다, 바이므로 ②의 조합과 일치한다.

NoteBase

문 22. 다음 밑줄 친 자바 소스 코드를 다형성(polymorphism)을 위한 메서드 오버로딩(method overloading)과 메서드 오버라이딩(overriding)에 따라 가장 적절하게 짝 지은 것은?

```
class Shape{
...(중략)...
    ㉠public int area(int len) { ...(중략)... }
    ㉡public int area(int width, int height) { ...(중략)... }
...(중략)...
}
class Circle extends Shape{
...(중략)...
    ㉢public int area(int radius) { ...(중략)... }
...(중략)...
}
```

① 오버로딩: ㉠, ㉡, 오버라이딩: ㉠, ㉢

② 오버로딩: ㉠, ㉡, 오버라이딩: ㉡, ㉢

③ 오버로딩: ㉠, ㉢, 오버라이딩: ㉡, ㉢

④ 오버로딩: ㉠, ㉢, 오버라이딩: ㉠, ㉡

정답 1번

간단한 해설

㉠과 ㉡은 같은 메서드 이름에 **매개변수 개수가 달라 오버로딩**이다. ㉢은 부모의 ㉠과 같은 시그니처를 자식 클래스에서 재정의하므로 오버라이딩이다.

NoteBase

개념 구분

㉠과 ㉡은 Shape 안에서 같은 이름 area를 사용하지만 매개변수 목록이 다르다. ㉠은 int 한 개, ㉡은 int 두 개를 받으므로 **오버로딩** 관계이다. 호출 시 인수의 개수와 타입 등에 따라 사용할 메서드가 구별된다.

㉢은 Shape를 상속한 Circle에 선언된 area(int radius)이다. 부모의 ㉠ area(int len)과 메서드 이름 및 매개변수 타입 목록이 같고 반환 타입도 int이므로, 부모의 인스턴스 메서드를 자식 클래스에서 재정의하는 **오버라이딩**이다. len과 radius처럼 매개변수 이름이 달라도 서로 다른 시그니처가 되는 것은 아니다.

㉡과 ㉢은 매개변수 개수가 다르므로 오버라이딩 관계가 아니다. 같은 반환 타입을 쓴다는 사실만으로 재정의가 성립하지 않으며, 반환 타입만 바꾸는 것만으로 오버로딩을 만들 수도 없다.

따라서 오버로딩은 ㉠·㉡, 오버라이딩은 ㉠·㉢으로 연결한 ①이 해당한다.

NoteBase

문 23. 다음에서 설명하는 아키텍처 스타일로 가장 적절한 것은?

사용자 인터페이스로부터 비즈니스 로직을 분리하여 애플리케이션의 시각적 요소나 그 이면에서 실행하는 비즈니스 로직을 서로 영향을 주지 않고 쉽게 수정 가능한 스타일의 아키텍처이다. 모델, 뷰, 제어 세 부분으로 이루어진 구조로 되어 있다.

① 계층 스타일

② MVC 스타일

③ 데이터 흐름 스타일

④ 클라이언트 - 서버 스타일

정답 2번

간단한 해설

모델·뷰·제어로 책임을 나누어 업무 로직과 사용자 인터페이스를 분리하는 것은 MVC 스타일이다. 데이터·업무 처리, 화면 표현, 입력 제어를 나누어 변경의 영향을 줄인다.

구성 요소별 해설

MVC는 Model, View, Controller로 책임을 나누어 사용자 인터페이스와 데이터·업무 처리를 분리하는 아키텍처 스타일이다. 제시문에서 모델·뷰·제어라는 세 구성요소를 직접 제시한 것이 결정적인 단서이다.

모델(Model): 애플리케이션의 데이터와 상태, 업무 규칙을 담당한다. 화면에 어떻게 표시할지보다 업무상 어떤 처리를 수행하고 상태를 어떻게 바꿀지를 다룬다.

뷰(View): 모델의 정보를 사용자에게 보여 주는 화면과 표현을 담당한다. 같은 데이터라도 다른 화면으로 표시할 수 있도록 업무 처리와 표시 책임을 분리한다.

제어(Controller): 사용자 입력이나 요청을 해석하여 적절한 모델의 동작을 호출하고 처리 흐름을 조정한다. 구체적인 뷰 갱신 방식이나 연결 방식은 구현에 따라 달라질 수 있지만 입력 처리와 업무·표현 책임의 분리가 핵심이다.

이처럼 책임을 분리하면 화면 변경이 업무 로직에 미치는 영향이나 그 반대의 영향을 줄일 수 있다. 완전히 아무 영향도 주지 않는다는 보장보다는 결합도를 낮춰 수정과 유지보수를 쉽게 한다는 의미이다. 세 요소로 구분하는 MVC 스타일인 ②에 해당한다.

NoteBase

문 24. 파이프 필터 소프트웨어 아키텍처에 대한 설명으로 가장 적절한 것은?

① 모든 필터가 동시에 작동하는 병렬처리 형식으로 처리 속도가 향상된다.

② 서비스 크기를 작게 나눈 아키텍처로 서비스를 독립적으로 개발하고 확장할 수 있다.

③ 서브 시스템이 입력 데이터를 받아 처리하고 결과를 다른 시스템에 보내는 작업이 반복된다.

④ 소프트웨어 기능을 수직으로 상호작용하는 여러 층으로 분리한 구조로 각 층 사이에 상호작용이 필요한 시스템에 가장 적합하다.

정답 3번

간단한 해설

파이프-필터 구조에서는 **한 필터의 출력이 다음 필터의 입력**으로 전달되면서 데이터 처리가 이어진다. 모든 필터가 반드시 동시에 실행된다는 의미는 아니다.

선지별 해설

① X: 파이프-필터 구조라고 해서 **모든 필터의 동시 실행과 속도 향상이 보장되는 것은 아니다**. 각 데이터 항목은 필요한 처리 단계 사이의 의존관계를 따라 전달된다. 스트림의 서로 다른 부분을 각 필터가 겹쳐 처리하는 파이프라인 병렬화는 가능하지만, 실행 방식과 데이터 특성·병목에 따라 실제 동시성 및 성능은 달라진다.

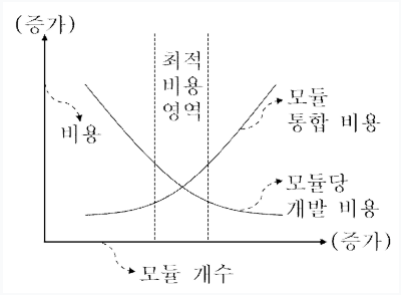
② X: 작은 독립 서비스 단위로 시스템을 분해하여 서비스별 개발과 확장을 강조하는 설명은 마이크로서비스 아키텍처의 특징이다. 파이프-필터의 핵심 구분 기준은 서비스의 크기 자체가 아니라 데이터 변환을 담당하는 필터들을 파이프로 연결한다는 점이다.

