제132회 컴퓨터시스템응용기술사 해설집

2024.01.27

국가기술자격 기술사 시험문제

기술사 제 132 회 제 3 교시 (시험시간: 100 분)

분야	정보통신	자격종목	컴퓨터시스템응용기술사	수검번호		성명	

※ 다음 문제 중 4 문제를 선택하여 설명하시오. (각 10 점)

1. 앰비언트 컴퓨팅(Ambient Computing)은 인간의 개일이 없어도 시스템이 스스로 동작하는 기술이다. 이와 관련하여 아래 사항을 설명하시오.
   가. 앰비언트 컴퓨팅의 개념
   나. 개념도 및 기술요소
   다. 앰비언트 컴퓨팅과 IoT(Internet of Things) 비교

2. 가상머신(Virtual Machine)과 컨테이너(Container)에 대하여 구체적으로 설명하고, 차이점을 상세히 설명하시오.

3. 맨체스터 코딩(Manchester Coding)은 데이터 저장과 디지털 데이터 통신 분야에서 다양하게 활용되는 코딩방식이다. 이와 관련하여 아래사항을 설명하시오.
   가. 맨체스터(Manchester) 코딩방식의 개념
   나. 맨체스터 인코딩 및 디코딩
   다. 맨체스터 코딩방식과 차등(Differential) 맨체스터 코딩방식 비교

4. 중앙처리장치(CPU) 내에 구성된 제어장치(Control Unit)의 구현 방법과 관련하여 아래 사항을 설명하시오.

가. micro-programmed 구현방법

나. hard-wired 구현방법

다. 구현 방법 간 상호비교

5. 객체지향 프로그래밍 기법을 활용한 소프트웨어 설계시 고려해야 할 원칙(일명 SOLID 원칙) 5 가지를 제시하고 설명하시오.

6. 정보통신산업진흥원에서 제시한 소프트웨어사업 영향평가에 대하여 아래 사항을 설명하시오.

가. 영향평가 대상기관

나. 소프트웨어사업 영향평가 체계

다. 평가항목

01	앰비언트 컴퓨팅(Ambient Computing)		
문제	앰비언트 컴퓨팅(Ambient Computing)은 인간의 개입이 없어도 시스템이 스스로 동작하는 기술이다. 이와 관련하여 아래 사항을 설명하시오. 가. 앰비언트 컴퓨팅의 개념 나. 개념도 및 기술요소 다. 앰비언트 컴퓨팅과 IoT(Internet of Things) 비교		
도메인	디지털서비스	난이도	중(상/중/하)
키워드	Zero UI, UI/UX, VR/AR, 5G		
출제배경	121회 기출 심화 문제		
참고문헌	ITPE기술사회		
해설자	전일 기술사(제 114회 정보관리기술사 / rosemachine@naver.com)		

I. 보이지 않는 IT 기술, 앰비언트 컴퓨팅의 개념

가. 앰비언트 컴퓨팅의 정의

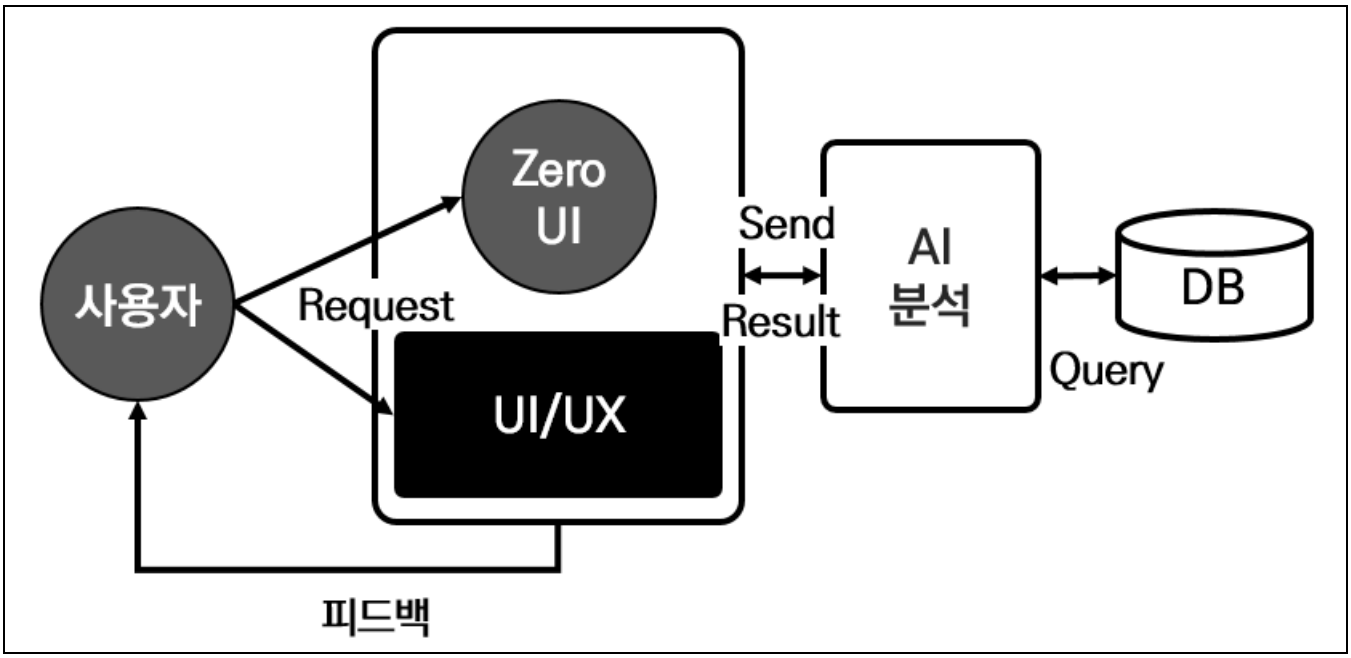
- IT 기기가 사용자의 일상에서 너무 자연스럽게 자리잡아, 인지되지 못한 상태로 활용되는 개념

나. 앰비언트 컴퓨팅의 등장배경

- IT 기기의 발달: 다양한 Device 를 통해 제공되는 서비스의 생활화
- 빠른 서비스: 서비스 제공까지의 소요시간 감소

II. 앰비언트 컴퓨팅의 개념도 및 기술요소

가. 앰비언트 컴퓨팅의 개념도



- 사용자가 IT 기술이라고 인지하지 못할 정도의 자연스러운 Device 활용

나. 앰비언트 컴퓨팅의 기술 요소

구 분	기술 요소	설 명
Front end	Zero UI	햅틱 피드백, 상황인식, 제스처, 음성인식기술 등 이용
	UI / UX	사용자 중심의 인터페이스를 통해 편의성 향상 서비스
	VR / AR	증강, 가상현실 기술을 통해 현실세계를 반영하는 서비스
Network	5G	초고속, 초저지연 기술을 통해 사용자에게 빠른 응답제공
분석기술	AI 기술	Deep Learning, ML 등을 이용하여 빠르고 정확한 분석 제공

- 서비스를 위한 입출력, 데이터 전송, 데이터 분석등의 기술들로 앰비언트 컴퓨팅 구현

III. 앰비언트 컴퓨팅과 IoT 비교

가. 앰비언트 컴퓨팅과 IoT 개념 비교

구분	앰비언트 컴퓨팅	IoT
정의	- 사용자가 특별히 의식하거나 조작하지 않아도 자연스럽게 환경 속에 이미 녹아 든 컴퓨팅	- 네트워크에 연결된 고유하게 식별 가능한 사물들(Things)이 인간의 명시적 개입 없이 상호 정보를 주고받으며 인간 중심적인 서비스를 제공할 수 있는 기반 인프라 기술
등장배경	- IoT 는 사용자가 직접 시나리오를 개발하고 기준 설정을 해야하는 번거로움 발생	- 이동통신망을 이용하여 사람과 사물, 사물과 사물 간 지능통신을 할 수 있는 M2M 의 개념을 인터넷으로 확장하여 사물은 물론, 현실과 가상세계의 모든 정보와 상호작용하는 개념으로 진화

나. 앰비언트 컴퓨팅과 IoT 상세 비교

구분	앰비언트 컴퓨팅	IoT
목적	- 사람 중심의 어느 순간에나 나타나 사용자의 별다른 노력 없이도 컴퓨팅 기술이 작동하는 것을 목적	- 사물 기기 간의 데이터를 주고받는 처리와 분석 등의 기술적인 부분에 초점
등장 시기	- 2010 년대 초반	- 2000 년대 초반
초기 유사 용어	- 유비쿼터스 컴퓨팅	- M2M
주요 기술	- Zero UI, UI / UX, VR / AR, 5G, AI	- 센싱, 유선통신, 무선통신, 접근제어, 인터페이스
활용 사례	- 아마존(Amazon)에서 구현한 알렉사(Alexa)	- 지능형 주차 서비스

