教 育 資 料
90 – 02 – 004

소 프 트 웨 어 공 학

Software Engineering

經濟企劃院 調査統計局

목 차

소프트웨어공학 서설

Introduction to Software Engineering

소프트웨어의 중요성

▷ THE NEW DRIVING FORCE

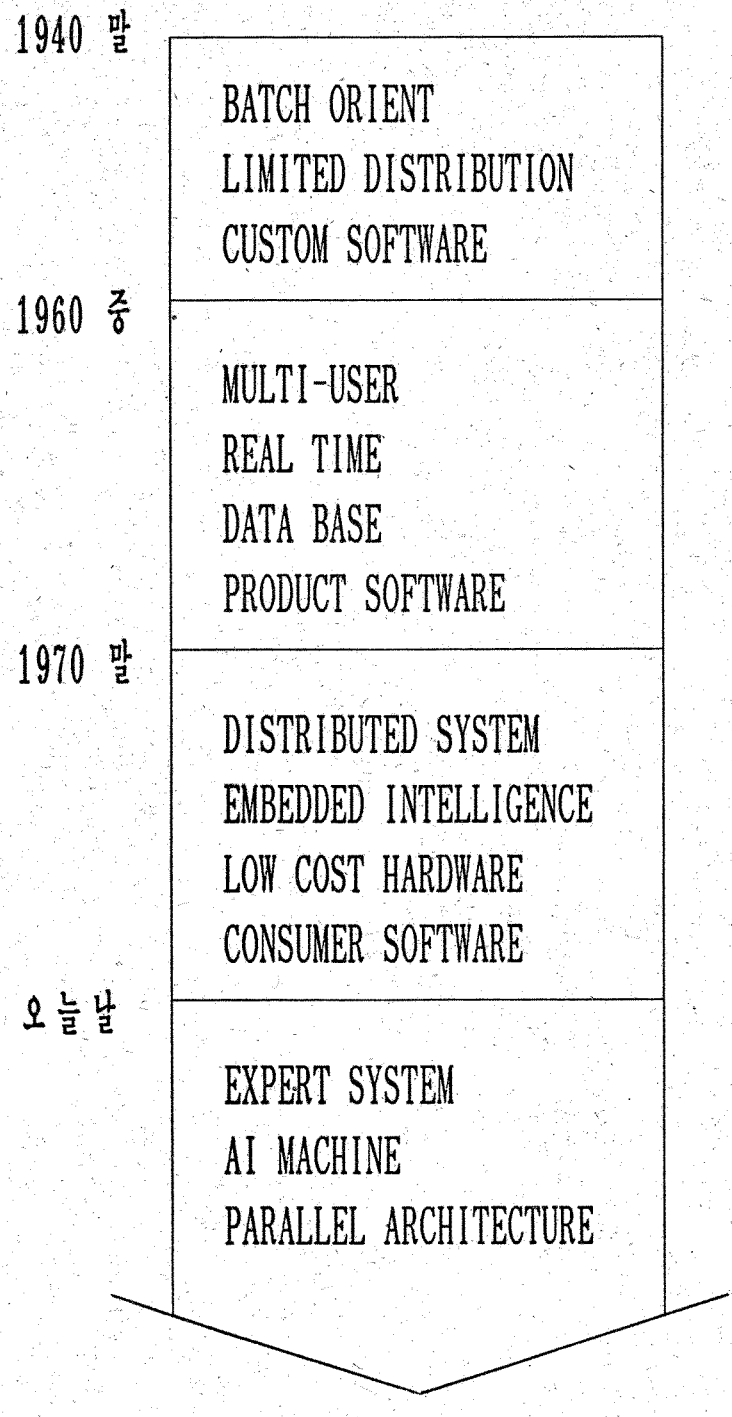
▷ THE NEW INDUSTRIAL REVOLUTION

▷ 산업사회에서 정보사회로

▷ 생산품 및 회사의 질을 결정하는 KEY FACTOR

▷ 자동화 체계중 가장 큰 예산항목

소프트웨어의 발달사


1940 말
BATCH ORIENT
LIMITED DISTRIBUTION
CUSTOM SOFTWARE
1960 중
MULTI-USER
REAL TIME
DATA BASE
PRODUCT SOFTWARE
1970 말
DISTRIBUTED SYSTEM
EMBEDDED INTELLIGENCE
LOW COST HARDWARE
CONSUMER SOFTWARE
오늘날
EXPERT SYSTEM
AI MACHINE
PARALLEL ARCHITECTURE

소프트웨어란 ?

▷ SET OF INSTRUCTION

▷ DATA STRUCTURE

▷ DOCUMENTS

소프트웨어의 특성

▷ LOGICAL : PHYSICAL

▷ DEVELOPMENT : MANUFACTURING

▷ INTERFACE의 OBSCURITY

응용분야에 따른 소프트웨어의 분류

- SYSTEM SOFTWARE
- REAL TIME SOFTWARE
- BUSINESS SOFTWARE
- ENGINEERING AND SCIENTIFIC SOFTWARE
- EMBEDDED SOFTWARE
- PERSONAL COMPUTER SOFTWARE
- ARTIFICIAL INTELLIGENCE SOFTWARE

SOFTWARE CRISIS

▷ COMPUTER SOFTWARE의 개발중 당면하는 문제들의 집합

- 시간 및 비용 예측의 부정확성
- 시스템 개발후 사용자의 불만족
- 소프트웨어 품질의 낮은 신뢰성
- 기존 소프트웨어의 유지보수 어려움

▷ SOFTWARE CRISIS의 주요 원인

- 소프트웨어의 자체 특성
- HUMAN FAILINGS

SOFTWARE에 관한 고정관념

▷ 관리자의 고정 관념

- 기존 소프트웨어 개발 접근 방법의 고수
- 소프트웨어의 개발도구 보다 최신 하드웨어에 더 관심
- 지연되는 계획에 더 많은 인원 투여하여 해결