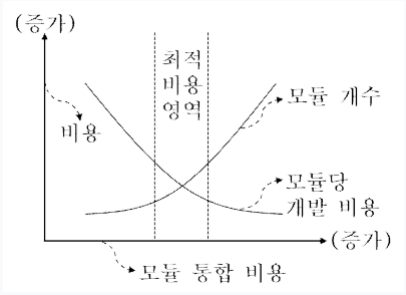
③ O: 필터는 입력 데이터를 받아 가공하고 그 결과를 출력한다. 파이프는 한 필터의 출력을 다음 필터의 입력으로 전달하므로, 여러 단계의 처리와 전달이 이어지는 구조이다. 예를 들어 데이터를 추출한 뒤 변환하고 다음 처리 단계로 보내는 방식으로 구성할 수 있다.

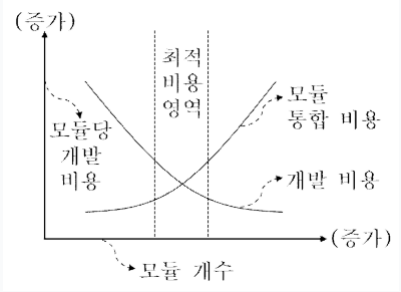
④ X: 기능을 여러 계층으로 나누고 계층 사이의 상호작용으로 시스템을 구성하는 설명은 계층형 아키텍처이다. 파이프-필터는 이러한 수직적 계층 분할보다 데이터가 변환 단계를 따라 이동하는 흐름에 초점을 둔다.

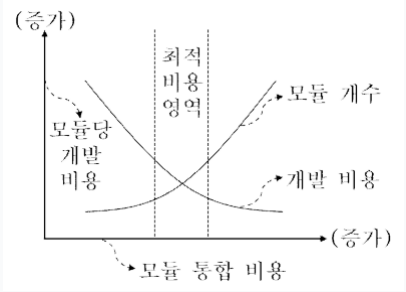
NoteBase

문 25. 최적 소프트웨어 개발 비용에 관한 그래프로 가장 적절한 것은?

① 

② 

③ 

④ 

정답 1번

간단한 해설

모듈을 잘게 나누면 **개별 모듈의 개발 부담은 줄지만 통합 부담은 커진다.** 최적의 모듈화 수준은 이 두 비용의 상충 관계를 함께 고려하여 정한다.

NoteBase

도식 해설

축의 의미: 가로축은 분할한 **모듈의 개수**, 세로축은 **비용**이다. 같은 소프트웨어를 얼마나 잘게 나누느냐에 따라 개발과 통합에 드는 비용이 어떻게 달라지는지 비교한다.

두 곡선의 의미: 모듈 수가 늘어나면 각 모듈이 작고 단순해지므로 그림의 모듈당 개발 비용은 감소한다. 반면 연결해야 할 모듈과 인터페이스가 많아져 모듈 통합 비용은 증가한다. 따라서 감소 곡선은 모듈당 개발 비용, 증가 곡선은 모듈 통합 비용이어야 한다.

최적 비용 영역: 모듈을 너무 적게 나누면 각 모듈의 복잡성이 커지고, 지나치게 잘게 나누면 통합 부담이 커진다. 이 두 부담을 함께 고려하여 전체 개발 비용이 낮아지는 적정 분할 수준을 찾는다. 두 곡선의 교점이 언제나 비용 합계의 정확한 최솟값이라고 단정하는 것은 아니다.

선택지 구분: ①은 가로축과 세로축, 증가·감소 곡선의 이름이 모두 위 관계에 맞는다. ②는 가로축의 모듈 개수와 증가 곡선의 모듈 통합 비용을 바꾸어 표시했다. ③은 두 비용을 비교할 세로축을 모듈당 개발 비용으로 한정하고 감소 곡선을 개발 비용으로 표시했다. ④는 ②의 축·곡선 혼동에 더해 ③처럼 세로축과 감소 곡선의 이름도 부적절하다.

따라서 모듈 개수가 증가할 때 모듈당 개발 비용은 감소하고 모듈 통합 비용은 증가하는 관계를 올바르게 표시한 ①이 가장 적절하다.

NoteBase

문 26. UML 4+1뷰를 이용한 시스템 표현 시, 시스템의 병렬 처리나 동기화 처리 등 시스템의 런타임(runtime) 동작에 초점을 맞춘 관점으로 가장 적절한 것은?

① 논리 관점(Logical view)

② 프로세스 관점(Process view)

③ 유스케이스 관점(Usecase view)

④ 구현 관점(Implementation view)

정답 2번

간단한 해설

병렬 처리와 동기화는 실행 중인 프로세스·스레드 사이의 관계에 관한 문제이다. 이를 중심으로 시스템을 설명하는 것은 **프로세스 관점**이다.

개념 구분

프로세스 관점은 실행 중인 프로세스나 스레드의 구성과 상호작용을 다룬다. 여러 실행 단위가 병렬로 동작할 때의 통신, 동기화, 처리 흐름 등을 설명하므로 **병렬 처리·동기화·런타임 동작**이라는 단서에 직접 대응한다.

논리 관점은 시스템이 제공할 기능을 뒷받침하는 주요 클래스·객체와 그 관계 등 논리적 구조를 중심으로 한다.

구현 관점은 소스 코드, 모듈, 컴포넌트 등 개발 산출물의 조직을 중심으로 하므로 실행 단위 간 동기화가 핵심인 프로세스 관점과 구별된다.

유스케이스 관점은 사용자와 시스템의 상호작용을 시나리오로 표현하여 주요 요구를 드러내고 다른 관점들을 연결·검증하는 역할을 한다. 4+1에서 '+1'에 해당하는 이 관점은 사용자가 원하는 동작을 설명하지만, 내부 병렬 실행 구조 자체에 초점을 맞추는 관점은 아니다.

따라서 제시된 설명에 해당하는 것은 ② 프로세스 관점이다.

NoteBase

문 27. 응집도와 결합도 정렬 방향에 따라 괄호(㉠~㉣)에 들어갈 소프트웨어 품질을 가장 적절하게 나열한 것은? (단, 결합도와 응집도는 모두 정렬되어 있다)

(㉠) ←	결합도	→ (㉡)
데이터 결합, ...(생략)... ,외부 결합, ...(생략)... , 내용 결합		

(㉢) ←	응집도	→ (㉣)
기능적 응집, ...(생략)... ,절차적 응집, ...(생략)... , 우연적 응집		

① ㉠: 나쁨, ㉡: 좋음, ㉢: 나쁨, ㉣: 좋음

② ㉠: 나쁨, ㉡: 좋음, ㉢: 좋음, ㉣: 나쁨

③ ㉠: 좋음, ㉡: 나쁨, ㉢: 나쁨, ㉣: 좋음

④ ㉠: 좋음, ㉡: 나쁨, ㉢: 좋음, ㉣: 나쁨

정답 4번

간단한 해설

품질 향상에는 **낮은 결합도와 높은 응집도**가 바람직하다. 그림의 왼쪽은 데이터 결합과 기능적 응집이므로 좋고, 오른쪽은 내용 결합과 우연적 응집이므로 나쁘다.