- 앰비언트 컴퓨팅을 실행하는 기기는 자체적으로 축적한 데이터를 토대로 이용자의 상황과 상태를 판단하고 필요한 작업을 조용히 수행

IV. 앰비언트 컴퓨팅 향후 적용 분야

구분	적용 분야
스마트 홈/시티	- 주택 내의 다양한 기기와 시스템을 연결하여 편리한 생활 제공
의료분야	- 환자의 건강 상태를 실시간으로 모니터링하고 의료기기와 연결하여 의사들에게 중요한

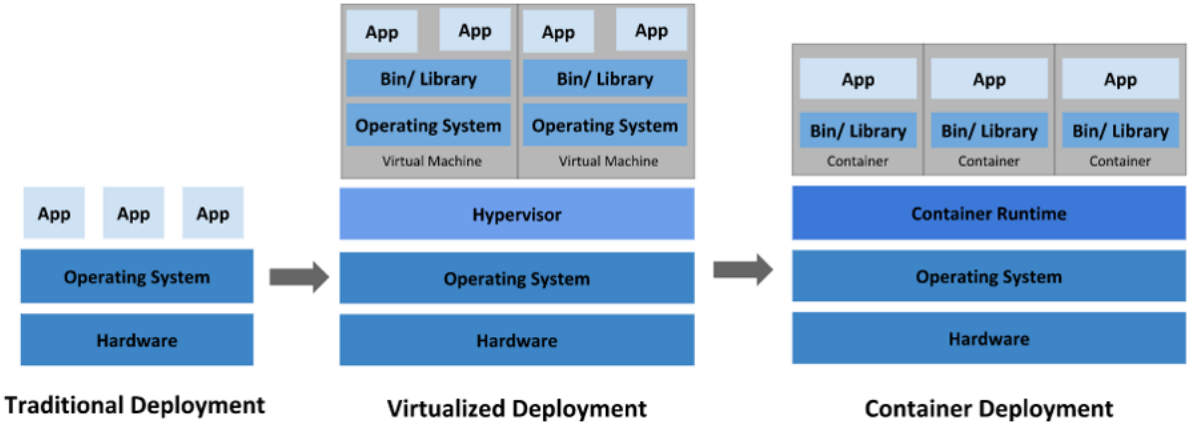
	정보를 제공
교육분야	- 개인화된 학습경험을 제공하고 학습자의 진도와 성과를 추적하여 최적의 학습 환경을 조성

- 대부분의 전문가들은 지금부터 20 년 후에는 앰비언트 컴퓨팅이 우리의 일상생활 속에 안착할 것이라고 전망

"끝"

02	가상머신(Virtual Machine) & 컨테이너(Container)		
문제	가상머신(Virtual Machine)과 컨테이너(Container)에 대하여 구체적으로 설명하고, 차이점을 상세히 설명하시오.		
도메인	디지털서비스	난이도	상(상/중/하)
키워드	하이퍼바이저, 전가상화, 반가상화, Type-1, Type-2, 컨테이너, LXC, libcontainer, Namespace, Cgroups		
출제배경	가상머신과 컨테이너 가상화 문제에 대한 반복 출제		
참고문헌	ITPE 서브노트		
해설자	단합반멘토 안경환 기술사(제 110회 정보관리기술사 / akh.itpe@gmail.com)		

I. 가상화 시스템 구조 발전 단계



- 하드웨어/운영체제/어플리케이션으로 구성된 전통적인 배치구조에서 하이퍼바이저 기반 가상화 방식과 컨테이너 런타임 기반 가상화 방식으로 진화

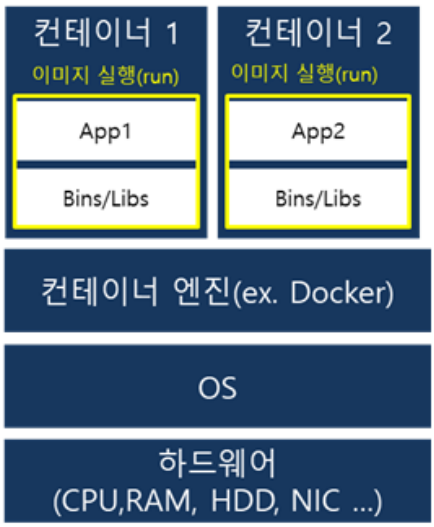
II. 가상머신(Virtual Machine)과 컨테이너(Container) 설명

가. 가상머신(Virtual Machine) 설명

개념	- 물리적 하드웨어 시스템(오프프레미스 또는 온프레미스에 위치)에 구축되어 자체 CPU, 메모리, 네트워크 인터페이스 및 스토리지를 갖추고 가상 컴퓨터 시스템으로 작동하는 가상 기술
개념도	

구성요소	하이퍼 바이저	- 호스트 컴퓨터에서 다수의 운영체제를 동시에 실행하기 위한 논리적 플랫폼으로, 여러 개의 OS가 단일 하드웨어 호스트를 공유할 수 있도록 하는 기술
	호스트 OS	- 실제 Server에 설치되는 OS
	게스트 OS	- 가상 머신에 설치되는 OS
가상화 방식	- 전가상화	- 하이퍼바이저를 통하여 기반 하드웨어를 공유 - OS를 수정하지 않고 실행 가능(단, OS가 기반 하드웨어를 지원해야 함)
	- 반가상화	- 가상화 인식을 OS에서 통합하여 사용 - 게스트 OS들이 하이퍼바이저에 맞게 수정하는 작업이 필요
HW 관리	- Type-1(Native/Bare Metal)	- 베어메탈 서버에 하이퍼바이저 기동
	- Type-2(Hosted)	- Host OS 에서 하이퍼바이저 기동

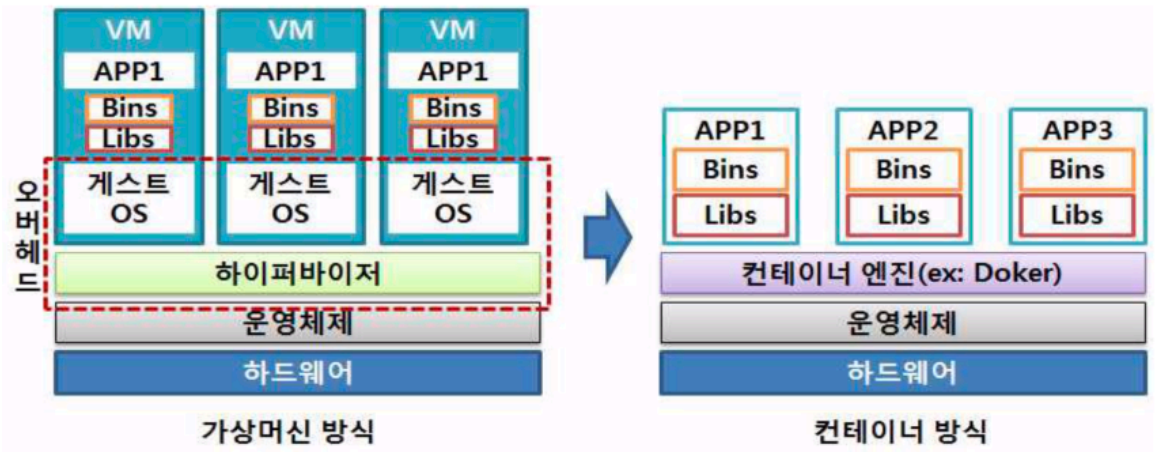
나. Container 설명

개념	- 데스크탑, 기존의 IT 또는 클라우드 등 어디서나 실행될 수 있도록 애플리케이션 코드가 해당 라이브러리 및 Dependency 항목과 함께 패키징 되어 있는 경량 가상화 기술	
개념도		
구성요소	컨테이너 엔진	- 운영환경을 고립시켜, 독립적인 공간을 제공하는 엔진
	컨테이너	- 컨테이너 엔진에 의해 생성된 가상의 격리된 애플리케이션 실행 공간
	호스트 OS	- 실제 Server에 설치되는 OS
가상화 방식	- LXC	- 리눅스 커널의 cgroups와 namespaces를 이용 컨테이너를 관리하는 모듈
	- libcontainer	- LXC, libvirt에 종속되지 않는 Docker 가상화 방식
HW 관리	- Namespace	- 운영환경을 고립시켜, 독립적인 공간을 제공하는 기능
	- Cgroups	- 프로세스 그룹에 대한 자원 제한, 격리, 모니터링 수행