▷ 고객(사용자)의 고정 관념

- 초기단계에서 목표에 대한 일반적인 기술로 충분
- 새 요구에 대한 수시 변경이 가능하다 생각

▷ 프로그래머의 고정 관념

- 분석, 설계 및 테스팅을 위한 방법의 부재
- 소프트웨어 구현전까지 품질평가 불가능
- 마구잡이식 유지보수

소프트웨어 공학

▷ 실 기계에서 효과적으로 작동하는, 신뢰성 있고 경제적인 소프트웨어를 구축하기 위해 공학적 원칙을 수립하고 적용하는 것

▷ 소프트웨어의 개발과, 조작, 유지보수를 위한 체계적 접근 방법

▷ 예측 비용과 시간 내에, 소프트웨어 생산품을 체계적으로 생산하고 유지보수하기 위한 기술적이고 관리적인 원칙

소프트웨어 공학의 요소

▷ METHODOLOGY

소프트웨어를 구축하기 위한 기술적 방법론

▷ TOOL

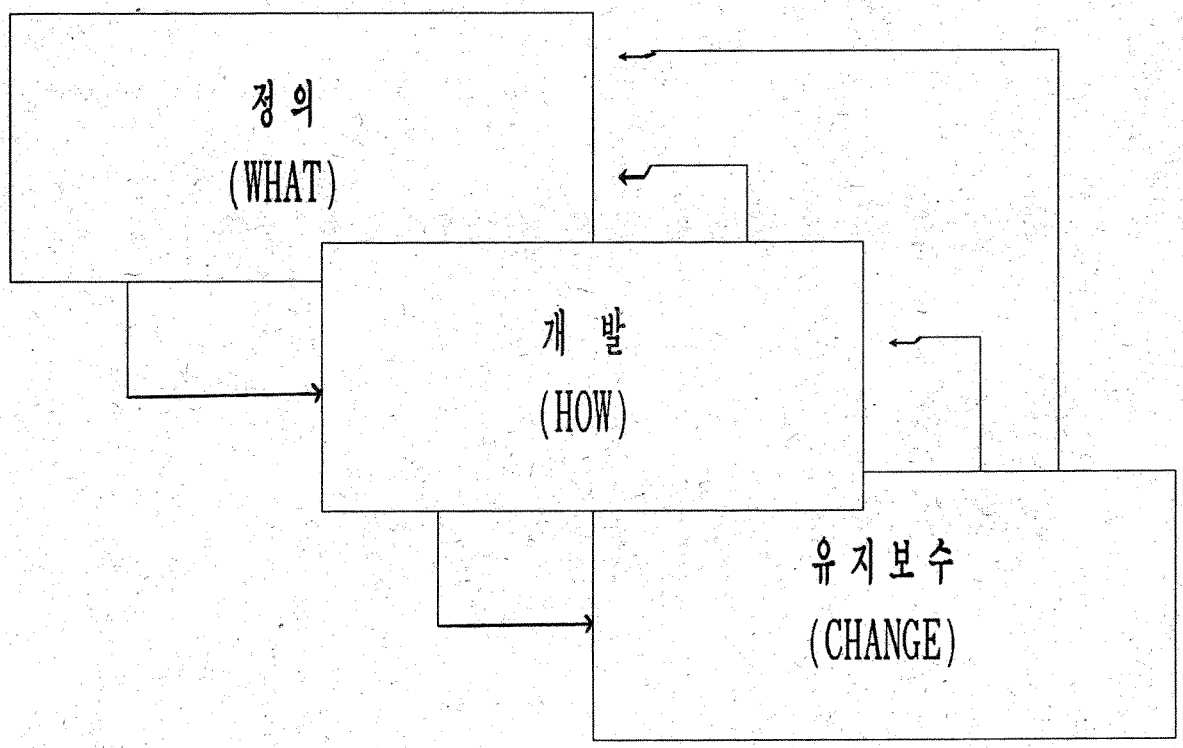
METHODOLOGY의 자동화 또는 반자동화로의 구현

▷ PROCEDURES

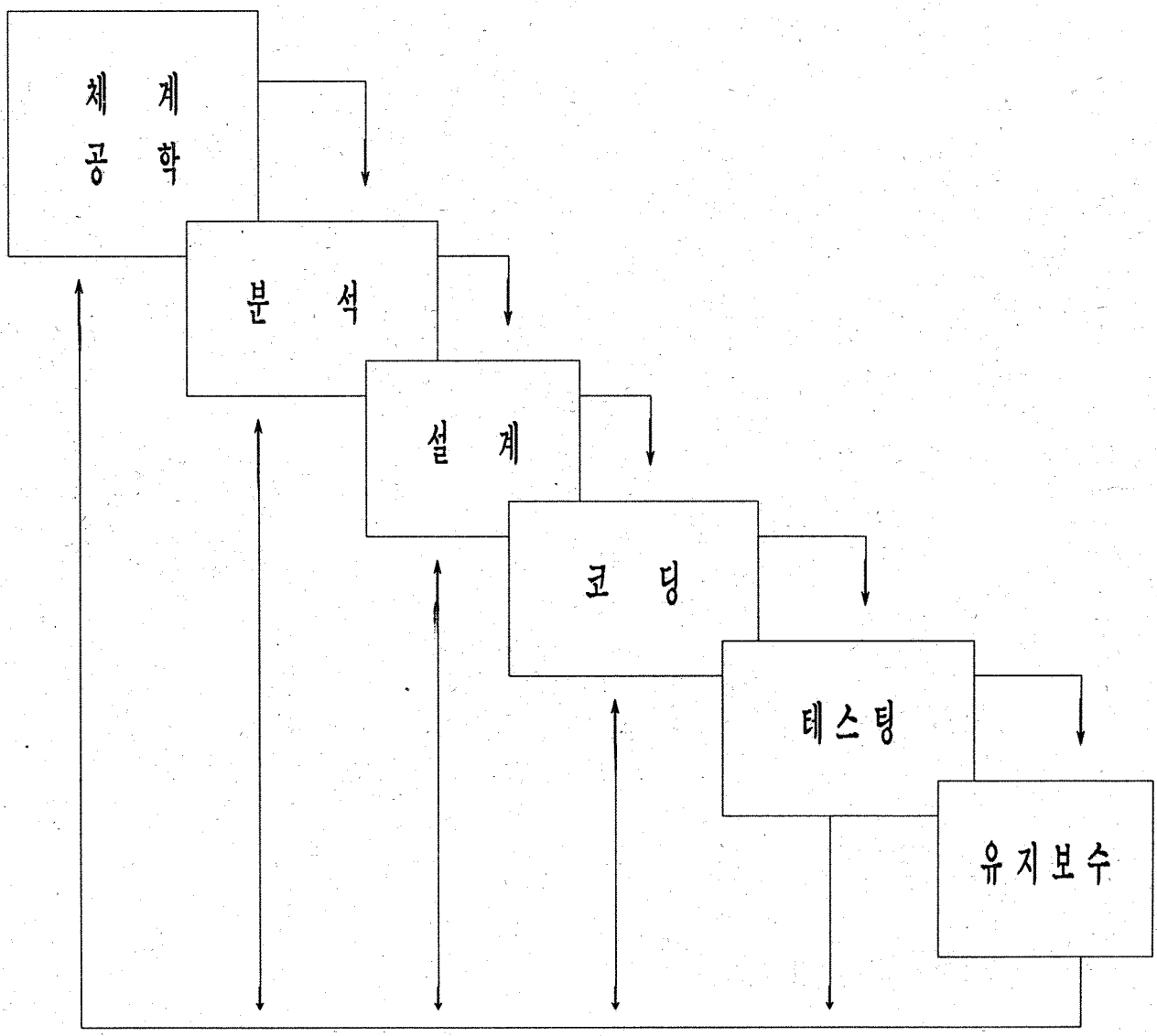
METHODOLOGY와 TOOL을 통합하는 매개체

소프트웨어 수명주기 모델

▷ 일반적인 모델

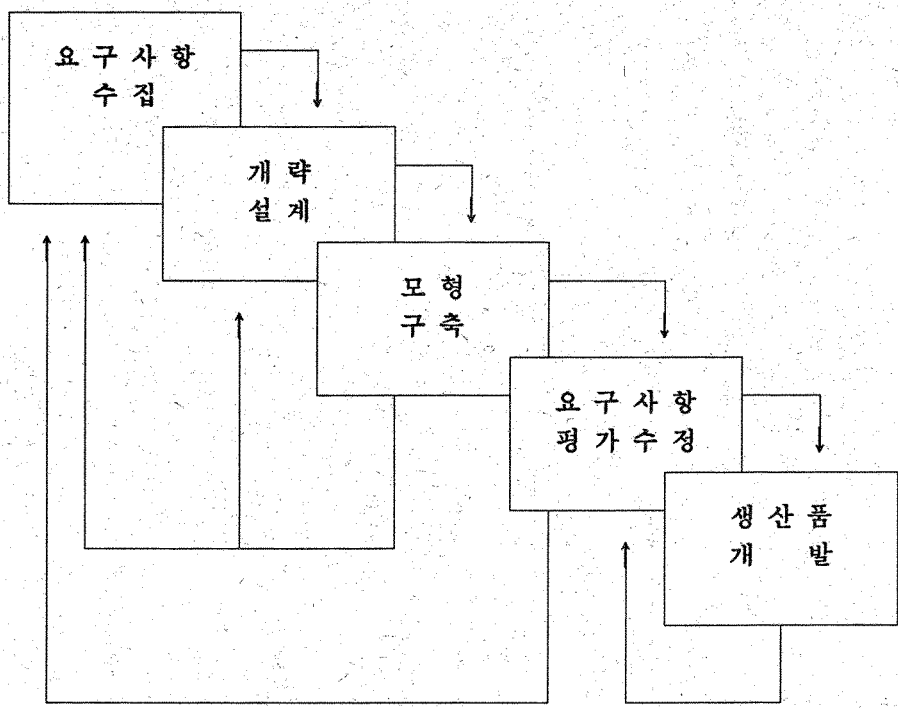


전통적인 수명주기 모델



▷ WATERFALL MODEL

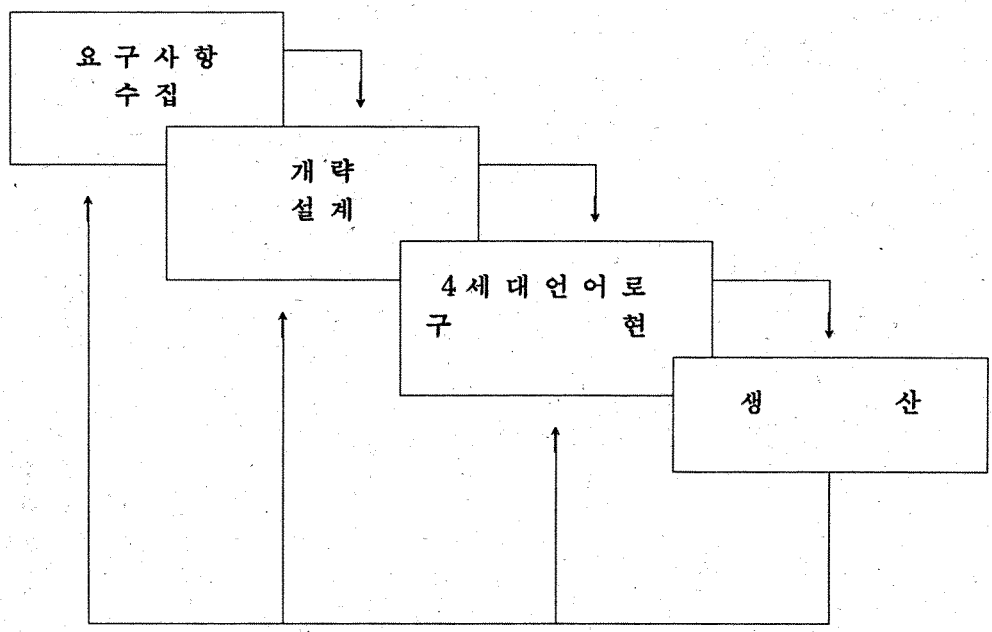
PROTOTYPING 수명주기 모델



▷ PROTOTYPE의 형태

- PAPER PROTOTYPE
- WORKING PROTOTYPE
- 현존프로그램 + 새로운 사용자 개발 프로그램

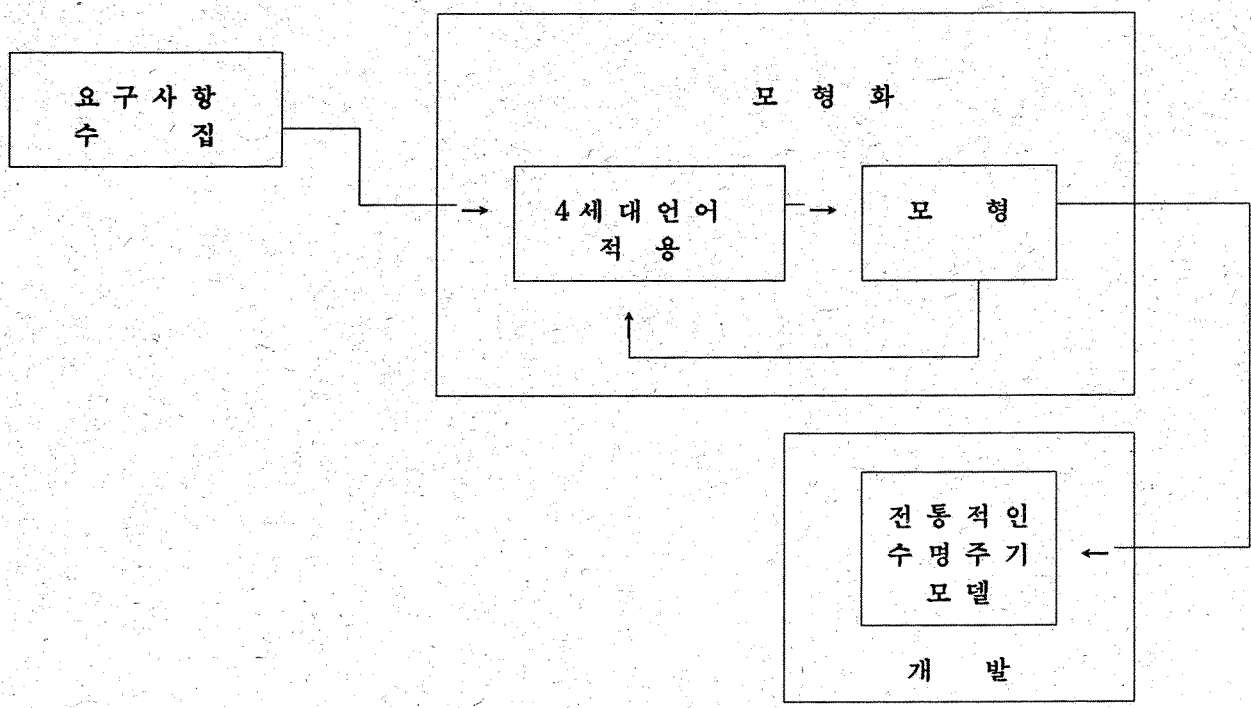
4세대언어를 이용한 수명주기 모델



▷ BUSINESS INFORMATION SYSTEM에 한정

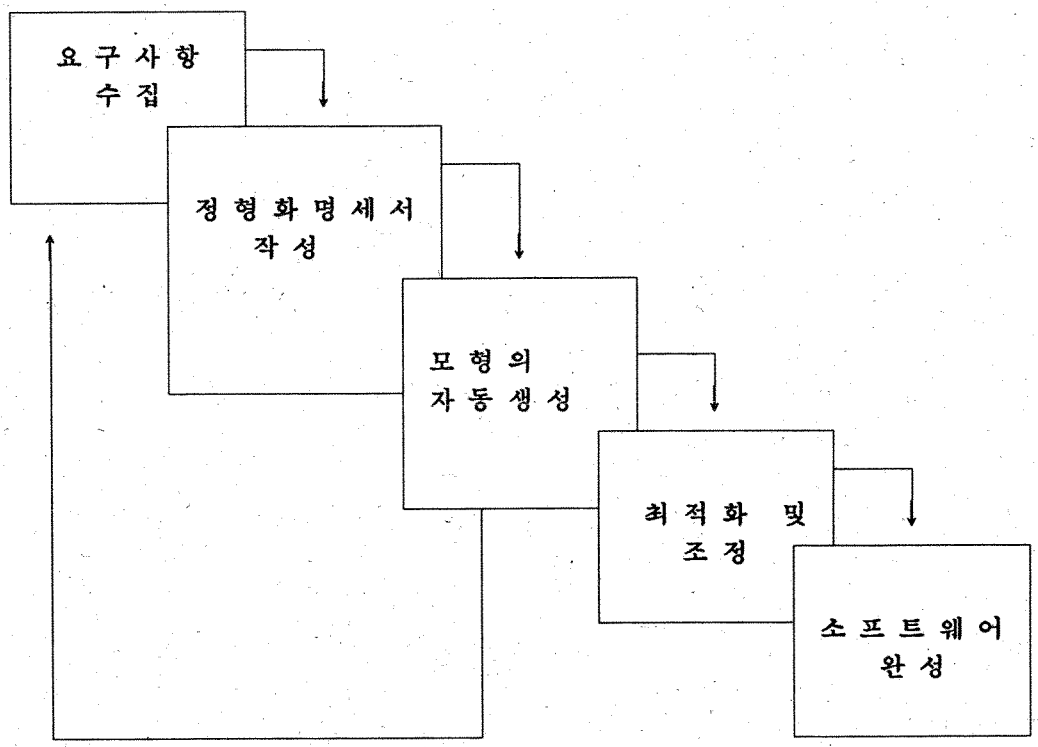
▷ 요구사항을 4세대언어로 직접 구현

조합형 수명주기 모델



▷ 응용분야의 특성에 따라 다양하게 조합 가능

자동생성 수명주기 모델



▷ 자동생성 도구의 개발 및 응용이 초기단계

▷ 완전한 자동화 및 모든 분야의 실용화가 어려움

COMPUTER-BASED SYSTEM

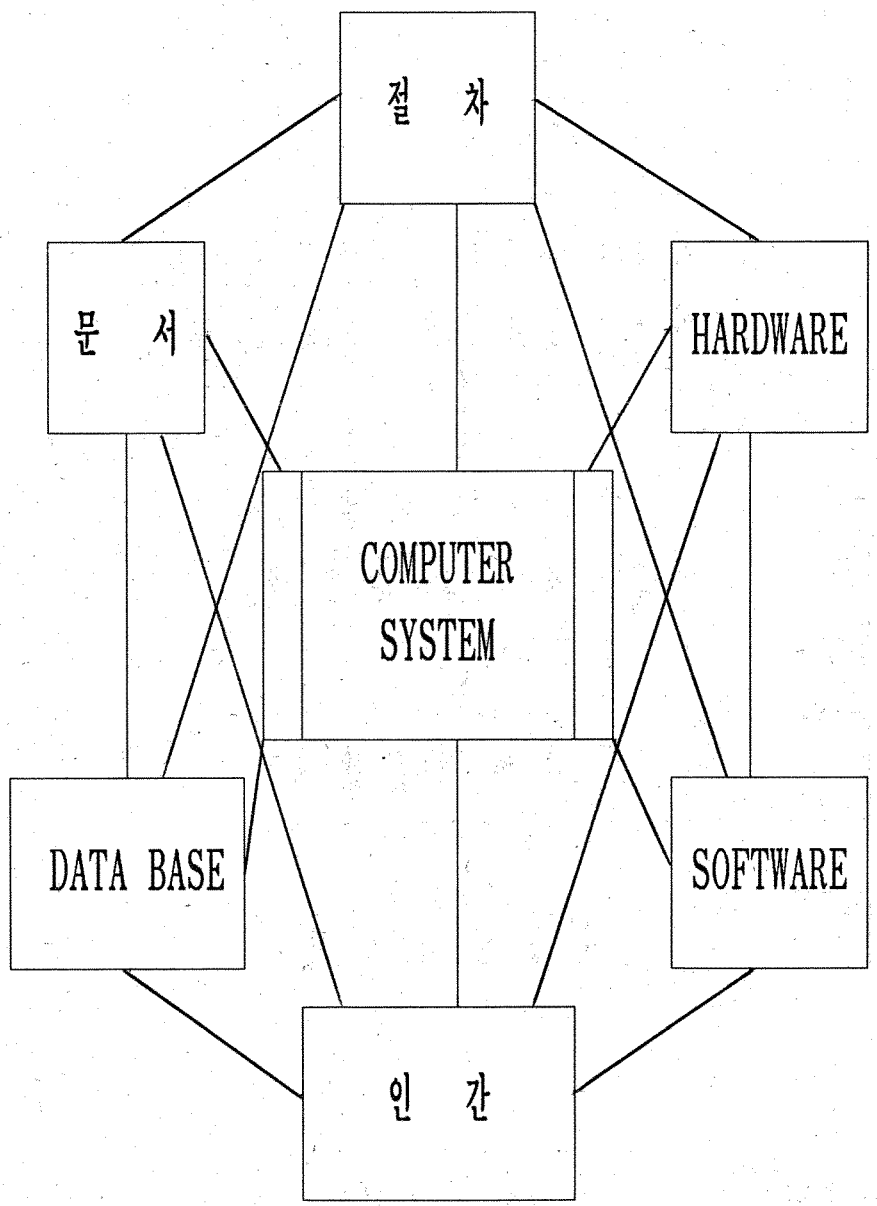