NoteBase

도식 해설

좋은 모듈 설계는 모듈 사이의 의존 정도인 결합도는 낮추고, 한 모듈 안 요소들이 하나의 목적에 관련되는 정도인 응집도는 높인다. 두 그림의 좌우 방향은 각 끝에 표시된 종류를 보고 판단해야 한다.

결합도 그림: 왼쪽 데이터 결합은 필요한 데이터를 매개변수로 주고받는 관계이고, 오른쪽 내용 결합은 다른 모듈의 내부 내용에 직접 의존하거나 접근하는 강한 결합이다. 중간의 외부 결합은 외부에서 정해진 형식·규약 등에 함께 의존하는 관계이다. 제시된 정렬에서 왼쪽으로 갈수록 결합도가 낮아 품질이 좋고, 오른쪽으로 갈수록 결합도가 높아 품질이 나쁘므로 ㉠은 좋음, ㉡은 나쁨이다.

응집도 그림: 왼쪽 기능적 응집은 모듈의 요소들이 하나의 기능을 위해 모인 높은 응집이고, 오른쪽 우연적 응집은 관련성이 낮은 요소들이 우연히 모인 낮은 응집이다. 중간의 절차적 응집은 정해진 실행 순서 때문에 요소들이 묶인 경우로, 하나의 기능적 목적을 공유하는 것보다 약하다. 따라서 왼쪽은 품질이 좋고 오른쪽은 나쁘므로 ㉢은 좋음, ㉣은 나쁨이다.

㉠·㉡·㉢·㉣에 들어갈 값은 좋음·나쁨·좋음·나쁨으로 ④에 해당한다. 이 그림에서는 두 경우 모두 왼쪽이 좋지만, 결합도는 낮아지는 방향이고 응집도는 높아지는 방향이라는 차이를 구별해야 한다.

NoteBase

문 28. 다음 C 소스 코드의 두 함수 사이의 결합도와 funcB()의 응집도를 가장 적절하게 나열한 것은?

```
void funcA() {
    int x, y, res, flag;
    scanf("%d %d %d\n", &x, &y, &flag);
    res = funcB(x, y, flag);
    printf("%d\n", res);
}
int funcB(int a, int b, int f) {
    if(f == 1) return (a * b);
    else return (a / b);
}
```

① 제어 결합, 논리적 응집

② 제어 결합, 기능적 응집

③ 공통 결합, 논리적 응집

④ 공통 결합, 기능적 응집

정답 1번

간단한 해설

flag가 funcB()에서 수행할 연산을 선택하므로 두 함수는 **제어 결합** 관계이다. funcB()는 곱셈과 나눗셈이라는 유사한 기능 중 하나를 제어값에 따라 선택하므로 **논리적 응집**에 해당한다.

NoteBase

핵심 풀이

두 함수 사이의 결합도: funcA()는 funcB(x, y, flag)를 호출한다. 이때 x와 y는 연산에 사용할 자료이고, flag는 funcB()의 매개변수 f로 전달되어 곱셈과 나눗셈 중 수행할 처리를 결정한다. 한 모듈이 다른 모듈의 내부 처리 흐름을 선택하는 **제어 정보를 전달하므로 제어 결합**이다. 단순히 계산 자료만 주고받는 자료 결합과 구별된다.

공통 결합과의 구별: 공통 결합은 여러 모듈이 전역 변수 같은 공통 자료 영역을 공유할 때 나타난다. 그림의 x, y, res, flag는 funcA() 안에 선언된 지역 변수이며, funcB()에는 인수로 값이 전달된다. 두 함수가 같은 전역 자료 영역을 공유하는 구조가 아니므로 공통 결합으로 보지 않는다.

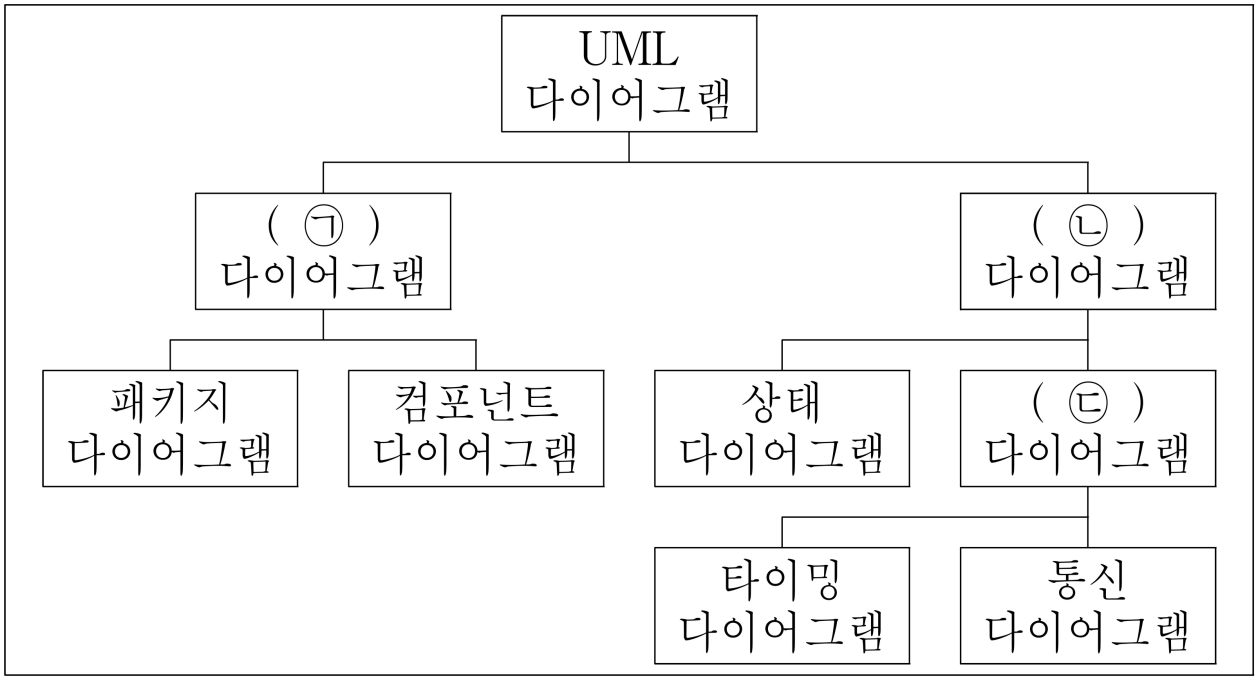
funcB()의 응집도: f가 1이면 곱셈 결과를 반환하고, 그 밖에는 나눗셈 결과를 반환한다. 두 처리는 산술 연산이라는 범주에서 관련되지만, 하나의 결과를 위해 모두 수행하는 작업들이 아니라 **외부에서 받은 제어값에 따라 선택되는 서로 다른 기능**이다. 이처럼 유사한 종류의 기능을 묶고 그중 하나를 선택하는 구조는 논리적 응집이다.