- 최근 클라우드 컴퓨팅 사용 증가에 따라 컨테이너 기반의 가상화가 성능 측면에서 각광

III. 가상머신 방식과 컨테이너 방식 가상화 비교

가. 가상머신 방식과 컨테이너 기반 가상화 방식 구조 차이



- 하이퍼바이저와 게스트 OS를 포함하는 가상머신 방식은 상당한 오버헤드가 존재함
- 컨테이너 기반 가상화 구조는 애플리케이션의 실행에 필요한 라이브러리(Library, Libs), 바이너리(Binary Bins), 기타 구성파일 등을 패키지로 묶어서 배포하면, 구동환경이 바뀌어도 실행에 필요한 파일이 함께 따라다니기 때문에 오류를 최소화함

나. 가상머신 방식과 컨테이너 방식 비교

구분	가상머신 방식	컨테이너 방식
개념	- 가상머신을 사용하여 여러 운영체제 또는 환경을 동시에 실행할 수 있는 가상화 기술	- 실행에 필요한 모든 파일을 포함한 전체 실행(runtime) 환경에서 애플리케이션을 패키징하고 격리하는 기술
이식성	- VM당 모놀리틱한 서비스 - VM 단위의 이동, 복제와 생성 가능	- 실행에 필요한 모든 종속성 및 구성을 함께 배포(실행환경의 일관성) - 마이크로(Micro) 서비스 구축에 최적
효율성	- 1VM당 1서비스 - 성능 오버헤드 존재	- 호스트 OS 커널 공유이므로, 필요한 만큼 자원사용
서비스 요청 처리 신속성	- 최소 수 GB 이상의 추가 VM을 생성하여 대응	- 게스트 OS가 없는 수 MB 단위 컨테이너 생성
라이선스 비용	- VM 개수만큼 지불	- Host 1대의 비용만 지불
안정성	- 각각 독립된 VM들로 안정적인 운영 가능(완전한 분리)	- 통제된 영역이지만, OS 커널을 공유하므로, 장애발생시 같이 영향 받음

- 컨테이너 방식은 가상머신 방식에 비해 이식성, 효율성, 서비스 요청 처리 신속성, 라이선스 비용측면에서 우수하나, 안정성 부분에선 독립된 VM구조로 운영되는 가상머신 방식이 유리
- 컨테이너 런타임 기반 가상화 방식으로 진화하면서 다양한 분야에 적용 중

VI. 컨테이너 기반 가상화 적용분야

적용분야	설명
웹 애플리케이션 배포	- Docker와 Kubernetes와 같은 도구를 사용하여 애플리케이션을 컨테이너로 패키지화하고 확장 가능하게 배포
마이크로서비스 아키텍처	- 마이크로서비스는 독립적으로 컨테이너로 실행되며 필요한 경우 확장 가능
DevOps 프로세스	- 컨테이너는 개발자와 운영팀 간의 협력을 촉진하며 CI/CD 프로세스를 강화 - 코드 변경 사항을 컨테이너로 묶어서 더 빠르게 테스트하고 배포
데이터베이스 가상화	- 컨테이너를 사용하여 데이터베이스 가상화 - 테스트 및 개발환경을 관리하고 데이터베이스 인스턴스를 쉽게 배포

- 이식성, 효율성면에서 장점이 있는 컨테이너 기반 가상화 방식은 마이크로서비스 아키텍처, DevOps 프로세스 등 다양한 분야에 적용이 증가하고 있는 추세임 **"끝"**

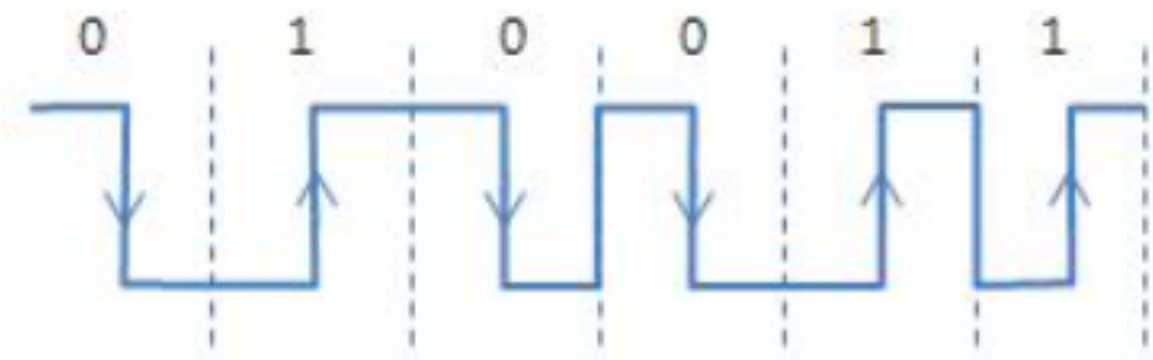
03	맨체스터 코딩(Manchester Coding)		
문제	맨체스터 코딩(Manchester Coding)은 데이터 저장과 디지털 데이터 통신 분야에서 다양하게 활용되는 코딩방식이다. 이와 관련하여 아래사항을 설명하시오. 가. 맨체스터(Manchester) 코딩방식의 개념 나. 맨체스터 인코딩 및 디코딩 다. 맨체스터 코딩방식과 차등(Differential) 맨체스터 코딩방식 비교		
도메인	네트워크	난이도	중(상/중/하)
키워드	클럭, 엣지, 신호변화		
출제배경	디지털 변환을 위한 기본적인 선로 부호 방식에 대한 이해		
참고문헌	ITPE 서브노트 [정보통신기술용어해설] https://asthtls.tistory.com/1004		
해설자	모멘텀 안수현 기술사(제119회 정보관리기술사 / tino1999@naver.com)		

I. 맨체스터(Manchester) 코딩방식의 개념

- 수신측 동기화의 용이성을 강조하도록, 비트 중간에 극성 변화가 있게 한, 선로부호 방식

II. 맨체스터 인코딩 및 디코딩

가. 맨체스터 인코딩



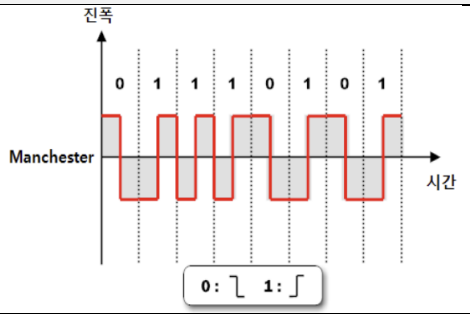
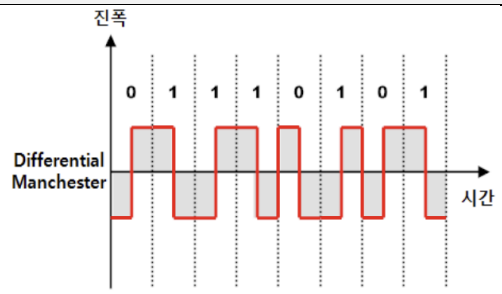
용도	데이터를 전송하기 위한 디지털 통신에서 사용
신호특성	"1"과 "0" 비트 각각에 대해 상승과 하강 에지를 사용하여 신호를 표현
클럭추출	에지의 간격을 이용하여 클록 신호를 추출
신호 변화	- 비트 중간에서, 하향 천이 => `0` - 비트 중간에서, 상향 천이 => `1`

- 맨체스터 신호의 극성은, 각 비트의 중앙에서 반전되며, 천이 방향이 논리상태를 결정

나. 맨체스터 디코딩

구분	설명
용도	맨체스터로 인코딩된 데이터를 수신하여 실제 데이터를 추출하기 위해 사용
신호해석	상승 에지에서 시작하는 경우 "1" 비트로 해석하고, 하강 에지에서 시작하는 경우 "0" 비트로 해석
클럭추출	에지 간의 간격을 이용하여 클록 신호를 추출
신호 해석 유연성	데이터가 변할 때마다 에지가 발생하므로 신호의 변화를 감지하기 쉬움