▷ SYSTEM

유기적인 통합체를 형성하기 위한 관련 사물들의 배열 또는 집합

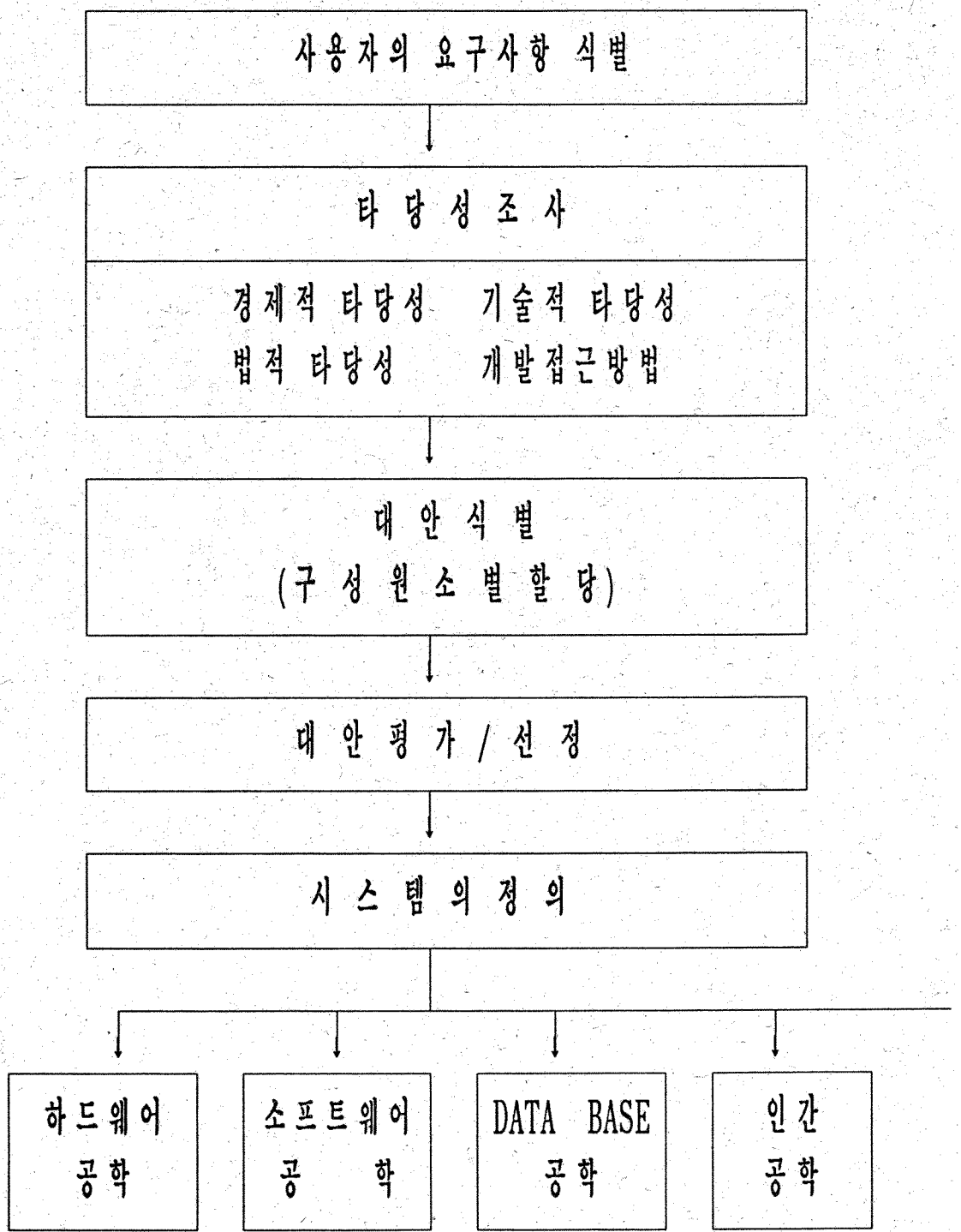
▷ COMPUTER-BASED SYSTEM

정보처리에 의해 임의의 방법, 절차, 또는 통제를 성취하기 위한 구성원소들의 배열 또는 집합

COMPUTER-BASED SYSTEM의 구성원소



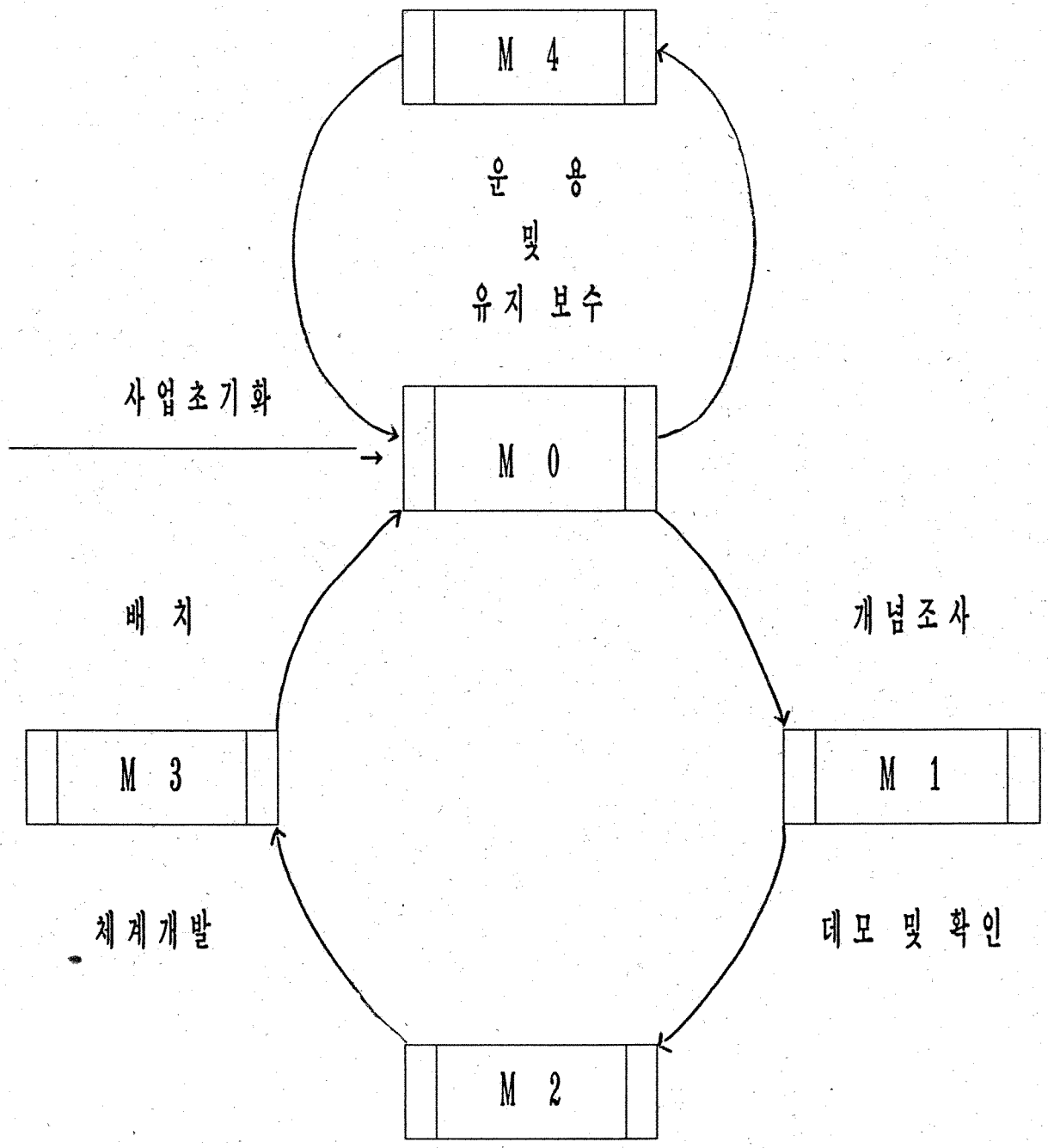
SYSTEM ENGINEERING



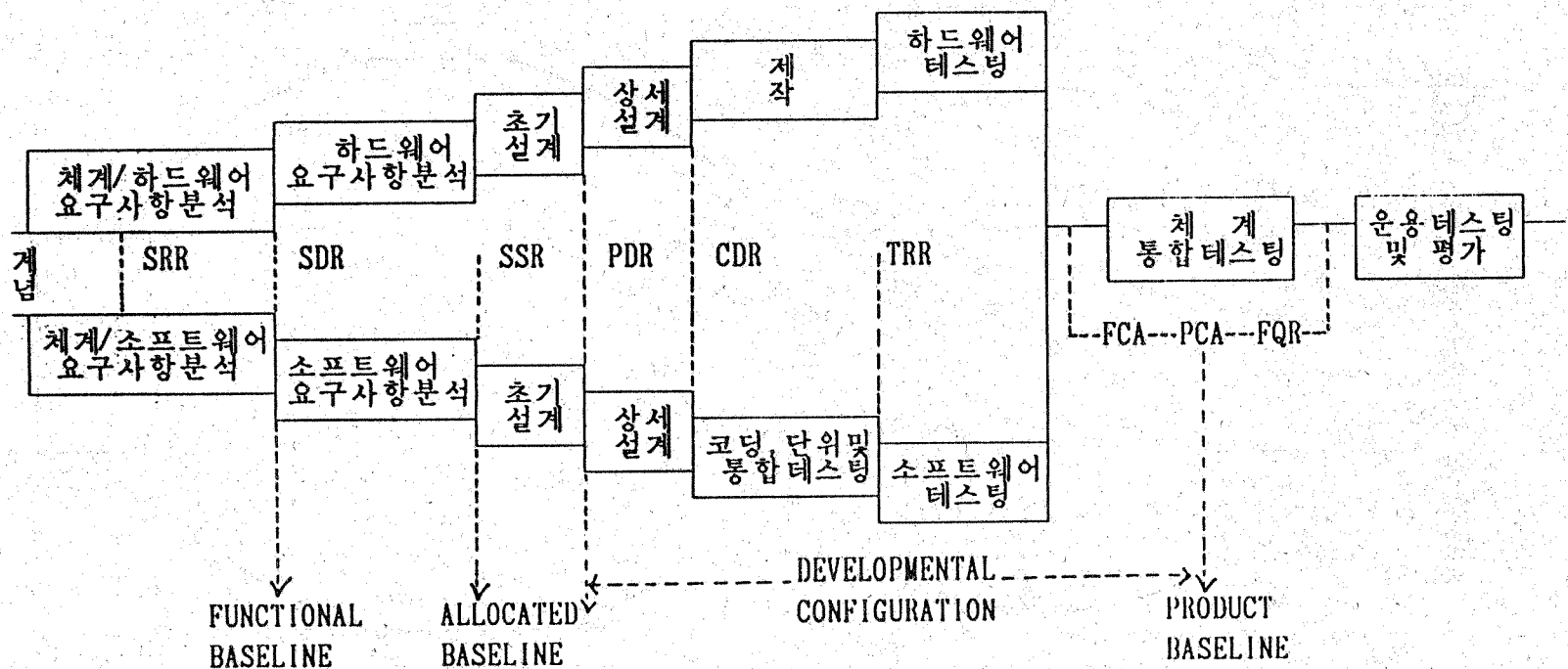
소프트웨어 사업계획

Software Project Planning

미국방성의 SYSTEM LIFE CYCLE



미국방성의 SYSTEM DEVELOPMENT CYCLE



소프트웨어 사업계획

▷ 소프트웨어 공학 과정에서의 첫 활동

▷ 소프트웨어 개발을 위한 종합 계획서 제공

▷ DEFINITION

- 소프트웨어의 SCOPE을 결정

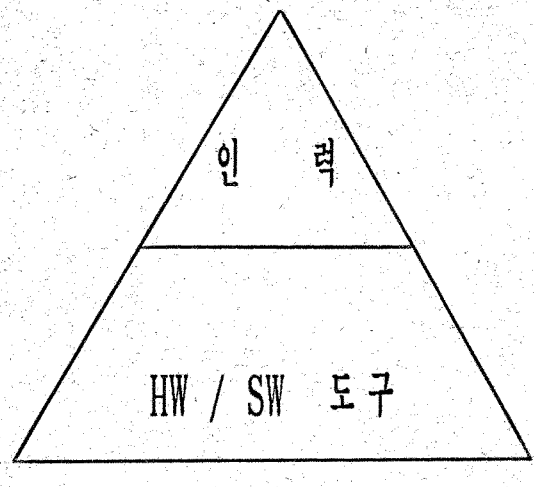
- FUNCTION, PERFORMANCE, INTERFACE 정의

▷ ESTIMATION

- RESOURCES, COST, SCHEDULE 예측

RESOURCE 예측

▷



▷ RESOURCE에 대한 설명

RESOURCE의 AVAILABILITY

요구되는 일자

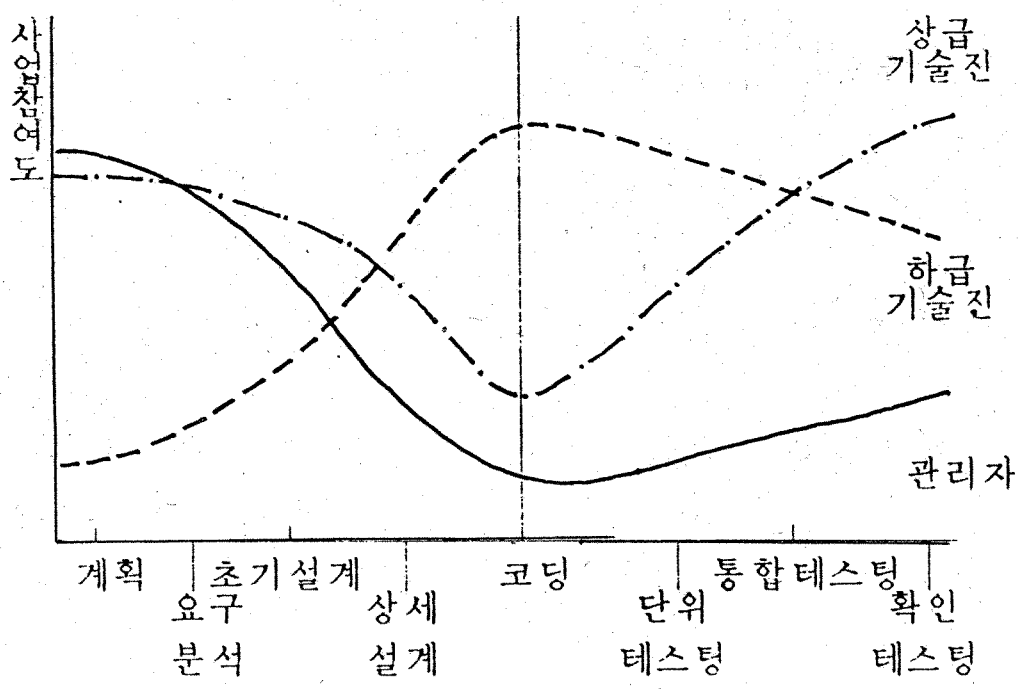
필요기간

인력자원 예측

▷ 가장 중요한 개발자원

▷ 필요 인력자원의 직책, 기술 필요수 예측

▷ 전형적인 프로젝트 인력 참여 모형



하드웨어자원 예측

▷ 개발용 시스템(HOST SYSTEM)

- 개발상에 필요한 컴퓨터와 주변기기
- 다수의 사용자를 지원
- 다량의 기억용량을 보유

▷ 목표로하는 시스템(TARGET MACHINE)

- COMPUTER-BASED SYSTEM의 구성원소로
- 소프트웨어를 수행시킬 처리기

▷ 개발상의 하드웨어 도구

- NUMERICAL CONTROL SOFTWARE
- EMBEDDED SOFTWARE

소프트웨어자원 예측

▷ Code-Oriented Tools

- Language Manipulation
- Language Specific