기능적 응집과의 구별: 기능적 응집은 모듈의 모든 구성 요소가 하나의 명확한 기능을 수행하는 데 함께 기여하는 경우이다. 이 코드에서는 곱셈과 나눗셈이 선택적인 별개 기능으로 묶여 있으므로, 함수를 하나로 작성했다는 이유만으로 기능적 응집이라고 판단하지 않는다.

두 함수 사이의 결합도는 제어 결합이고 funcB()의 응집도는 논리적 응집이므로 ①과 일치한다.

NoteBase

문 29. 다음 분류의 괄호(㉠~㉢)에 들어갈 용어를 가장 적절하게 나열한 것은?



① ㉠: 동작, ㉡: 구조, ㉢: 상호작용

② ㉠: 동작, ㉡: 상호작용, ㉢: 구조

③ ㉠: 구조, ㉡: 상호작용, ㉢: 동작

④ ㉠: 구조, ㉡: 동작, ㉢: 상호작용

정답 4번

간단한 해설

UML 다이어그램은 크게 **구조와 동작**으로 나뉘며, 상호작용 다이어그램은 동작 다이어그램에 포함된다.
패키지·컴포넌트는 구조를, 타이밍·통신은 상호작용을 나타낸다.

NoteBase

개념 구분

㉠ 구조: **시스템을 이루는 요소와 그 관계**를 정적인 관점에서 나타낸다. 패키지 다이어그램은 모델 요소를 묶는 패키지와 그 관계를, 컴포넌트 다이어그램은 구성 요소와 인터페이스·의존 관계를 표현하므로 이 범주에 속한다.

㉡ 동작: 시스템이 수행하는 행위나 시간에 따른 변화를 나타낸다. 그림의 상태 다이어그램은 객체 등의 상태와 사건에 따른 전이를 표현한다. 상호작용 다이어그램도 참여자 사이에서 일어나는 동작을 다루므로 동작 다이어그램의 하위 범주이다.

㉢ 상호작용: 참여자들이 메시지를 주고받는 모습을 중심으로 동작을 나타낸다. 타이밍 다이어그램은 시간에 따른 상태·값의 변화와 시간 제약에, 통신 다이어그램은 참여자 사이의 연결과 메시지 교환에 초점을 둔다.

분류도의 핵심은 **구조와 동작이 상위의 두 범주이고, 상호작용은 동작에 포함된다**는 점이다. ①·②는 패키지와 컴포넌트가 속한 ㉠을 동작으로 잘못 분류하고, ③은 상위 범주인 동작과 그 하위 범주인 상호작용을 뒤집었다. 그림은 일부 유형만 예시한 것이므로 제시된 다이어그램이 각 범주의 전부라는 뜻은 아니다.

따라서 ㉠ 구조, ㉡ 동작, ㉢ 상호작용이며, 이 대응과 일치하는 선택지는 ④이다.

NoteBase

문 30. 다음 설명에 관한 사용사례 다이어그램(usecase diagram)으로 가장 적절한 것은?

학생은 수강 신청과 성적 확인을 위해 학생 인증 과정이 필요하다. 또한, 성적이 높은 경우 장학금 확인이 가능하다.

① 

② 

③ 

④ 

정답 3번

간단한 해설

공통 필수 기능인 학생 인증은 **include 화살표가 향하는 대상**이다. 조건부 추가 기능인 장학금 확인의 extend 화살표는 기본 사용사례인 성적 확인으로 향한다.

도식 해설

학생 인증은 수강 신청과 성적 확인을 수행할 때 공통으로 필요한 기능이다. 따라서 두 사용사례가 학생 인증을 포함하는 include 관계이며, 화살표는 **포함하는 사용사례에서 포함되는 사용사례**로 향한다. 즉 '수강 신청 → 학생 인증', '성적 확인 → 학생 인증'이어야 한다.

장학금 확인은 성적이 높은 경우에 추가되는 기능이므로 성적 확인을 확장하는 extend 관계이다. 이 화살표는 **확장 기능에서 기본 사용사례**로 향하므로 '장학금 확인 → 성적 확인'이어야 한다. 기능이 나중에 실행된다는 이유로 성적 확인에서 장학금 확인 쪽으로 그리는 것이 아니다.

①은 두 include 화살표와 extend 화살표가 모두 반대이다.

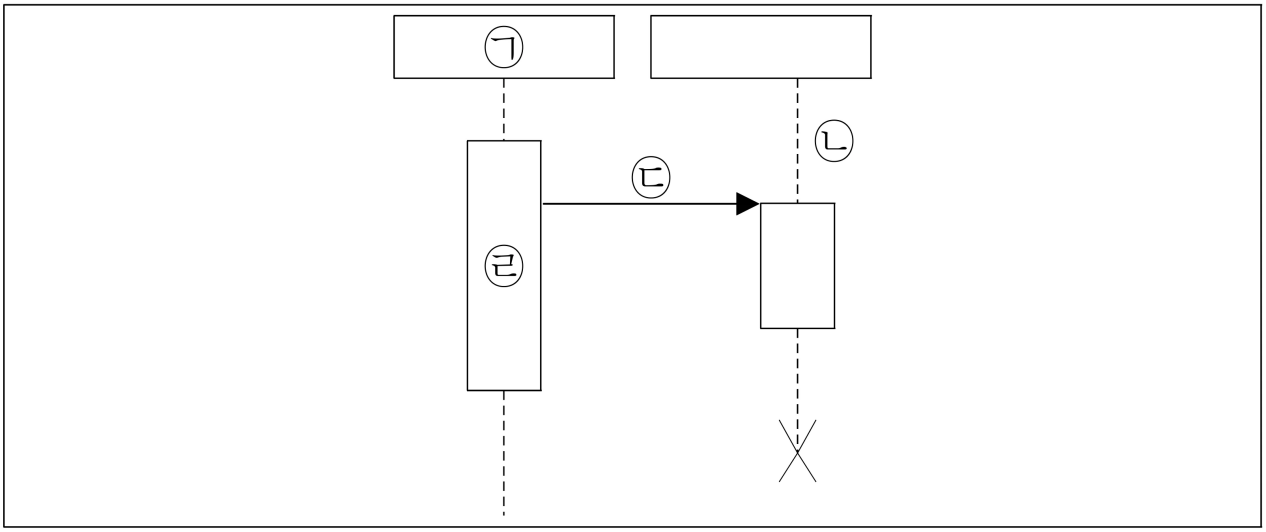
②는 extend 방향은 맞지만 두 include 방향이 반대이다.

④는 두 include 방향은 맞지만 extend 방향이 반대이다.

③은 수강 신청과 성적 확인에서 학생 인증으로 include 화살표가 향하고, 장학금 확인에서 성적 확인으로 extend 화살표가 향하므로 주어진 조건에 해당한다.

NoteBase

문 31. 다음 순서(sequence) 다이어그램의 각 요소(㉠~㉣)에 대한 용어를 가장 적절하게 나열한 것은?