III. 맨체스터 코딩방식과 차등(Differential) 맨체스터 코딩방식 비교

항목	맨체스터	차등 맨체스터
개념도		
신호 표현방법	- "1" 비트는 상승 에지로 시작하여 하강 에지로 종료 - "0" 비트는 하강 에지로 시작하여 상승 에지로 종료	- "1" 비트는 상승 에지에서 시작하여 상승 에지로 종료 - "0" 비트는 하강 에지에서 시작하여 하강 에지로 종료
특징	- 각 비트마다 반드시 에지가 발생하므로 클록 동기화가 간편 - 전압이 변하는 지점이 비트의 중간이므로 DC 컴포넌트가 없음	- 각 비트 사이에 반드시 에지가 존재하지 않으므로 클록 동기화가 어려움 - DC 컴포넌트가 없어짐으로써 전압의 변화가 더 미세함.
클록 복구	클록 신호를 추출하기 쉬움	에지 간의 간격이 일정하지 않아 클록 추출이 어려울 수 있음

"끝"

04	제어장치(Control Unit)		
문제	중앙처리장치(CPU) 내에 구성된 제어장치(Control Unit)의 구현 방법과 관련하여 아래 사항을 설명하시오. 가. micro-programmed 구현방법 나. hard-wired 구현방법 다. 구현 방법 간 상호비교		
도메인	컴퓨터 구조	난이도	중(상/중/하)
키워드	RISC, CISC, 신호 제어 시퀀스, 명령 수행, 조합회로, PLD, HW방식		
출제배경	CPU 구성요소 중 하나인 제어장치에 대한 구현방법 이해 점검		
참고문헌	ITPE 기술사회		
해설자	BP반 김찬일 기술사(제 130회 정보관리기술사 / s2carey@naver.com)		

I. CPU의 명령어장치, 제어장치의 개요

가. 제어장치(Control Unit)의 정의

- 프로그램 명령어를 해석하고, 실행하기 위한 제어 신호를 순차발생, 전달하는 CPU 구성요소

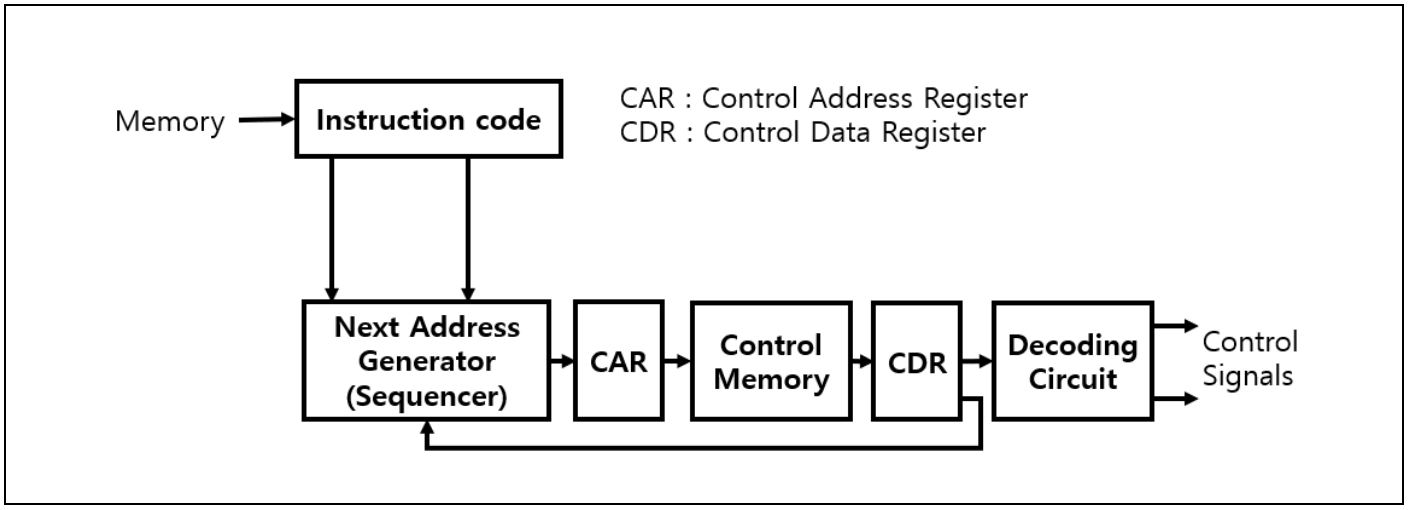
나. 제어장치의 기능

구분	내용
명령어 해독	명령어의 연산 부호를 해독한다.
제어 신호 생성	명령어의 연산 부호를 해독한 결과와 기능 필드, 상태 레지스터, 외부 신호등과 같은 정보를 사용하여 명령어 실행에 필요한 제어 신호를 생성
제어 신호 인가	데이터 경로를 구성하는 각종 장치에 생성된 제어 신호를 인가함으로써 명령어가 지시하는 작업을 수행

- 제어장치는 구현방법이 크게 마이크로 프로그래밍 방식과 하드웨어 방식의 구현 방법으로 구분되어 짐.

II. micro-programmed 구현방법

가. micro-programmed 구현방법의 개념



개념	- 제어신호들의 조합을 미리 구성해 ROM 등에 프로그램으로 저장하여 필요시 신호발생, CISC 구조에 사용되는 방법

- 하드웨어 설계를 소프트웨어로 추상화 하여 제어 신호를 생성하고, 이를 이용해 명령어를 실행하는 목적

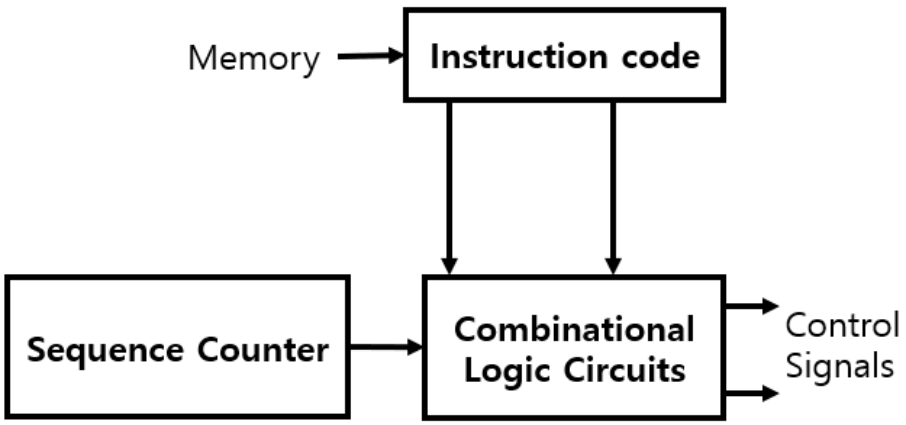
나. micro-programmed 구현방법의 상세설명

구분	항목	내용
설계	- 명령어 세트 설계	- 명령어는 하드웨어 제어 신호로 변환될 것이므로, 각 명령어에 대한 동작과 제어 신호를 명확히 정의
	- 마이크로 명령어 집합 설계	- 각 명령어에 대한 마이크로 프로그램을 작성하기 위해 마이크로 명령어 집합을 설계
	- 마이크로 프로그램의 작성	- 명령어별로 제어 신호를 생성하는 마이크로 프로그램을 작성
구현	- 마이크로 메모리 구현	- 마이크로 메모리는 ROM (Read-Only Memory) 또는 RAM (Random Access Memory)으로 구현
	- 제어 유닛에 통합	- 마이크로 프로그램이 작성되고 마이크로 메모리에 저장되면, 이를 제어 유닛과 통합
최적화	- 디버깅 및 최적화	- 구현된 마이크로 프로그램을 디버깅하고 최적화하는 과정을 수행하여 올바르게 동작하고 최적 성능 발휘 보장

- 마이크로 프로그래밍과 대조적으로 제어 유닛의 동작을 논리 게이트와 회로로 직접 구현하는 방식이 존재

III. hard-wired 구현방법

가. hard-wired 구현방법 개념



개념	- 제어장치의 기능을 조합회로를 이용하여 Hardware 로 구현, RISC 구조에 사용되는 구현방법

- 제어 유닛의 동작을 논리 게이트와 회로로 직접 구현하여 빠르게 작동되어지는 하드웨어적 구현방법

나. hard-wired 구현방법의 상세설명

구분	항목	내용
설계	- 명령어 세트 설계	- 마이크로 프로그래밍과 마찬가지로 하드웨어 구현도 명령어 세트에 기반함. 각 명령어는 특정 동작을 수행하도록 설계
	- 유한 상태 기계 설계	- 유한 상태 기계는 시스템이 특정 상태에서 다른 상태로 전이하도록 제어하는 데 사용
	- 논리 회로 설계	- 명령어의 동작을 구현하기 위해 논리 게이트를 사용하여 논리 회로를 설계
구현	- 시퀀셜 논리 회로 사용	- 하드웨어 구현에서는 주로 시퀀셜 논리 회로를 사용 이는 현재 상태와 입력에 따라 다음 상태와 출력을 결정하는 회로
	- 제어 신호 생성	- 논리 회로는 각 명령어에 대한 제어 신호를 생성
	- 회로 통합	- 설계한 논리 회로를 통합하여 전체 제어 유닛을 형성
최적화	- 디버깅 및 최적화	- 완성된 하드웨어 제어 유닛을 디버깅하고 최적화하는 과정

- hard-wired 구현방법과 micro-programmed 구현방법은 여러 특징 및 항목별 차이점들이 존재함.