▷ Methodology Tools

- Methodology Supprot
- Project Management
- Computer Aided Software Engineering(CASE)

▷ Fourth Generation Tools

- Data Base Tools
- Code Generators
- Prototyping Aids

SOFTWARE MEASUREMENT

▷ FUNDAMENTAL TO ENGINEERING DISCIPLINE

▷ LORD KELVIN'S SAYING

- BEGINNING OF KNOWLEDGE

▷ SOFTWARE MEASUREMENT의 목적

- PRODUCT의 품질을 제시
- PRODUCT를 생산한 사람의 생산성을 평가
- 신 소프트웨어 기법 및 도구로 부터 얻을 수 있는 이득을 평가
- 예측을 위한 기준치를 형성

SOFTWARE METRICS의 종류

▷ DIRECT MEASURES/ INDIRECT MEASURES

- 비용, 인력, 프로그램크기, 속도
- 기능성, 품질, 복잡도, 효율성, 신뢰성

▷ TECHNICAL METRICS

QUALITY METRICS

PRODUCTIVITY METRICS

▷ SIZE-ORIENTED METRICS

FUNCTION-ORIENTED METRICS

HUMAN-ORIENTED METRICS

SIZE-ORIENTED METRICS

▷ 과거 프로젝트 수행시 다음과 같은 역사 자료 기록

- 인력(MAN-MONTH)
- 총투자비용(₩,$)
- 프로그램 크기(KLOC, LOC)
- 문서의 양(PAGE)
- 에러의 수

▷ 과거 자료를 기초로 생산성 및 품질 METRICS 산출

- 생산성 = 프로그램 크기 / 인력
- 품 질 = 에러수 / 프로그램 크기

▷ 그 외의 관심있는 METRICS 산출

- 한라인 작성비용 = 총투자비용 / 프로그램크기
- 한라인당 평균 관련 문서양 = 문서의 양 / 프로그램크기

FUNCTION-ORIENTED METRICS

▷ 다음과 같은 정보의 값을 수집

- NUMBER OF USER INPUTS
- NUMBER OF USER OUTPUTS
- NUMBER OF USER INQUIRIES
- NUMBER OF FILES
- NUMBER OF EXTERNAL INTERFACE