① ㉠: 객체, ㉡: 생명선, ㉢: 메서드 호출, ㉣: 활동 막대

② ㉠: 객체, ㉡: 활동 막대, ㉢: 메서드 호출, ㉣: 생명선

③ ㉠: 메서드 호출, ㉡: 활동 막대, ㉢: 객체, ㉣: 생명선

④ ㉠: 메서드 호출, ㉡: 생명선, ㉢: 객체, ㉣: 활동 막대

정답 1번

간단한 해설

수직 점선은 객체의 존재를 나타내는 **생명선**이고, 그 위의 세로 직사각형은 실행 구간을 나타내는 **활동 막대**이다. 위쪽 직사각형은 객체, 객체 사이 화살표는 메서드 호출을 나타낸다.

NoteBase

도식 해설

㉠ 객체: 도식 위쪽의 가로 직사각형은 상호작용에 참여하는 객체를 나타내는 머리 부분이다. 이 아래로 해당 객체의 생명선이 이어진다.

㉡ 생명선: 객체 아래로 내려오는 **수직 점선**은 시간 흐름에 따른 객체의 존재를 나타낸다. 순서 다이어그램에서는 대체로 위에서 아래로 시간이 진행하며, 오른쪽 아래의 X는 해당 객체의 소멸을 나타낸다.

㉢ 메서드 호출: 왼쪽에서 오른쪽 객체로 향하는 실선 화살표는 메시지를 보내 메서드 실행을 요청하는 것을 나타낸다. 그림처럼 채워진 화살촉을 사용하는 호출은 동기 호출을 표현한다.

㉣ 활동 막대: 생명선 위에 겹쳐진 **길쭉한 세로 직사각형**은 객체가 동작을 수행하는 실행 구간을 나타낸다. 객체의 존재를 나타내는 생명선 전체와 구별된다.

따라서 ㉠ 객체, ㉡ 생명선, ㉢ 메서드 호출, ㉣ 활동 막대이며 ①과 일치한다.

NoteBase

문 32. 다음 자바 소스 코드에 적용한 디자인 패턴으로 가장 적절한 것은?

```
abstract class Shape{ ...(중략)... }
class Circle extends Shape {
...(중략)...
    public:
        Circle(Circle src) { super(src);
            this.radius = src.getRadius(); ...(중략)... }
        Shape Clone() { return new Circle(this); }
        int getRadius() { return radius; }
    private:
        int radius;
...(중략)...
}
```

① 어댑터(adapter)

② 싱글톤(singleton)

③ 프로토타입(prototype)

④ 플라이웨이트(flyweight)

정답 3번

간단한 해설

Clone()은 현재 객체를 인수로 new Circle(this)를 호출하여 **기존 객체의 상태를 복사한 새 객체**를 만든다.
이러한 객체 생성 방식은 **프로토타입 패턴**에 해당한다.

NoteBase

핵심 풀이

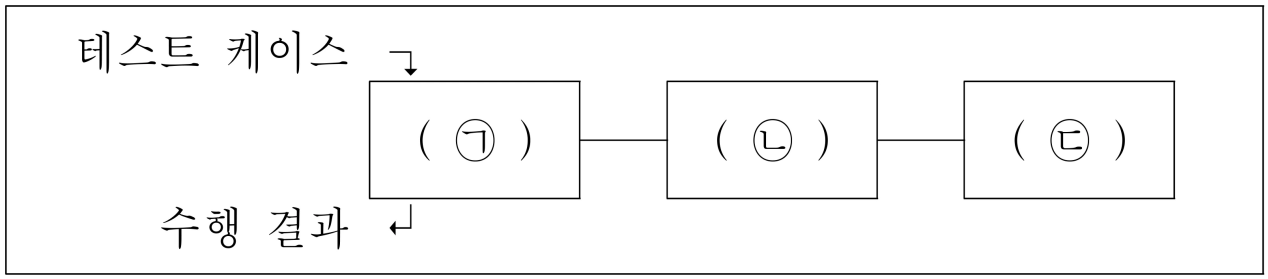
프로토타입 패턴은 기존 객체를 원형으로 삼아 그 상태를 복사한 새 객체를 만드는 생성 패턴이다. 제시된 Clone()의 'return new Circle(this);'는 현재 객체인 this를 복사 생성자에 전달하여 **별개의 Circle 객체를 생성**하고, 이를 Shape 형식으로 반환한다.

복사 생성자 Circle(Circle src)는 복사할 원본 객체를 src로 받는다. super(src)는 원본을 상위 클래스 생성자에 전달하고, 'this.radius = src.getRadius();'는 원본의 반지름 값을 새 객체의 radius에 복사한다. 즉, Clone() 호출 → 복사 생성자 호출 → 원본 상태 복사라는 흐름이 프로토타입 패턴을 드러낸다.

싱글톤은 인스턴스를 하나로 제한하고 그 인스턴스에 접근하게 하는 패턴이다. 여기서는 Clone()을 호출할 때마다 new Circle(this)로 새 객체를 생성하므로 **하나의 객체를 계속 사용하는 싱글톤**과 구별된다.
플라이웨이트는 여러 곳에서 사용할 공통 상태를 공유하는 데, 어댑터는 기존 인터페이스를 요구되는 인터페이스로 변환하는 데 초점이 있으므로, 이 코드의 핵심인 객체 복제와 다르다.

NoteBase

문 33. 다음 단위 테스트 환경의 괄호(㉠~㉢)에 들어갈 용어로 가장 적절한 것은?



① ㉠: 스터브, ㉡: 테스트 모듈, ㉢: 드라이버

② ㉠: 스터브, ㉡: 드라이버, ㉢: 테스트 모듈

③ ㉠: 드라이버, ㉡: 테스트 모듈, ㉢: 스터브

④ ㉠: 드라이버, ㉡: 스터브, ㉢: 테스트 모듈

정답 3번

간단한 해설

테스트 케이스를 받아 시험 대상을 호출하고 결과를 확인하는 것은 **드라이버**이다. 시험 대상은 테스트 모듈이고, 대상이 호출하는 하위 기능을 대신하는 것은 스터브이다.

도식 해설

단위 테스트에서는 시험 대상 모듈을 호출할 상위 프로그램과, 시험 대상이 이용하는 하위 모듈을 대신하는 보조 코드가 필요할 수 있다. 호출하는 쪽은 드라이버, 호출받는 하위 기능을 대신하는 쪽은 스터브이다.

㉠ 드라이버: 그림에서 테스트 케이스가 입력되고 수행 결과가 출력되는 왼쪽 상자이다. 테스트 입력을 준비하여 대상 모듈을 호출하고 실행 결과를 수집·확인하는 역할을 하므로 드라이버에 해당한다.

㉡ 테스트 모듈: 드라이버와 하위 보조 모듈 사이에 놓인 실제 시험 대상이다. 드라이버의 입력을 받아 기능을 수행하며, 처리 과정에서 필요한 하위 기능을 호출한다.

㉢ 스터브: 시험 대상 모듈이 사용하는 하위 모듈의 동작을 간단히 대신하는 코드이다. 필요한 응답이나 미리 정한 값을 제공하여 하위 모듈이 아직 완성되지 않았거나 분리된 상태에서도 시험을 진행하도록 돕는다.