IV. 구현방법간 상호 비교

구분	**micro-programmed 구현방법**	**hard-wired 구현방법**
구현	- 메모리에 저장된 마이크로 코드	- 고정된 놀리 게이트 및 회로 세트
유연성	- 더욱 유연하고 수정이 쉬움	- 유연성이 떨어지고 수정이 어려움
명령어 세트	- 복잡한 명령어 세트 지원	- 제한된 명령어 세트 지원
복잡성	- 복잡한 디자인, 구현 어려움	- 심플한 디자인, 구현이 용이함
속도	- 디코딩으로 인해 작동속도 느려짐	- 빠른 작동
디버깅	- 디버그 및 테스트가 쉬움	- 디버그 및 테스트가 어려움
규모 비용	- 더 큰 크기, 높은 비용	- 더 작은 크기, 더 낮은 비용
유지 관리	- 업그레이드 및 유지 관리 쉬움	- 업그레이드 및 유지 관리가 어려움

- 하드웨어 구현은 일반적으로 마이크로 프로그래밍보다 성능이 뛰어남, 설계 및 변경 복잡 비용 상승 가능함.

"끝"

05	SOLID 원칙		
문제	객체 지향 프로그래밍 기법을 활용한 소프트웨어 설계 시 고려해야 할 원칙(일명 SOLID 원칙) 5가지를 제시하고 설명하시오.		
도메인	소프트웨어공학	난이도	하(상/중/하)
키워드	SRP (Single Response Principle), OCP (Open Closed Principle), LSP (Liskov Substitution Principle), ISP (Interface Segregation Principle), DIP (Dependency Inversion Principle)		
출제배경	객체지향 프로그래밍의 기본 확인		
참고문헌	ITPE 기술사회 자료집		
해설자	정상반 정상 기술사(제 124회 정보관리기술사 / itpe_peak@naver.com)		

I. 재사용성과 유지보수 향상을 위한, 객체지향 설계 5대 원칙의 개요

가. 재사용성을 극대화 하는 객체지향 설계의 개념

- **소프트웨어 개발 및 유지보수성 향상을 위한 설계관점의 기본원칙**
- 코드를 좀더 유지보수하기 쉽고, 유연하고, 확장하기 쉽게 만들기 위한 설계

나. 객체지향 설계의 특징

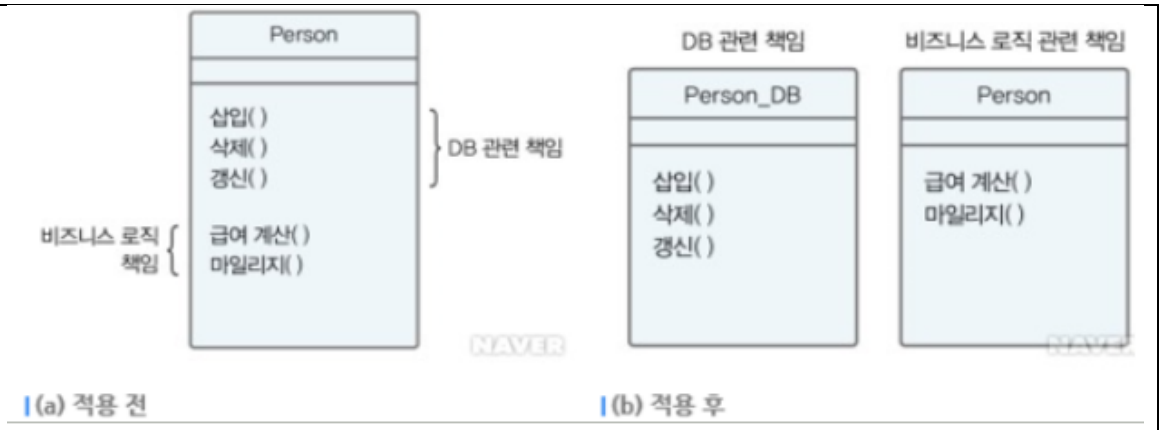
특징	설명
생산성 향상	- 재사용성, 유지보수성, 이식성 통한 생산, 품질의 향상
모형의 적합성	- 현실 세계 및 인간 사고 방식과 유사
일관성, 추적성	- 전체 공정에서 각 단계간의 전환과 변경이 자연스럽고 신속함

- 재 사용성, 유지 보수성, 이식성 통한 생산성 및 품질의 향상
- 모형의 적합성: 현실 세계 및 인간의 사고 방식과 유사
- 일관성, 추적성: 전체 공정에서 각 단계간의 전환과 변경이 자연스럽고 신속함

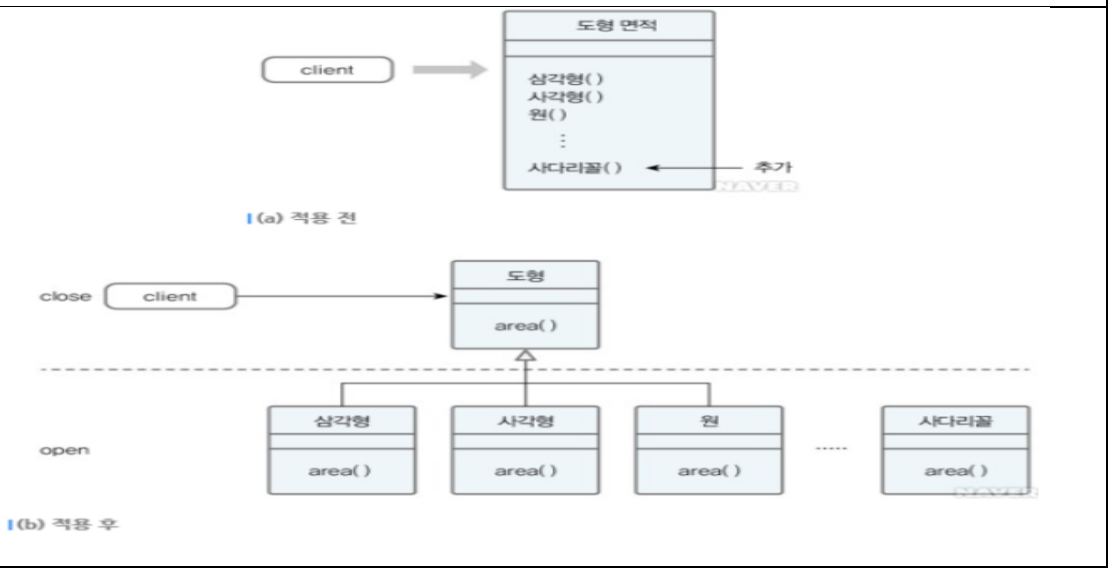
II. SRP 와 OCP 설명

가. 단일책임의 원칙 (SRP : Single Response Principle)

구분	설명
정의	- 시스템의 모든 객체는 하나의 책임만을 가지며, 객체가 제공하는 모든 서비스는 그 하나만의 책임만을 수행해야 한다는 설계 원칙
특징	- 응집도 향상으로 유지보수성 향상 - 하나의 책임만을 수행하기 때문에 변화에 적응도 높음 - 두 개 이상의 책임을 가지면, 온전한 책임을 다할 수 있도록 분리

예시	
예시 설명	- (a)는 단일책임 원칙을 따르지 않은 클래스로 메서드가 DB관련 메서드들과 비즈니스 관련 메서드들로 구성되어 있음. - 단일책임의 원칙을 지키기 위해 (b)와 같이 각각의 책임을 갖는 두 개의 클래스로 분할

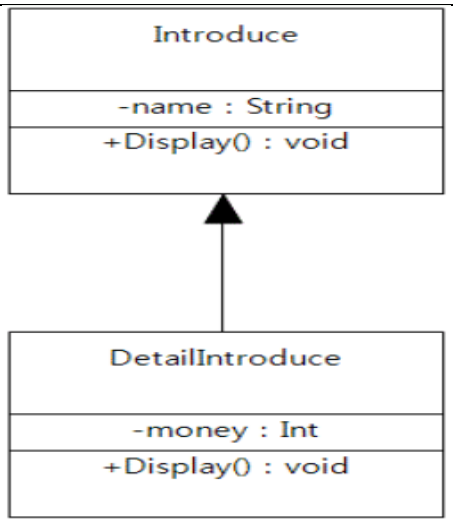
나. 개방 폐쇄 원칙 (OCP : Open Closed Principle)