▷ FUNCTION POINT 산출

FP = 정보값의 합 * {C1 + C2 * (Fi)}

C1, C2 : 응용분야에 따라 경험적으로 결정
Fi : 복잡도 조정값(i=1...14)

▷ 생산성 및 품질 METRICS 산출

- 생산성 = FP / 인력
- 품 질 = 에러수 / FP

METRICS간의 비교

▷ 다른언어로 개발된 소프트웨어 간의 비교

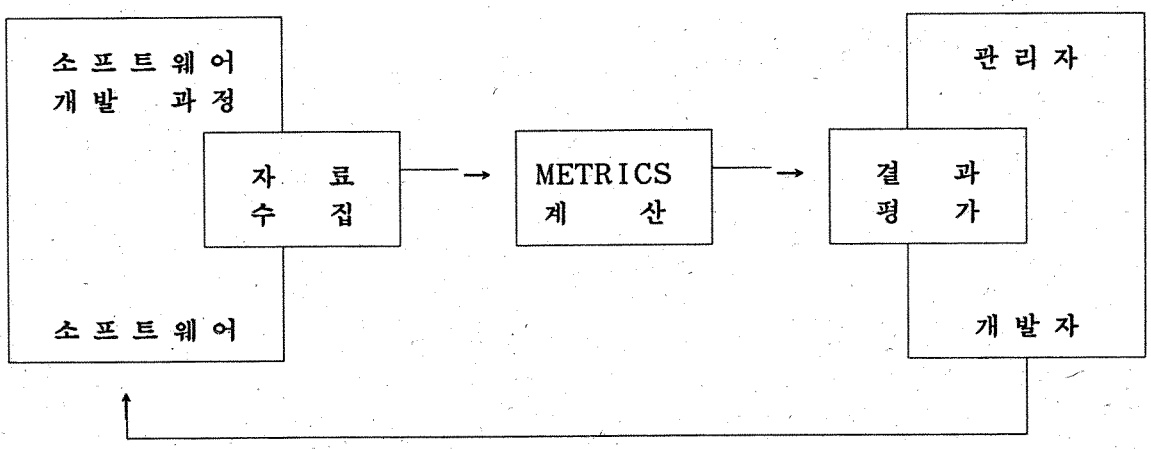
프로그래밍 언어	LOC / FP
COBOL	110
PL/1	65
4GL	25

▷ 프로젝트간의 METRICS 비교시 고려사항

- PEOPLE FACTOR(인력의 수준 및 규모)
- PROBLEM FACTOR(문제의 복잡도)
- PROCESS FACTOR(개발과정의 기술 정도)
- PRODUCT FACTOR(생산품의 신뢰성 및 성능)
- RESOURCE FACTOR(자원의 유용성)

SOFTWARE MEASURE의 예측

▷



▷ 역사자료 수집시 고려사항

- 정확하게 기록된 프로젝트로 부터 수집
- 가능한 많은 프로젝트에서 수집
- 일관성 있는 측정에 의한 수집
- 예측하고자 하는 프로젝트와 유사한 응용분야에서 수집

분할 기법에 의한 예측

▷ 예측 가능한 작은 FUNCTION으로 소프트웨어를 분할

▷ 분할된 PRIMITIVE FUNCTION에, 계획자의 경험과 직관에 의해 과거자료를 토대로 소프트웨어 MEASURE 값을 3수준(상한치, 최적치, 하한치)으로 할당

▷ BETA 분포에 따라 가중된 평균치를 계산

평균치 = (상한치 + 4 * 최적치 + 하한치) / 6

▷ PRIMITIVE FUNCTION으로 부터 최상위 FUNCTION까지 합산하여 최종 MEASURE 값을 산출

▷ 그 밖의 다른 SOFTWARE METRICS 계산

분할 기법에 의한 예측 예

▷ 다음과 같이 CAD SYSTEM 고려

- USER INTERFACE AND CONTROL FACILITIES(UICF)
- TWO-DIMENSIONAL GEOMETRIC ANALYSIS(2DGA)
- THREE-DIMENSIONAL GEOMETRIC ANALYSIS(3DGA)
- DATA STRUCTURE MANAGEMENT(DSM)
- COMPUTER GRAPHICS DISPLAY FACILITIES(CGDF)
- PERIPHERAL CONTROL(PC)
- DESIGN ANALYSIS MODULES(DAM)

LOC에 의한 예측

▷ 과거 LOC 및 그 밖의 자료 값을 수집하여 각 기능에 할당

FUNCTION	하한치	최적치	상한치	평균치	$/라인	라인/월
UCIF	1800	2400	2650	2340	14	315
2DGA	4100	5200	7400	5380	20	220
3DGA	4600	6900	8600	6800	20	220
DSM	2950	3400	3600	3350	18	240
CGDF	4050	4900	6200	4950	22	200
PC	2000	2100	2450	2140	28	140
DAM	6600	8500	9800	8400	18	300

LOC에 의한 예측

▷ 그밖의 다른 SOFTWARE METRICS 계산

FUNCTION	평균치	$/라인	라인/월	비용	MM
UCIF	2340	14	315	32,760	7.4
2DGA	5380	20	220	107,600	24.4
3DGA	6800	20	220	136,000	30.9
DSM	3350	18	240	60,300	13.9
CGDF	4950	22	200	108,900	24.7
PC	2140	28	140	59,920	15.2
DAM	8400	18	300	151,200	28.0
	33360			656,680	144.5

인력에 의한 예측

▷ 개발 단계별 과거 인력 자료를 수집하여 각 기능에 할당

FUNCTION	요구분석	설계	코딩	테스트	총합계
UCIF	1.0	2.0	0.5	3.5	7
2DGA	2.0	10.0	4.5	9.5	26
3DGA	2.5	6.0	20.0	11.0	31.5
DSM	2.0	6.0	3.0	4.0	15
CGDF	1.5	11.0	4.0	10.5	27
PC	1.5	6.0	3.5	5	16
DAM	4	14	5	7	30
	14.5	61	26.5	50.5	152.5

인력에 의한 예측

▷ 각 단계별 과거 MM당 비용 수집

요구분석	설계	코딩	테스트
5,200	4,800	4,200	4,250

▷ 각 단계별 비용 및 총 비용 계산

요구분석	설계	코딩	테스트	총합계
75,400	292,800	112,625	227,250	708,075

예측 값의 평가

▷ LOC에 의한 비용 및 인력 예측

총 LOC : 33,360

총 비용 : 656,680

총 MM : 144.5

▷ 인력에 의한 비용 예측

총 비용 : 708,075

총 MM : 152.5

▷ 위의 두 예측 값을 비교

비용 오차 : 7%

인력오차 : 5%

실험적인 예측모델

▷ Static Single-Variable Model

예측치i = Ci1 * (Ei ** Ci2)

Ci1, Ci2 : 과거프로젝트로 부터 산출되는 상수

Ei : 예측치 i를 산출하기 위한 소프트웨어의 특성치

▷ Static Multi-Variable Model

예측치i = Ci1*E1 + Ci2*E2 + ... + Cin*En

Cij : 예측치 i에 대한 j번째 실험상수

Ej : j번째 소프트웨어 특성치

▷ Dynamic Multi-Variable Model

예측치i = 예측치i1 + 예측치i2 + ... + 예측치in

예측치ij : j단계에서의 예측치 i

COCOMO MODEL

▷ BASIC COCOMO MODEL

$$MM = Ba * (KDSI) ** Bb$$

$$TDEV = Bc * (MM) ** Bd$$

MM : 인력(MAN - MONTH)

TDEV : 개발 기간(월수)

Ba, Bb, Bc, Bd : 과거 자료를 기초로 얻은 상수

INTERMEDIATE COCOMO MODEL

$$MM = \{ Ia * (KDSI) ** Ib \} * (2\ EAFi)$$

EAFi : i 번째 EFFORT ADJUST FACTOR

▷ DETAILED COCOMO MODEL

$$MM = \sum_{i} [\{ Da * (KDSI) ** Db \} * Ei * (\prod_{j} EAFij)]$$

Ei : i 단계에서의 노력 비율

EAFij : i 단계에서의 j 번째 EAF

COCOMO MODEL의 ASSUMPTION

▷ DELIVERED SOURCE INSTRUCTION

- PROJECT요원에 의해 생성되고, MACHINE CODE로 변환되어 DELIVERY되는 모든 PROGRAM INSTRUCTION

▷ MAN-MONTH

- 152MAN-HOURS
- 19 MAN-DAYS
- 1/12 MAN-YEAR

▷ PERIOD COVERAGE

- SOFTWARE요구사항 검토부터 통합및 테스트까지

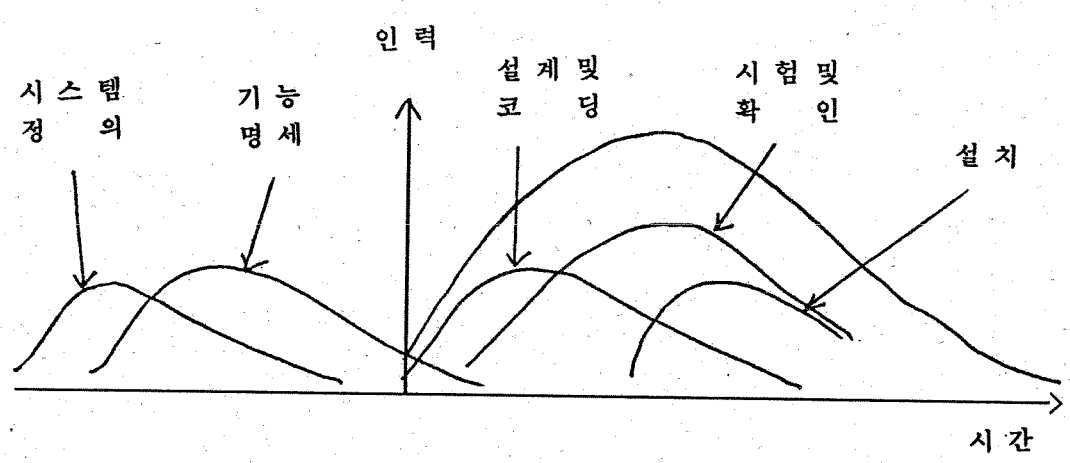
▷ 시스템명세서의 재수정 비용은 고려하지 않음

COCOMO MODEL의 DEVELOPMENT MODE

	ORGANIC MODE	SEMIDETACHED MODE	EMBEDDED MODE
시스템 목적에 대한 조직 구성원의 이해도	완전이해	상당이해	개략적 이해
관련 SOFTWARE SYS.에 대한 구성원들의 경험도	아주많음	많음	중간
요구사항과 SOFTWARE의 일치성에 대한 요구정도	기본적	높음	매우높음
관련 HARDWARE 및 운행절차의 병행 개발	다소	중간	아주많음
새로운 자료처리 및 알고리즘에 대한 요구정도	최적요구	다소요구	상당요구
완성시기에 대한 민감도	낮음	중간	높음
시스템 크기의 범위	< 50 KDSI	< 300 KDSI	모든크기