따라서 ㉠은 드라이버, ㉡은 테스트 모듈, ㉢은 스터브로 ③에 해당한다. 드라이버는 대상을 호출하는 쪽, 스터브는 대상으로부터 호출받는 쪽이라는 역할 차이가 핵심이다.

NoteBase

문 34. 다음 소스 코드에서 조건 커버리지(coverage)를 만족하지만, 분기 커버리지는 만족하지 않는 테스트데이터가 인자인 함수 호출을 가장 적절하게 나열한 것은? (단, 단축 연산은 적용하지 않는다)

소스 코드	void func(int x, int y) {
	  if((x > 10) && (y != 10)) { ...(중략)... }
	}

함수 호출	가. func(9, 9); 나. func(10, 10); 다. func(11, 10); 라. func(11, 11);

① 가, 나

② 나, 다

③ 다, 라

④ 가, 다

정답 4번

간단한 해설

조건 커버리지는 **개별 조건의 참·거짓**을, 분기 커버리지는 **전체 조건의 참·거짓**을 확인한다. AND 조건에서 두 개별 조건이 서로 반대의 진릿값을 갖도록 바꾸면, 각 조건은 참·거짓을 모두 경험해도 전체 결과는 계속 거짓일 수 있다.

NoteBase

계산 과정

판정 기준: 첫째 조건을 A: $x > 10$, 둘째 조건을 B: $y \neq 10$이라고 하자. 전체 분기 조건은 A B이다. **조건 커버리지**는 A와 B 각각에서 참·거짓이 모두 나와야 하고, **분기 커버리지**는 전체 조건의 참·거짓이 모두 나와야 한다. 문제에서 단축 연산을 적용하지 않으므로 A가 거짓이어도 B를 평가한다.

가. func(9, 9): A는 거짓, B는 참이다. 따라서 전체 조건은 거짓이다.

나. func(10, 10): A는 거짓, B도 거짓이다. 따라서 전체 조건은 거짓이다. 첫째 비교가 10 이상이 아니라 10 초과라는 점에 주의한다.

다. func(11, 10): A는 참, B는 거짓이다. 따라서 전체 조건은 거짓이다.

라. func(11, 11): A와 B가 모두 참이므로 전체 조건도 참이다.

① 가·나: A가 두 번 모두 거짓이므로 조건 커버리지를 만족하지 않는다. 전체 조건도 두 번 모두 거짓이다.

② 나·다: B가 두 번 모두 거짓이므로 조건 커버리지를 만족하지 않는다. 전체 조건도 두 번 모두 거짓이다.

③ 다·라: A가 두 번 모두 참이므로 조건 커버리지를 만족하지 않는다. 전체 조건은 거짓·참이므로 분기 커버리지는 만족한다.

④ 가·다: A는 거짓·참, B는 참·거짓으로 각각 두 결과를 모두 갖는다. 그러나 두 호출 모두 전체 조건은 거짓이므로 참 분기는 실행되지 않는다. 따라서 **조건 커버리지는 만족하고 분기 커버리지는 만족하지 않는** 가·다의 조합, 즉 ④가 정답이다.

NoteBase

문 35. 다음에서 정적 테스트 기법의 비정형 방법을 있는 대로 모두 고른 것은?

가. 개별 검토
나. 동료 검토
다. 검토 회의
라. 기호 계산
마. 모형 검증
바. 감사(auditing)

① 가, 나, 다, 라
② 가, 나, 다, 바
③ 가, 라, 마, 바
④ 나, 다, 라, 마

정답 2번

간단한 해설

기호 계산과 모형 검증은 **수학적 표현과 논리에 기초한 정형 방법**이다. 개별 검토·동료 검토·검토 회의·감사는 사람의 검토와 기준 대조를 중심으로 하는 비정형 방법에 해당한다.

NoteBase

보기별 해설

정적 테스트는 대상 프로그램을 실제 입력으로 실행하지 않고 코드·명세·설계 등의 산출물을 점검하는 활동이다. 이 문제의 정형·비정형 구분은 **수학적 모형과 논리에 기초한 검증인지**를 기준으로 한다. 검토나 감사의 절차가 엄격하고 공식적이라는 의미의 '정형적 리뷰'와 구별해야 한다.

가. 비정형 방법: 개별 검토는 검토자가 산출물을 직접 읽으며 누락·모순·오류를 찾는 방법이다. 사람의 이해와 점검 기준을 이용하며 수학적 증명이나 상태 공간 검증을 수행하는 방법은 아니다.

나. 비정형 방법: 동료 검토는 동료 개발자 등이 코드나 문서를 검토하여 결함을 발견하는 방법이다. 다른 사람의 관점과 경험을 활용하는 검토 활동에 해당한다.

다. 비정형 방법: 검토 회의는 여러 검토자가 발견한 문제를 공유하고 논의하여 산출물의 결함이나 개선 사항을 확인하는 방법이다. 회의 절차가 체계적이더라도 그것만으로 수학적 정형 검증이 되지는 않는다.

라. 정형 방법: 기호 계산은 구체적인 입력값 대신 기호를 사용하여 식과 조건을 분석하는 방법이다. 프로그램의 연산 결과와 경로 조건 등을 수학적 표현으로 다루므로 이 분류에서는 정형 방법에 해당한다.

마. 정형 방법: 모형 검증은 시스템을 상태와 전이로 나타낸 모형이 주어진 성질을 만족하는지 논리적으로 확인하는 방법이다. 모형의 상태 공간과 명세된 속성을 이용하는 정형 검증에 해당한다.

바. 비정형 방법: 감사는 산출물과 개발 과정이 정해진 기준·절차·계약 등의 요구에 부합하는지를 독립적으로 점검하는 활동이다. 감사가 공식적인 절차에 따라 수행되더라도, 여기서 말하는 **수학적 정형 검증과는 다르다**.

비정형 방법은 가, 나, 다, 바이므로 ②와 일치한다.

NoteBase

문 36. 다음 테스트 방법에 대한 명칭을 가장 적절하게 나열한 것은?

가. 개발 소프트웨어 시스템을 사용자에게 전달하기 전에 수행하는 테스트로 구축된 시스템에 각 요구사항이 올바르게 구현되었는지를 점검하는 데모 형식으로 테스트를 진행한다.
나. 변경한 부분을 중점적으로 테스트하기 위하여 변경에 영향을 받는 범위에 대한 테스트 케이스를 마련하여 테스트를 수행한다. 테스트 수행 범위를 최소화하기 때문에 비용 절감에 효과적이나 완전한 테스트가 이루어지지 않는다는 단점이 있다.

① 가: 인수 테스트, 나: 선택적 회귀 테스트
② 가: 인수 테스트, 나: 우선순위 회귀 테스트
③ 가: 통합 테스트, 나: 선택적 회귀 테스트
④ 가: 통합 테스트, 나: 우선순위 회귀 테스트

정답 1번

간단한 해설

가는 요구사항을 충족한 시스템을 받아들일 수 있는지 확인하는 **인수 테스트**이다. 나는 변경의 영향을 받는 케이스만 골라 수행하는 **선택적 회귀 테스트**이다.