구분	설명
정의	- 소프트웨어 Entity(classes, Modules, Function)는 **확장에는 열려있고 수정에는 닫혀있어야 한다**는 설계 원칙
특징	- 기존 코드의 변경 없이 확장을 통한 코드의 변경을 허용 - 오버라이딩 - 상속을 의미하지만, 크게 유연석 확보 차원의 원리 - 기능의 상속이 아닌 설계의 유연성을 강조 Open(클래스 수직관계), Close(클래스 수평관계) - Strategy 패턴 : 인터페이스 변경은 어려우나 구현은 열려 있음
예시	

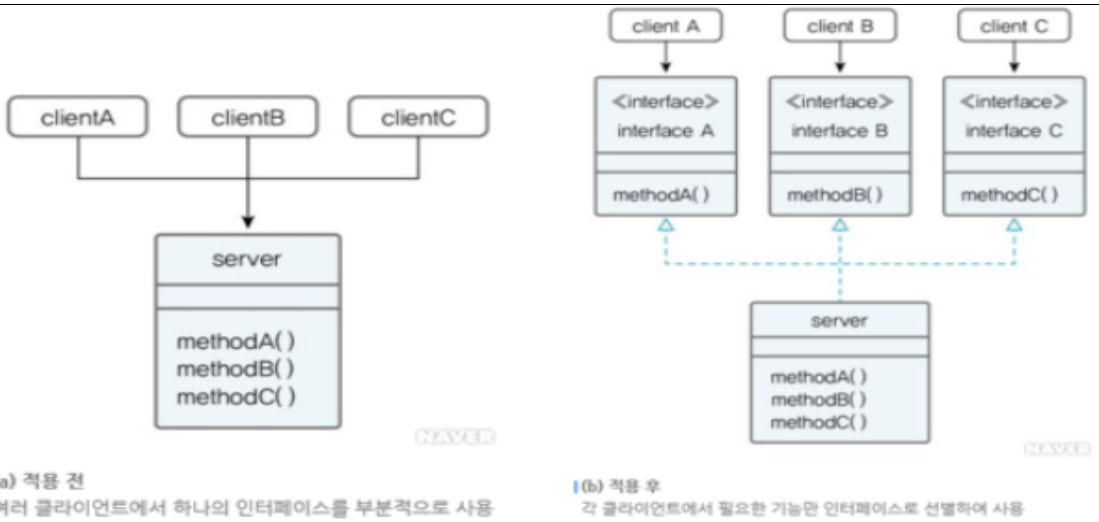
예시 설명	- 원칙을 따르지 않고 설계 시, 삼각형, 사각형, 원 등의 면적을 구하려면 (a)와 같이 각 도형의 면적을 구하는 메서드가 하나의 클래스에 모두 존재하게 됨. - 상속의 개념을 적용하여 (b)와 같이 설계하면 클라이언트가 추상클래스에 의존하므로 구체 클래스의 내부 변경에 영향을 받지 않음.

III. LSP와 ISP 설명

가. 리스코프 치환의 원칙 (LSP :Liskov Substitution Principle)

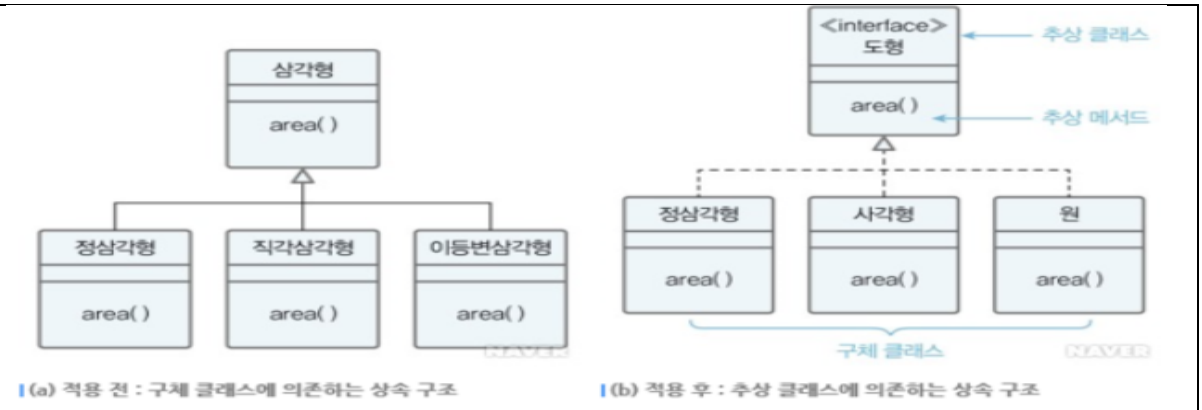
구분	설명
정의	**부모 클래스의 객체(타입과 매소드의 집합)들이 자식 클래스 사용되는 곳에 대체**될 수 있어야 한다는 설계 원칙 that objects of a superclass shall be replaceable with objects of its subclasses without breaking the application.
특징	- 클래스의 생성 목적에 맞게 설계하기 때문에 상/하위 클래스의 호환성 향상 - 하위 클래스는 상위 클래스의 책임을 넘지 않음 - 하위 클래스는 사용자의 요구 사항대로 설계됨.
예시	
예시 설명	- LSP를 위배하는 예제 - Introduce를 상속한 DetailIntroduce 클래스가 함수 q가 요구하는 행동을 하지 않고, 엉뚱한 소리를 하고 있음 - 위의 예는 error 클래스를 만들때 주로 나타날 수 있음 - 하위 클래스의 행동 또한 상위 클래스의 책임범위 내에서 생성되어야 함

나. 인터페이스 분리의 원칙(ISP/Interface Segregation Principle)

구분	설명
정의	- 어떤 클래스가 **다른 클래스에 종속될 때는 최소한의 인터페이스만을 사용해야 한다**는 설계 원칙
특징	- 인터페이스의 단일 책임을 강조 - 하나의 인터페이스에 해당 인터페이스의 목적에 부합되지 않는 기능을 선언하지 않기 때문에 구현 클래스에서 불필요한 기능의 구현 방지 - 소스 레벨의 가독성 향상
예시	
예시 설명	- 클라이언트가 사용하지 않는 인터페이스 때문에 영향을 받아서는 안 된다는 것. - 클래스는 사용자에게 꼭 필요한 메서드만 갖는 인터페이스를 제공 - 인터페이스 분리의 원칙은 (b)처럼 각 클라이언트에 맞는 인터페이스만 분리하여 사용

IV. 의존성 역전의 원칙(DIP/Dependancy Inversion Priciple)

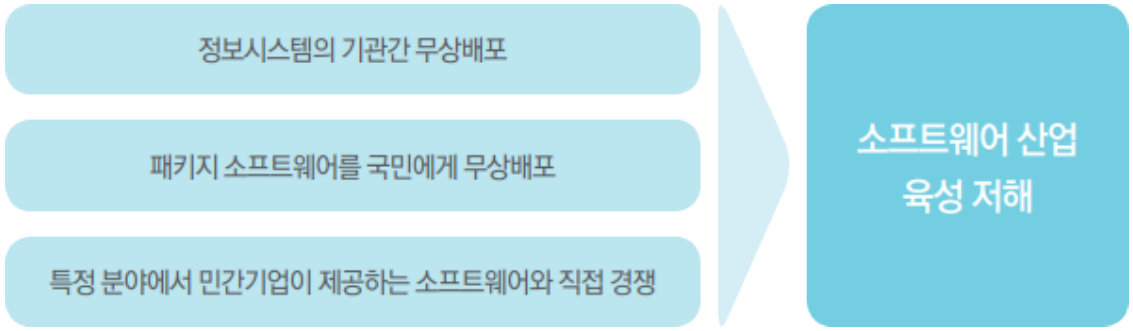
구분	설명
정의	- 높은 레벨의 모듈은 낮은 레벨의 모듈을 의존하면 안되며 **서로 추상에 의존해야 한다**는 설계 원칙 - 클라이언트 변경 최소화를 위해 클라이언트는 구체 클래스가 아닌 인터페이스나 추상 클래스에 의존한다는 설계 원칙
특징	- 구현 클래스에 의존성이 제거하여 낮은 결합도 유지 - 추상화된 클래스에 의존하기 때문에 확장성이 높아져서 유지보수성 향상