PUTNAM MODEL

▷ RAYLEIGH-NORDON CURVE



▷ 라인수와 인력 및 개발시간과의 관계를 나타내는 산술식을 산출

$$L = Ck * (K ** 1/3) * (Td ** 4/3)$$

L : 라 인 수
Ck : 기 술 상 태 상 수
K : 인 력 (MY)
Td : 개 발 시 간 (YEAR)

TIME STUDY MODEL

▷ 다음 요인들을 분석

a : 일일 작업 시간당 행정적 또는 간접적인
일로 소비하는 시간의 비율

t : 평균 작업 중단시간(분)

r : 중단후 평균 회복 시간(분)

k : 프로젝트 관련 인원에 의한 일일 작업 중단 횟수

p : 다른 원인으로 인한 일일 작업 중단 횟수

i : 기본 급여에 대한 일인당 간접비 비율

d : 기본 급여에 대한 초과 근무 수당

g : 일인당 일일평균 초과 근무 시간

n : 프로젝트 참여 인원수

md : 프로젝트 종료까지의 MAN-DAYS

TIME STUDY MODEL

▷ 일인당 일일 유효 작업시간 비율을 산출

$$w = 0.125 * \{ 8 + g - 8a - \frac{4r}{60} - \frac{p(t+r)}{60} - \frac{k(n-1)(1+r)}{60} \}$$

▷ W를 이용하여 다음과 같은 방정식을 개발

$$T = \frac{7}{5nw}$$

= 프로젝트 종료까지의 달력일수와 MAN-DAYS의 비율

$$c = ns\ (8 + 8i + gd)$$

= 시간당 평균 기본급여 s에 대한 일일 노력비용

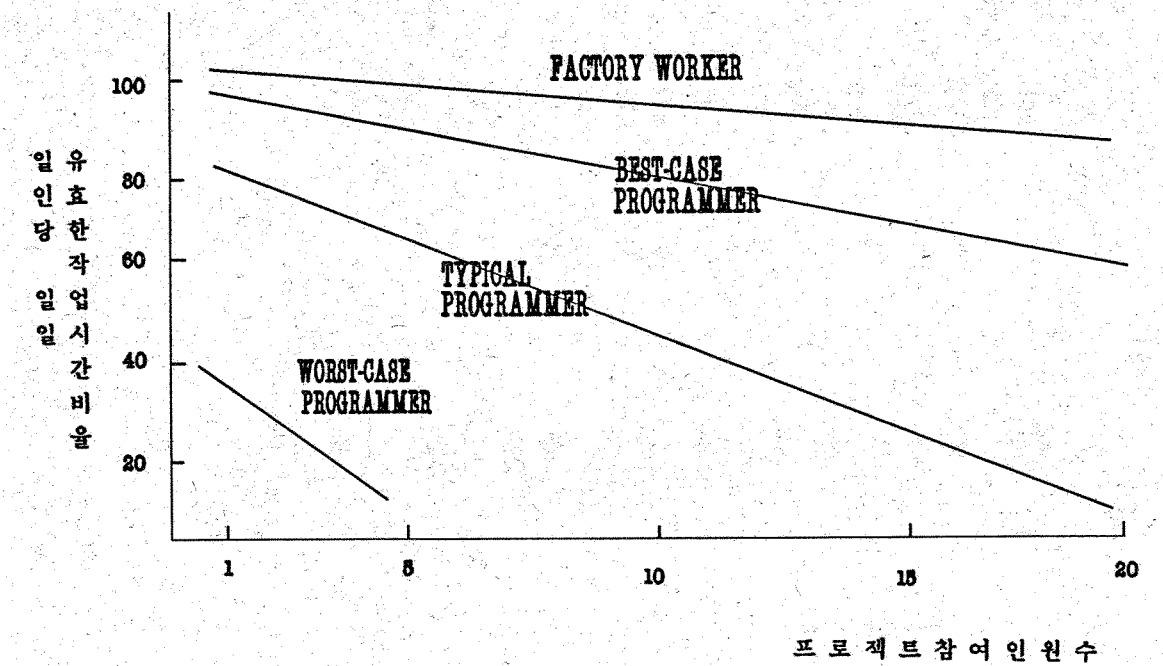
$$e = \frac{nw}{c}$$

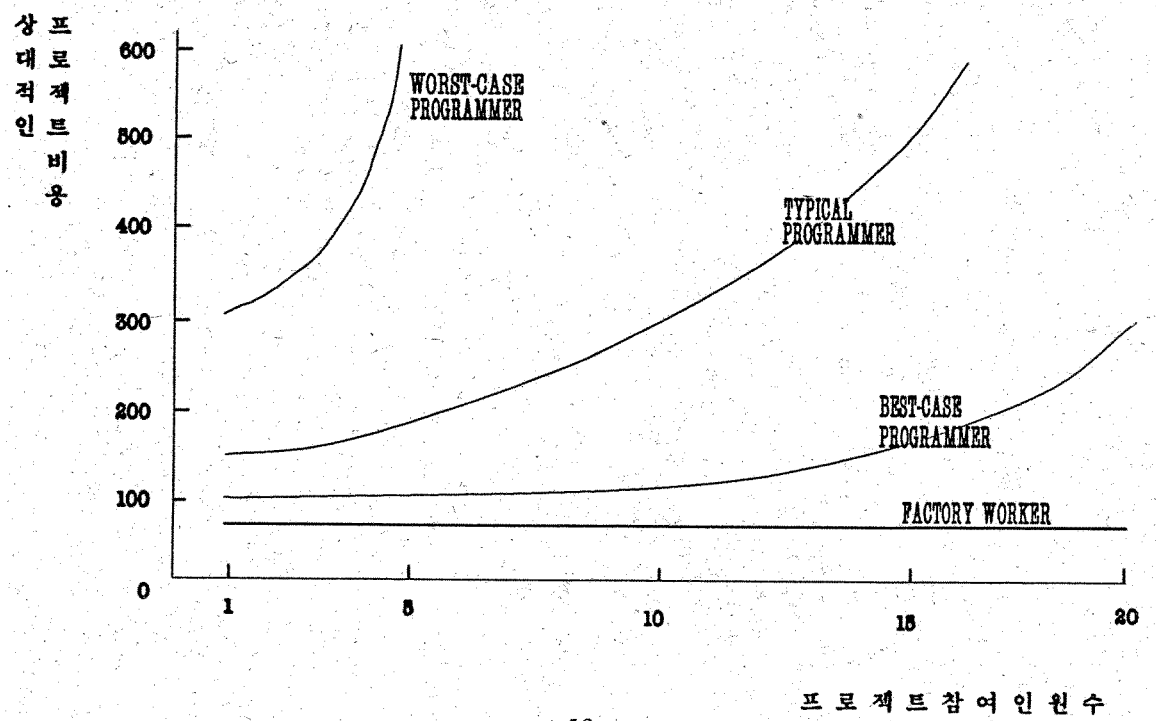
= 비용의 효율성

$$C = \frac{c}{nw}$$

= MAN-DAY당 프로젝트 비용

TIME STUDY MODEL





그외의 비용 예측 방법

▷ EXPERT JUDGEMENT

- SINGLE EXPERT JUDGEMENT
- ONE MORE EXPERT JUDGEMENT
- GROUP MEETING TECHNIQUE
- DELPHI TECHNIQUE
- WIDEBAND DELPHI TECHNIQUE

▷ ESTIMATIOM BY ANALOGY

▷ TASK-UNIT APPROACH

자동화 예측 도구

▷ 공통적인 입력자료

- 프로젝트 크기 또는 기능면에 대한 정량적인 예측 값
- 정상적인 프로젝트 특성
- 개발진과 개발환경

▷ SLIM

- PUTNAM예측모델, 선형계획법, 통계적 모의 실험, PERT기법 등을 적용

▷ ESTIMACS

- FUNCTION POINT방법 적용

일정계획시 고려사항

▷ 프로젝트 후기에는 인원을 증가하지 말아라

▷ 개발과정 중 규칙적인 기간마다 이정표를 세워라

▷ 병렬로 발생하는 업무간의 상호관계를 정의하라

▷ 분석 및 설계와 테스팅 단계에 집중하라

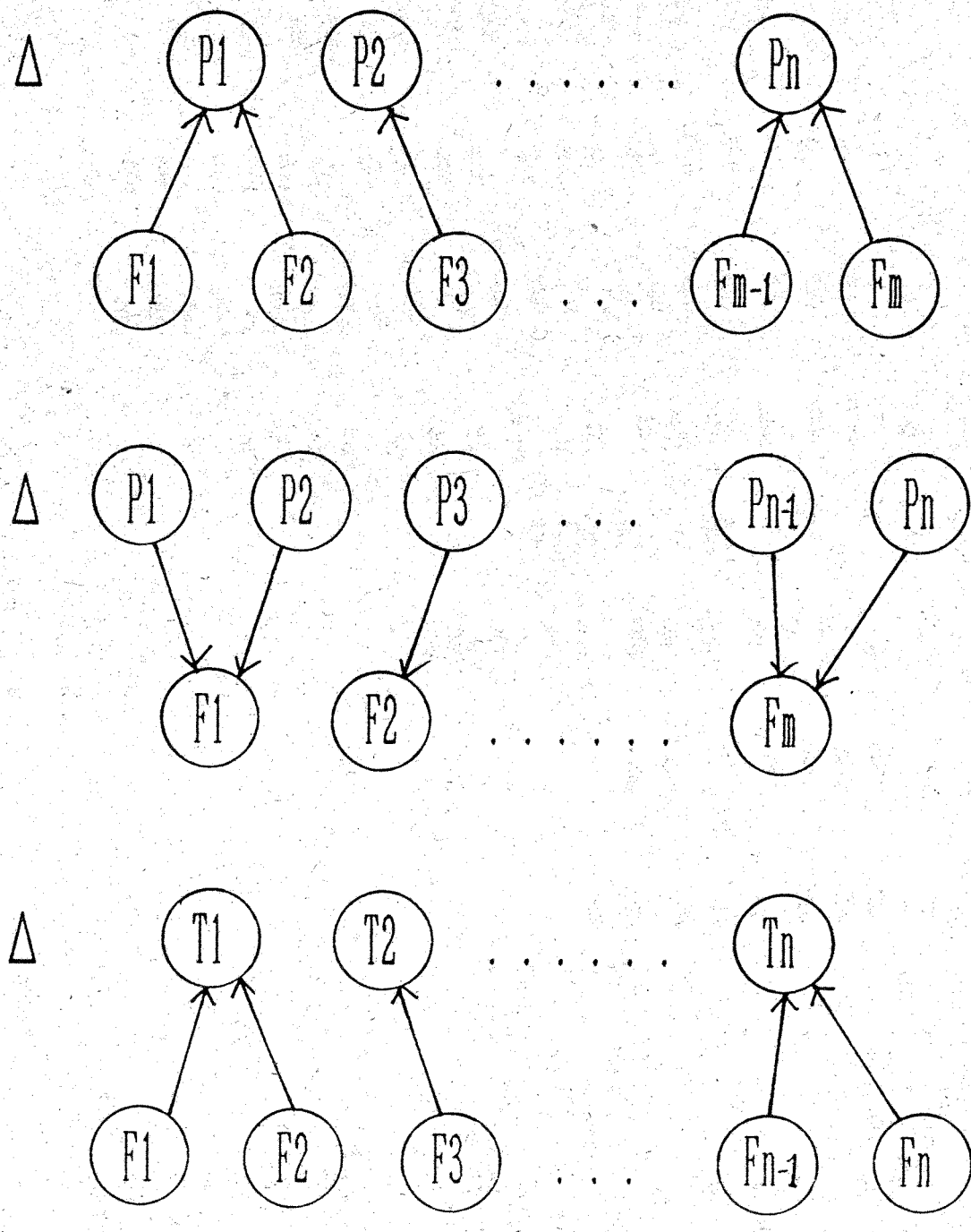
(40 : 20 : 40 RULE)