대응 관계 해설

가: 인수 테스트이다. 사용자에게 인도하기 전에 합의한 요구사항과 인수 기준을 충족하는지 확인하여 **시스템을 받아들일 수 있는지 판단**하는 것이 핵심이다. 지문의 데모는 이를 확인하는 한 방식이며 인수 테스트가 언제나 데모에만 한정되는 것은 아니다. 통합 테스트는 모듈이나 하위 시스템을 결합했을 때 인터페이스와 상호작용이 제대로 동작하는지 확인하는 데 초점이 있다.

나: 선택적 회귀 테스트이다. 변경 영향 분석을 바탕으로 **영향받는 범위의 테스트 케이스를 골라 재수행**하므로 전체 회귀 테스트보다 시간과 비용을 줄일 수 있다. 다만 영향 범위를 잘못 판단하거나 필요한 케이스를 제외하면 결함을 놓칠 위험이 있다. 우선순위 회귀 테스트는 주로 중요도나 결함 발견 가능성 등에 따라 실행 순서를 정하는 접근이므로, 실행할 부분집합을 선택하는 것과 구별된다.

따라서 가는 인수 테스트, 나는 선택적 회귀 테스트이며 ①과 일치한다.

NoteBase

문 37. 다음의 보기에서 설명하는 테스트로 가장 적절한 것은?

평소보다 많은 비정상적인 양, 값, 빈도, 부피 등으로 부하를 발생시켜 부하가 최고치인 상황에서 시스템의 반응을 살피고 이때 발생하는 오류를 찾는 방법이다.

① 성능 테스트
② 보안 테스트
③ 안정성 테스트
④ 스트레스 테스트

정답 4번

간단한 해설

평소보다 훨씬 많은 부하를 의도적으로 가해 극한 상황에서 시스템의 반응과 오류를 확인하는 것은 **스트레스 테스트**이다. 핵심 단서는 **비정상적으로 높은 부하**를 준다는 점이다.

공통 해설

스트레스 테스트는 정상적인 운용 범위를 넘는 부하나 극한 조건을 가하여 시스템이 어떻게 반응하고 어떤 오류를 일으키는지 확인하는 시험이다. 보기의 '평소보다 많은 비정상적인 양, 값, 빈도, 부피'는 부하를 의도적으로 높이는 조건이며, '부하가 최고치인 상황에서 시스템의 반응'을 살핀다는 설명이 이 시험의 목적과 연결된다. 예를 들어 동시에 들어오는 요청을 평소보다 크게 늘려 처리 지연이나 오류가 발생하는 양상을 확인할 수 있다.

스트레스 테스트는 성능 테스트의 한 유형이므로 두 용어가 완전히 별개인 것은 아니다. 다만 성능 테스트라는 넓은 명칭보다, **정상 범위를 넘는 부하에서의 동작**을 확인한다는 구체적인 조건에 맞는 명칭이 스트레스 테스트이다. 보기의 분류 기준은 보안 취약점이나 일반적인 안정성 자체가 아니라 극한 부하를 가하는 시험 방법이다.

NoteBase

문 38. 다음 유지보수에 해당하는 유형으로 가장 적절하지 않은 것은?

◯ 현재 운영 중인 중고차 관리시스템에 중고차를 판매할 수 있는 기능을 추가한다.
◯ 현재 운영 중인 중고차 관리시스템에서 세글자를 초과하는 판매자 이름에 대해 화면 출력이 불가하고 프린터 출력 시 동작이 멈추는 문제가 발생하여 이를 보수한다.
◯ 현재 운영 중인 중고차 관리시스템은 윈도우즈 시스템에서 동작하고 있으나, 리눅스 시스템에서도 동작하도록 보수한다.

① 적응 유지보수(adaptive maintenance)
② 완전 유지보수(perfective maintenance)
③ 수정 유지보수(corrective maintenance)
④ 예방 유지보수(preventive maintenance)

정답 4번

간단한 해설

기능 추가는 완전 유지보수, 발생한 출력 오류의 보수는 수정 유지보수, 운영체제 변경에 대응하는 작업은 적응 유지보수이다. **장래 문제를 미리 막는 예방 유지보수**에 해당하는 사례는 제시되지 않았다.

대응 관계 해설

유지보수 유형은 변경의 직접적인 목적을 기준으로 구분한다. 새로운 요구의 반영, 이미 발생한 결함의 수정, 실행환경 변화에 대한 적응, 장래 문제의 예방은 서로 다른 목적이다.

중고차 판매 기능 추가 → 완전 유지보수: 운영 중인 시스템에 새로운 사용자 요구를 반영하여 기능을 확장하는 작업이다. 현재 존재하는 오류를 고치는 것이 아니라 시스템의 기능과 유용성을 개선한다.

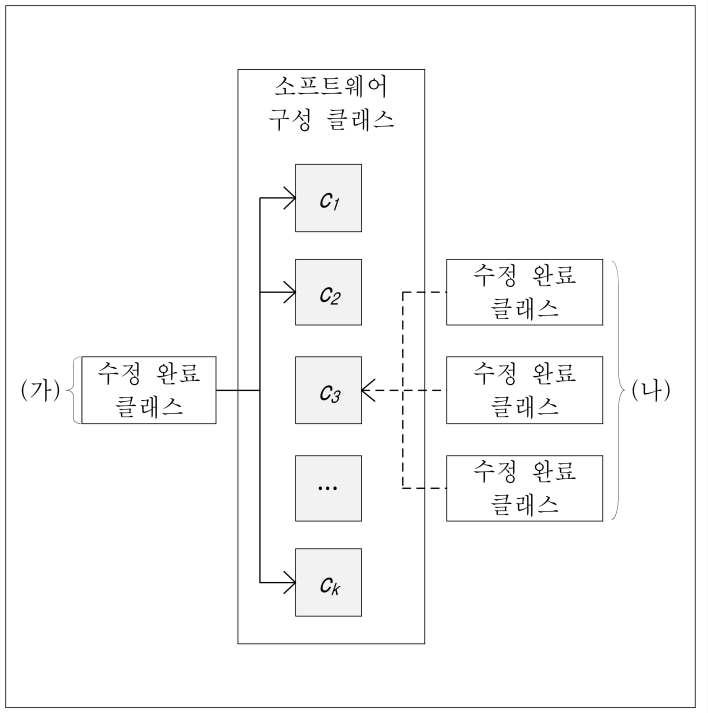
판매자 이름의 화면·프린터 출력 오류 보수 → 수정 유지보수: 세 글자를 초과하는 이름이 화면에 출력되지 않고 프린터 출력 중 멈추는 것은 실제 발생한 결함이다. 이를 분석하고 바로잡는 작업은 corrective maintenance에 해당한다.

리눅스에서도 동작하도록 보수 → 적응 유지보수: 기존 윈도우즈 환경 외에 다른 운영체제에서도 작동하도록 변경하는 작업이다. 소프트웨어가 변화하거나 추가된 실행환경에 적응하도록 하는 것이 목적이다.