예시	 (a) 적용 전 : 구체 클래스에 의존하는 상속 구조 (b) 적용 후 : 추상 클래스에 의존하는 상속 구조
예시 설명	- 의존관계 역전의 원칙은 상속 개념을 적용할 때 (a)와 같이 구체 클래스에서 상속을 받는 구조로 설계하지 말라는 것, 이유는 구체 클래스는 추상 클래스(인터페이스)보다 변하기 쉽기 때문 - 의존 관계를 구체 클래스에서 추상 클래스로 바꾸면 향후 변경에 따른 영향을 최소화할 수 있고, 느슨한 결합도를 유지할 수 있다. - (b)처럼 변화에 따른 충격에서 자유롭도록 추상 클래스를 만들고 그 추상 클래스와 상속 관계를 유지하도록 설계

"끝"

06	소프트웨어사업 영향평가		
문제	정보통신산업진흥원에서 제시한 소프트웨어사업 영향평가에 대하여 아래 사항을 설명하시오. 가. 영향평가 대상기관 나. 소프트웨어사업 영향평가 체계 다. 평가항목		
도메인	소프트웨어공학	난이도	중 (상/중/하)
키워드	「소프트웨어산업 진흥법」 제21조의4항, 「소프트웨어산업 진흥법 시행령」제17조2, 평가 기준, 평가 방법, 평가 절차, 민간 소프트웨어 시장 침해가능성, 소프트웨어사업의 필요성·공공성		
출제배경	소프트웨어 사업 영향평가에 대한 개정 및 128회 응용 출제 문제에 대한 반복 심화 출제		
참고문헌	ITPE 서브노트		
해설자	단합반멘토 안경환 기술사(제 110회 정보관리기술사 / akh.itpe@gmail.com)		

I. 민간 SW시장 및 SW 산업 생태계 영향을 평가 개선하는, SW 영향평가제도의 개요

개념	- 국가기관 등에서 소프트웨어사업의 예산편성, 발주, 소프트웨어 배포 및 서비스 제공을 추진하는 경우 민간 소프트웨어 시장 침해 등 소프트웨어 산업 생태계에 미치는 영향을 검토하여 사전 조정하는 제도
도입배경	

- 국가기관 등이 소프트웨어를 무상 배포함에 따라 민간 소프트웨어 시장을 위축시켜 소프트웨어 산업 육성을 저해한다는 문제가 지속 제기
- 업무효율성 증진, 비용절감, 대국민 서비스 향상 등의 이유로 소프트웨어를 개발한 후 무상 배포함에 따라 민간 소프트웨어 시장 위축

II. 소프트웨어사업 영향평가 대상기관

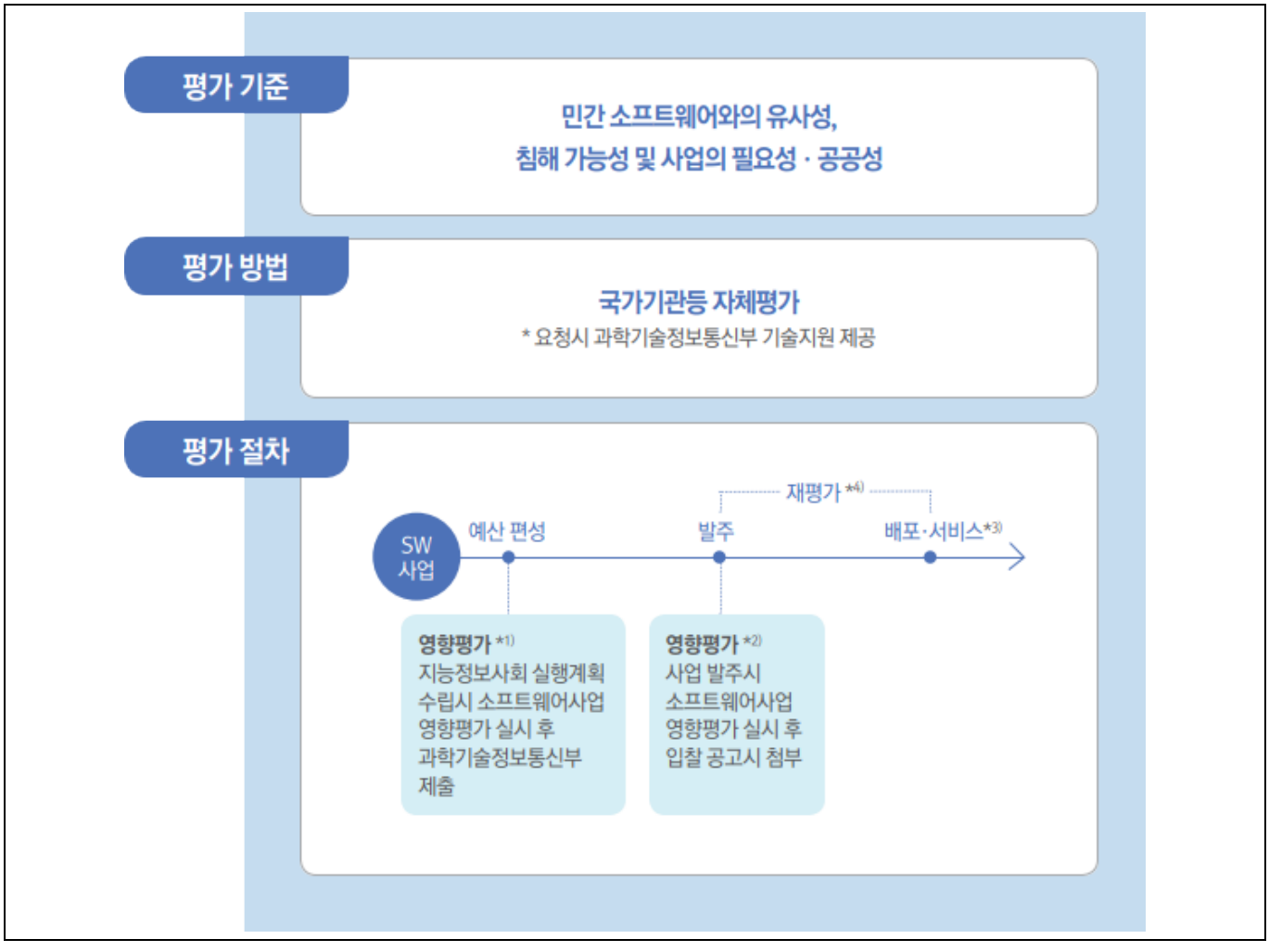
대상	법령
「소프트웨어산업 진흥법」 제21조의4항의 기관	- 제21조(소프트웨어프로세스 품질인증) ④ 국가기관, 지방자치단체, 국가 또는 지방자치단체가 투자하거나 출연한 법인·단체 등으로서 대통령령으로 정하는 기관(이하 "국가기관등"이라 한다)의 장은 소프트웨어사업을 추진하는 경우 소프트웨어 산출물의 품질 제고를 위하여 대통령령으로 정하는 바에 따라 소프트웨어프로세스 품질인증을 받은 자를 우대할 수 있다.
「소프트웨어산업 진흥법 시	- 제21조(국가기관등의 범위) 법 제21조제4항에서 "대통령령으로 정하는 기관

행령」 제17조2 규정에 해당하는 모든 기관	"이란 다음 각 호의 기관 1. 국가기관 2. 지방자치단체 3.「공공기관의 운영에 관한 법률」 제4조에 따른 공공기관 4. 정부가 자본금을 출자하여 최대지분을 보유하고 있는 기관 또는 단체. 다만, 정부가 자본금의 2분의 1 미만을 출자한 금융기관은 제외 5.「정부출연연구기관 등의 설립·운영 및 육성에 관한 법률」 제2조에 따른 정부출연연구기관 또는 「과학기술분야 정부출연연구기관 등의 설립·운영 및 육성에 관한 법률」 제2조에 따른 과학기술분야 정부출연연구기관 6. 법령에 따라 정부로부터 출연금을 받는 기관 또는 단체 7.「지방공기업법」에 따라 지방자치단체가 자본금의 2분의 1 이상을 출자한 지방공사 또는 지방공단