▷ 유지보수 단계를 잊지 말아라

▷ PERT/CPM

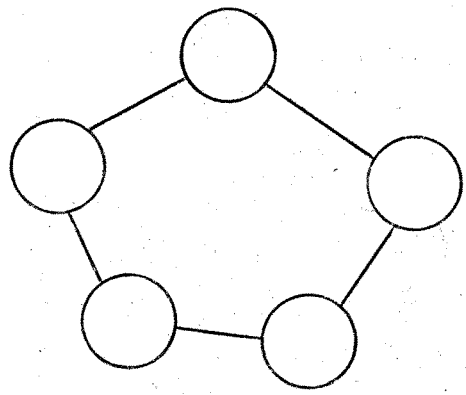
▷ 자동화 도구의 이용

팀 구성과 업무 분담

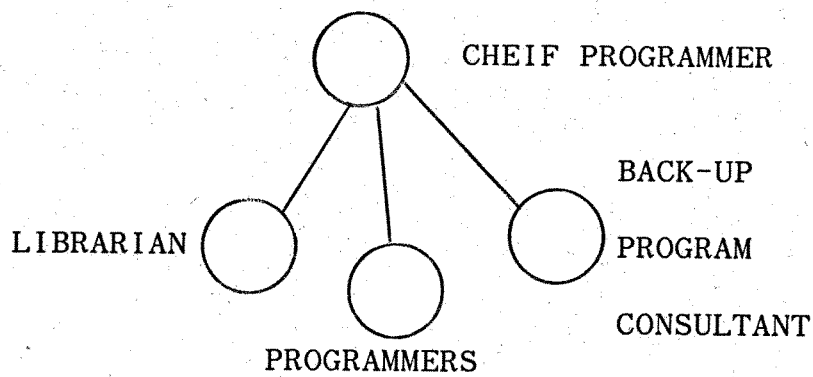

Δ
P1
P2
. . . . . .
Pn
F1
F2
F3
. . .
Fm-1
Fm
Δ
P1
P2
P3
. . .
Pn-1
Pn
F1
F2
. . . . . .
Fm
Δ
T1
T2
. . . . . .
Tn
F1
F2
F3
. . .
Fn-1
Fn

TEAM STRUCTURE

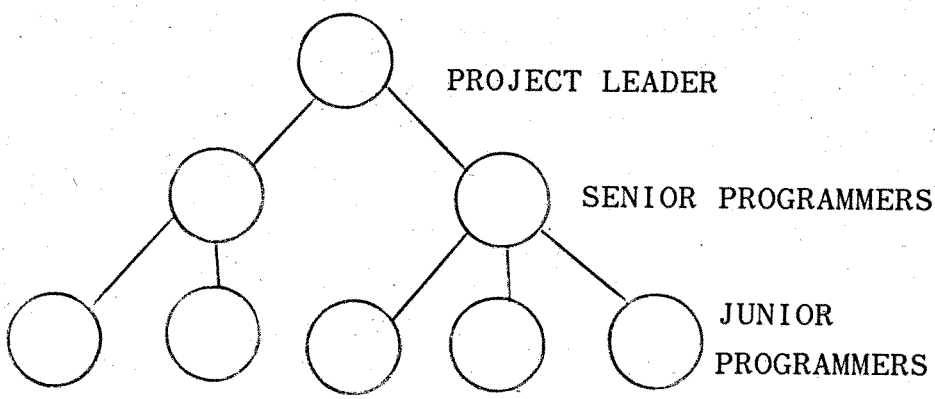
- DEMOCRATIC TEAM



- CHIEF-PROGRAMMER TEAM



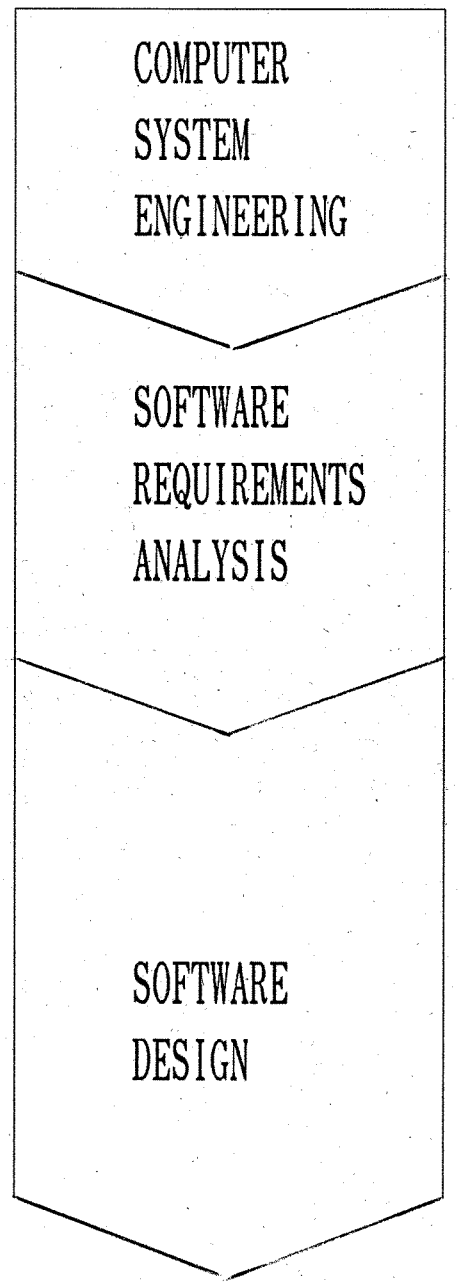
- HIERARCHICAL TEAM



소프트웨어 요구사항 분석

Software Requirements Analysis

REQUIREMENTS ANALYSIS



SYSTEM ENGNEER

- SOFTWARE ALLOCATION

SOFTWARE ENGINEER

- SOFTWARE ALLOCATION을 REFINE
- INFORMATION DOMAIN을 기술

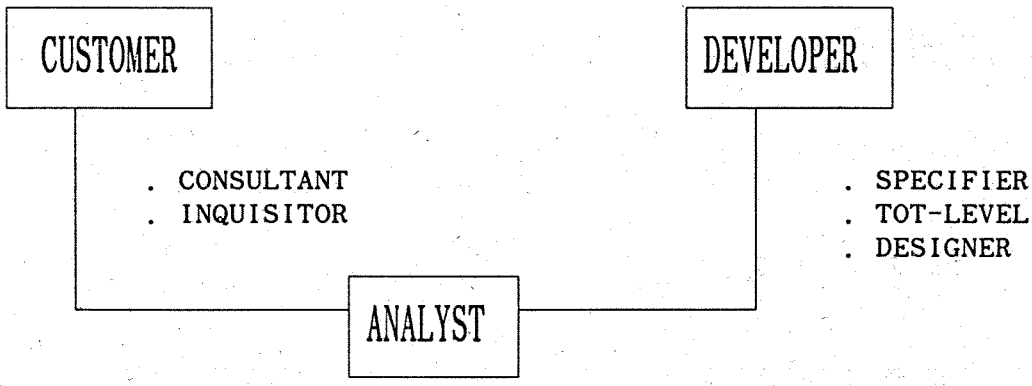
SOFTWARE DESIGNER

- DATA DESIGN
- ARCHITECTURAL DESIGN
- PROCEDURAL DESIGN

ANALYSIS TASKS

- ▷ PROBLEM RECOGNITION
  - SYSTEM SPECIFICATION
  - SOFTWARE PROJECT PLAN
  - COMMUNICATION PATH 구축
- ▷ EVALUATION AND SYNTHESIS
- ▷ SPECIFICATION
  - SOFTWARE REQUIREMENTS SPECIFICATION
  - VALIDATION CRITERIA
  - PRELIMINARY USER'S MANUAL
- ▷ REVIEW

분석가의 특성 및 역활



▷ 개략적 개념을 완전 파악하여 논리적인 부분으로 재구성하고, 각 부분의 SOLUTION을 종합하는 능력

▷ 상반되거나 혼동을 주는 SOURCE로 부터 적합한 해결 방안을 도출하는 능력

▷ 사용자의 환경을 이해하는 능력

▷ 서면 및 구두로의 의사 전달 능력

▷ 숲을 볼수 있는 능력

PROBLEM AREAS/CAUSE

▷ PROBLEM AREA

- 적절한 정보의 획득
- 문제의 복잡성 처리
- 변화에 대한 수용

▷ PROBLEM CAUSE

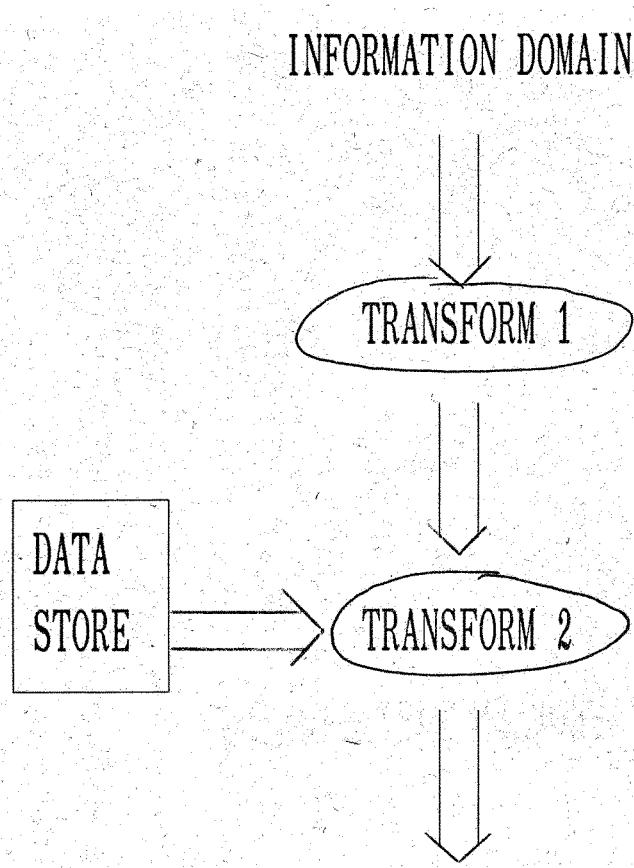
- 빈약한 의사 소통
- 부적합한 기법 및 도구 사용
- 요구분석 업무를 단 시일 내에 끝내려는 경향
- 대안 고려 실패