예방 유지보수는 향후 장애나 유지보수의 어려움을 줄이기 위해 잠재적 결함을 미리 제거하거나 내부 구조 등을 개선하는 작업이다. 제시된 세 사례는 각각 완전·수정·적응 유지보수이므로, 사례에 해당하지 않는 유형은 예방 유지보수인 ④이다.

NoteBase

문 39. 다음 그림은 변경 요청에 따른 수정이 필요한 클래스의 관계를 보여준다. 이런 형태의 코드 스멜(code smell) 용어를 올바르게 연결한 것으로 가장 적절한 것은? (단, 화살표는 클래스의 수정 완료에 따라 추가변경이 발생하는 클래스를 지칭한다)



① (가): Middle Man, (나): Message Chains

② (가): Middle Man, (나): Divergent Change

③ (가): Shotgun Surgery, (나): Message Chains

④ (가): Shotgun Surgery, (나): Divergent Change

정답 4번

간단한 해설

한 번의 변경 때문에 **여러 클래스를 수정**해야 하면 Shotgun Surgery이다. 반대로 **여러 변경 이유가 한 클래스에 집중**되면 Divergent Change이다.

NoteBase

도식 해설

(가)는 Shotgun Surgery이다. 하나의 변경을 처리한 뒤 여러 구성 클래스를 추가로 수정해야 하는 모습으로, **한 변경의 수정 지점이 여러 클래스에 흩어져** 있다. 관련 책임이 분산되어 있어 한 곳만 바꾸면 수정이 끝나지 않는 문제가 핵심이다.

(나)는 Divergent Change이다. 서로 다른 여러 변경에 따라 같은 클래스가 반복해서 수정되는 모습으로, **한 클래스에 서로 다른 변경 이유가 모여** 있다. 그림에서는 오른쪽 여러 클래스의 수정이 가운데 같은 클래스로 영향을 보내므로 한 클래스가 여러 관심사에 결합된 상태를 나타낸다.

Middle Man은 직접 수행하는 역할보다 다른 객체로 전달·위임하는 역할이 지나치게 많은 경우이다. Message Chains는 한 객체를 얻고 다시 그 객체를 통해 다음 객체를 얻는 식으로 호출이 길게 연결되는 경우이다. 이 그림의 화살표는 호출이나 위임이 아니라 추가 수정의 발생 방향을 뜻하므로 이 두 용어로 판단하지 않는다.

따라서 (가)는 Shotgun Surgery, (나)는 Divergent Change이며 ④와 일치한다.

NoteBase

문 40. 프로세스 성숙도를 평가하기 위한 CMMI와 SPICE에 대한 설명 중 가장 적절하지 않은 것은?

① CMMI-DEV의 주요 프로세스 영역은 프로세스 관리, 프로젝트 관리, 엔지니어링, 지원으로 구성된다.

② SPICE 모델에서 표준으로 정의된 프로세스를 사용하여 계획하고 관리하는 단계는 관리(managed) 단계이다.

③ CMMI와 SPICE의 중요한 차이점은 성숙도 레벨과 심사영역의 구분에 있으며, 두 모델의 성숙도의 최고 수준은 모두 레벨 5이다.

④ CMMI 모델에서 표준화 프로젝트 프로세스는 존재하나 프로젝트 목표와 활동에 대한 정량적 측정이 불가능한 단계는 정의(defined) 단계이다.

정답 2번

간단한 해설

SPICE에서 표준으로 정의된 프로세스를 사용하는 단계는 **레벨 3의 확립(established) 단계**이다. 레벨 2의 관리(managed) 단계는 프로세스 수행을 계획·점검·조정하고 작업 산출물을 관리하는 단계이다.

출제 포인트

CMMI와 SPICE의 평가 구조를 구분하고, 프로세스 관리·표준화·정량적 관리가 각각 어느 수준에 해당하는지 판단하는 문제이다.

핵심 개념

CMMI는 조직의 개발 프로세스를 평가하고 개선하기 위한 모델이다. 단계적 표현에서는 조직의 성숙도를 레벨 1부터 5까지 구분한다.
SPICE는 개별 프로세스가 목적을 달성하고 체계적으로 수행되는 정도를 평가하는 체계이다. 프로세스 능력 수준을 레벨 0부터 5까지 구분한다.
관리: 수행을 계획·점검·조정하고 산출물을 통제한다.
표준화: 조직의 표준 프로세스를 정의하고 각 프로젝트에 맞게 적용한다. CMMI의 레벨 3은 정의(defined), SPICE의 레벨 3은 확립(established)이다.
정량적 관리: 수치와 통계적 기법으로 프로세스 성과와 변동을 관리한다. CMMI에서는 레벨 4의 특징이다.

NoteBase

선지별 해설

① O
CMMI-DEV의 프로세스 영역은 프로세스 관리, 프로젝트 관리, 엔지니어링, 지원의 네 범주로 묶는다. 이는 성숙도 레벨의 구분이 아니라 프로세스 영역의 성격에 따른 분류이다.

② X
'표준으로 정의된 프로세스를 사용'하는 특징은 SPICE의 레벨 3인 확립(established) 단계에 해당한다. 레벨 2인 관리(managed) 단계에서는 수행을 계획·점검·조정하고 산출물을 관리한다. 선지의 '관리(managed) 단계'를 '확립(established) 단계'로 고친다.

③ O
CMMI의 단계적 표현은 조직의 성숙도 레벨을 평가하고, SPICE는 평가할 프로세스와 그 프로세스의 능력 수준을 구분한다. 이 문항이 비교하는 CMMI 성숙도와 SPICE 프로세스 능력 수준은 모두 최고 레벨이 5이다. 다만 SPICE의 레벨은 엄밀히는 개별 프로세스의 '능력 수준'이다.

④ O
CMMI의 레벨 3인 정의(defined) 단계에서는 조직의 표준 프로세스를 바탕으로 프로젝트 프로세스를 정한다. 프로세스 성과를 통계적으로 관리하는 단계는 레벨 4인 정량적 관리(quantitatively managed)이다. 선지의 '정량적 측정이 불가능'은 이 단계 구분에서 정량적 관리가 아직 확립되지 않았다는 취지로 이해한다. 레벨 3에서 수치 자료의 수집이나 측정 자체가 불가능하다는 뜻은 아니다.

오답 포인트

②의 '계획하고 관리'만 보면 관리 단계로 판단하기 쉽다. 그러나 수준을 가르는 표현은 '표준으로 정의된 프로세스를 사용'이다. ④에서는 단순한 수치 측정과 통계적 기법을 통한 정량적 관리를 구분한다.

다시 나오면 이렇게 판단

계획·점검·산출물 통제는 관리 수준, 조직 표준 프로세스의 정의·적용은 레벨 3으로 연결한다. 같은 레벨 3이라도 CMMI는 정의, SPICE는 확립이라는 명칭을 사용한다. 정량적·통계적 관리가 나오면 CMMI 레벨 4와 구분한다.

암기 포인트

표준 프로세스는 레벨 3: CMMI는 정의, SPICE는 확립. CMMI 레벨 4는 정량적 관리.

NoteBase