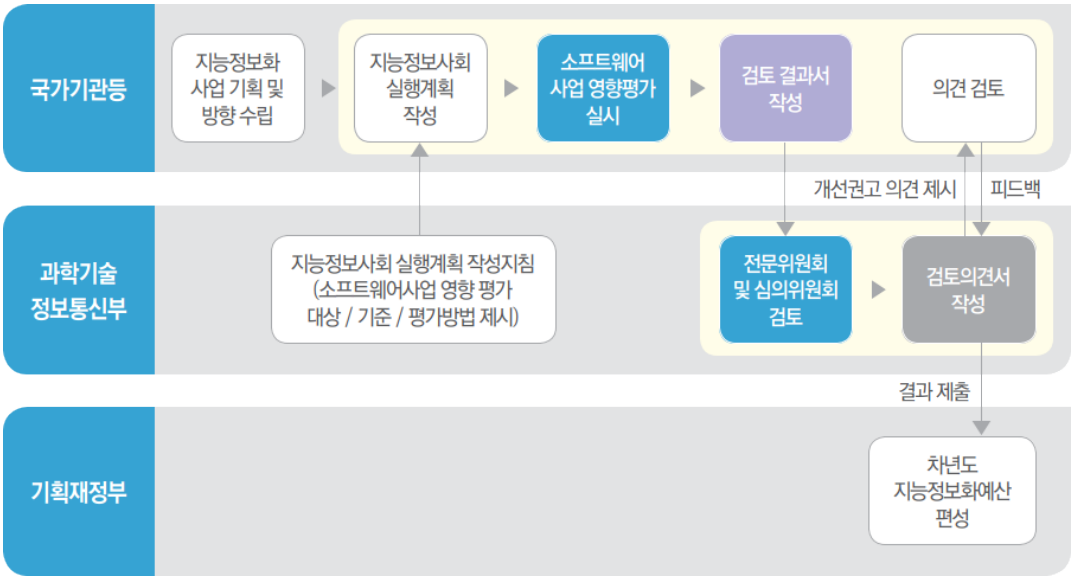
- 소프트웨어산업 진흥법 제 21조에서 대상 기관으로 국가기관, 지방자치단체, 대통령령으로 정한 기관으로 정의하고 소프트웨어산업 진흥법 시행령 제21조에서 세부 사항을 정의

III. 소프트웨어사업 영향평가 체계

가. 소프트웨어사업 영향평가 체계 구조도



나. 소프트웨어사업 영향평가 체계 구성

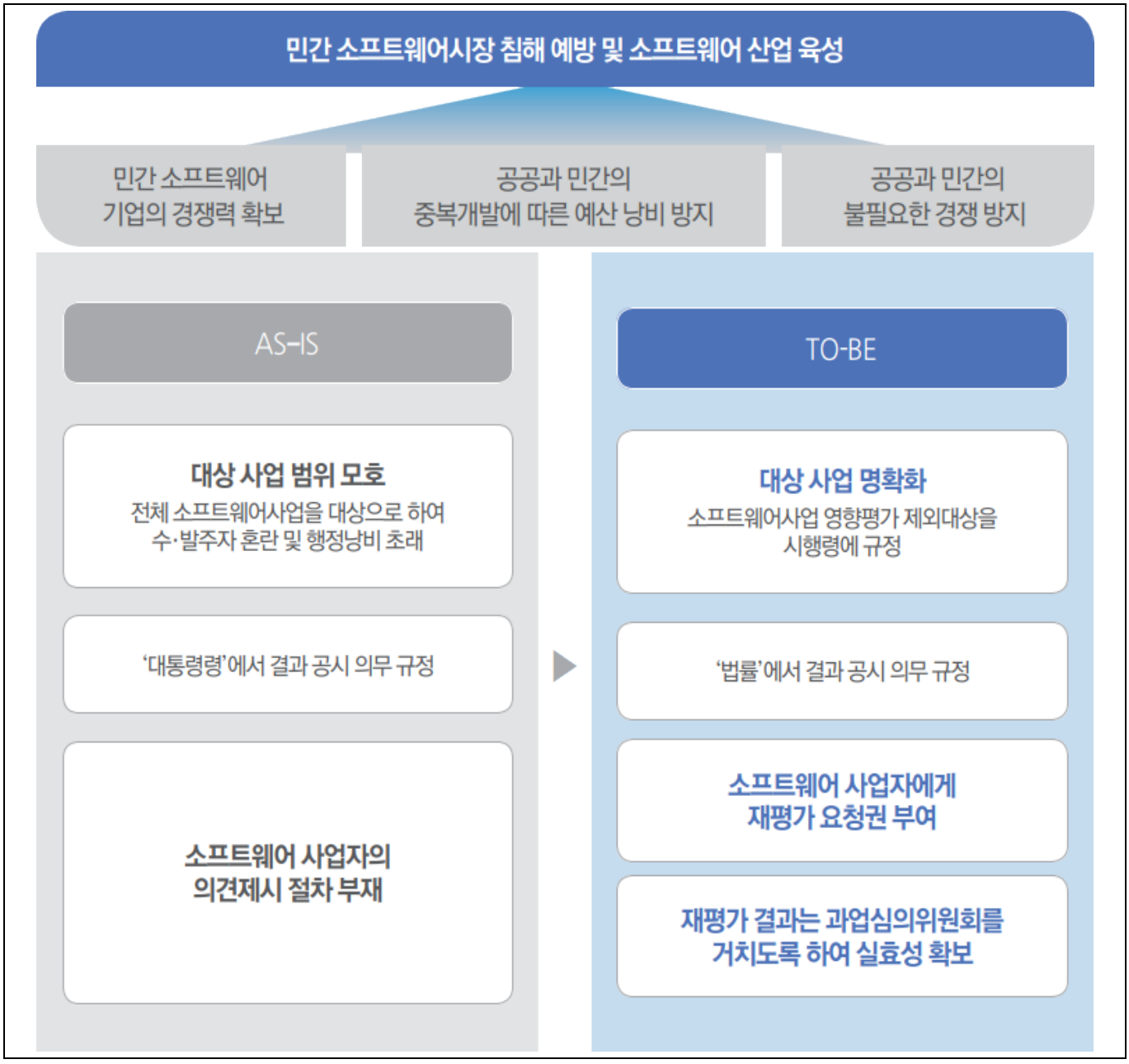
구성 요소	세부 요소	설명
평가 기준	- 평기 항목	- 소프트웨어사업 영향평가 대상 사업에 대한 민간 소프트웨어 시장 침해가능성과 소프트웨어사업의 필요성·공공성을 종합적으로 평가
	- 평가 결과	- 민간시장 침해 가능성의 유무
평가 방법	- 자체평가	- 국가기관등의 사업부서 또는 정보화부서에서 소관 소프트웨어 사업에 대하여 소프트웨어사업 영향평가 검토결과서를 작성
	- 기술지원	- 국가기관등은 소프트웨어사업 영향평가를 위하여 과학기술정보통신부·정보통신산업진흥원에 기술지원 요청 가능
평가 절차		

- 소프트웨어사업 영향평가를 위해 민간 시장의 침해 가능성과 공공성 등의 다면 측정을 통해 평가 수행

IV. 소프트웨어사업 영향 평가항목

평가 항목	설명	
민간 소프트웨어 시장 침해가능성	- 소프트웨어사업의 주요기능과 동일·유사한 소프트웨어를 민간에서 제공하는지 여부	
소프트웨어사업의 필요성·공공성	- 민간 소프트웨어가 있음에도 불구하고 사업을 추진해야 하는 필요성·공공성을 기재	
	- (사례) 사업 필요성·공공성이 높은 경우	- (주요 정책 사업 등) 일자리창출, 00페이 등 일반 사회 구성원 전체에 영향을 주는 등 필요성·공공성이 높은 사업
	- (사례) 사업 필요성·공공성이 낮은 경우	- (무상배포) 00설계프로그램 무상 배포, 메신저 SW 무상 배포 등과 같이 공공기관에서 개발한 프로그램을 민간에 무상으로 배포하여 민간 소프트웨어 시장을 침해하는 사업

V. 소프트웨어사업 영향평가를 통한 기대 효과



"끝"

ITPE 기술사회
제132회 정보처리기술사 기출문제 해설집

대 상 정보관리기술사, 컴퓨터시스템응용기술사, 정보통신기술사, 정보시스템감리사 시험
발행일 2024년 08월 27일
집 필 강정배PE, 안경환PE, 정상PE, 안수현PE, 김찬일PE, 전일PE
출 판 **ITPE(Information Technology Professional Engineer)**
주 소 ITPE 대치점 서울시 강남구 선릉로 86길 17 선릉엠티빌딩 7층
ITPE 선릉점 서울시 강남구 선릉로 86길 15, 3층 IT교육센터 아이티피이
ITPE 강남점 서울시 강남구 테헤란로 52길 21 파라다이스벤처타워 3층 303호
ITPE 영등포점 서울시 영등포구 당산동2가 하나비즈타워 7층 ITPE
연락처 070-4077-1267 / itpe@itpe.co.kr