ANALYSIS PRINCIPLES

▷ FUNCTIONAL DOMAIN과 INFORMATION DOMAIN의 표현 및 이해

▷ 계층적 형태로의 PROBLEM PARTITIONING

▷ SYSTEM의 LOGICAL/PHYSICAL REPRSENTATION



- INFORMATION FLOW
- INFORMATION CONTENT
- INFORMATION STRUCTURE

복잡도의 해결방법

▷ PARTITION

- 한 시점에서 SYSTEM의 특정 구성원소에만 집중하는 기법

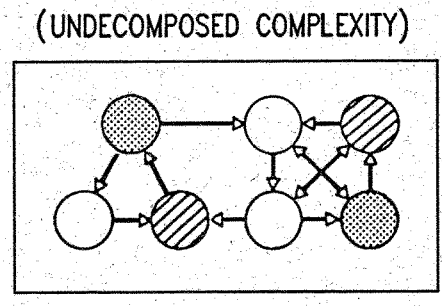
▷ ABSTRACTION

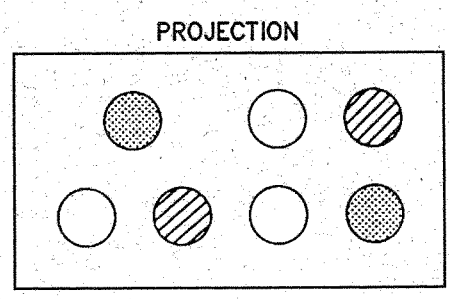
- 상세한 사항을 억제하고 필요한 특성에만 집중하는 기법

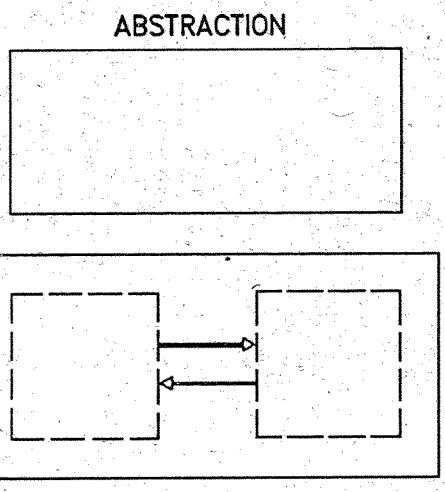
▷ PROJECTION

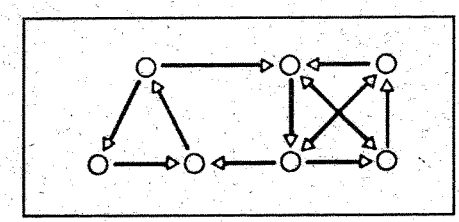
- 다른 관점으로 시스템을 이해하는 기법

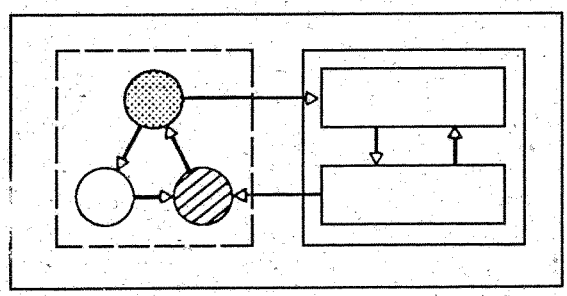
COMPLEXITY DECOMPOSION

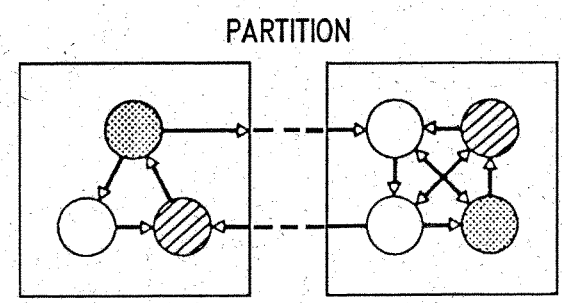












LOGICAL/PHYSICAL VIEW

▷ LOGICAL VIEW

- 세부적인 구현사항을 고려하지 않고 단지

  수행되어야 할 FUNCTION과 처리되어야 할 정보를 기술

- ESSENTIAL MODEL

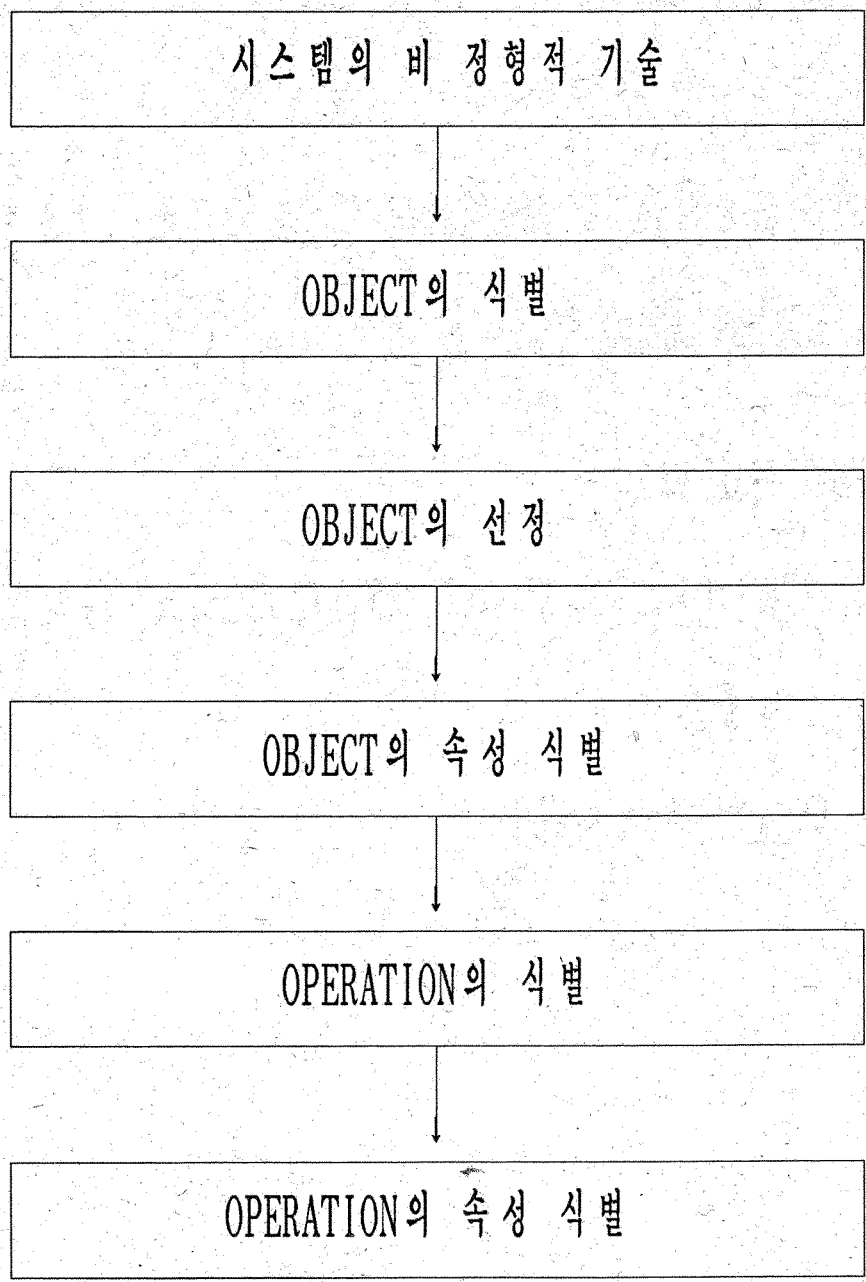
▷ PHYSICAL VIEW

- FUNCTIONAL과 INFORMATION STRUCTURE를 처리할

  실세계를 기술

- INCARNATION MODEL

OBJECT-ORIENTED ANALYSIS



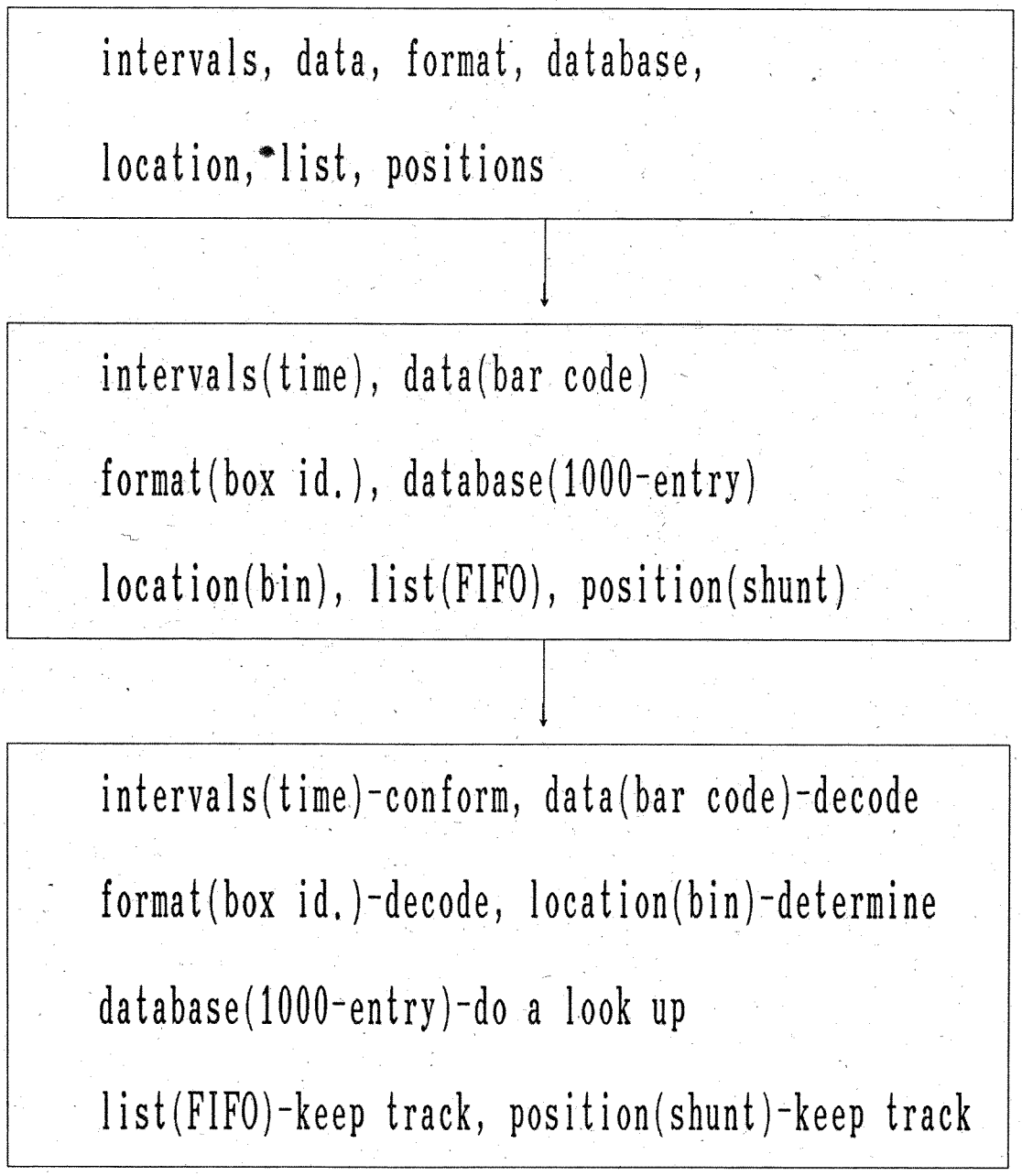
OBJECT-ORIENTED ANALYSIS의 예

CLSS software will receive input information from a bar code reader at time intervals that conform to the conveyor line speed. Bar code data will be decoded into box identification format. The software will do a look-up in a 1000-entry data base to determine proper bin location for the box currently at the reader (sorting station). A FIFO list will be used to keep track of shunt positions for each box as it moves past the sorting station.

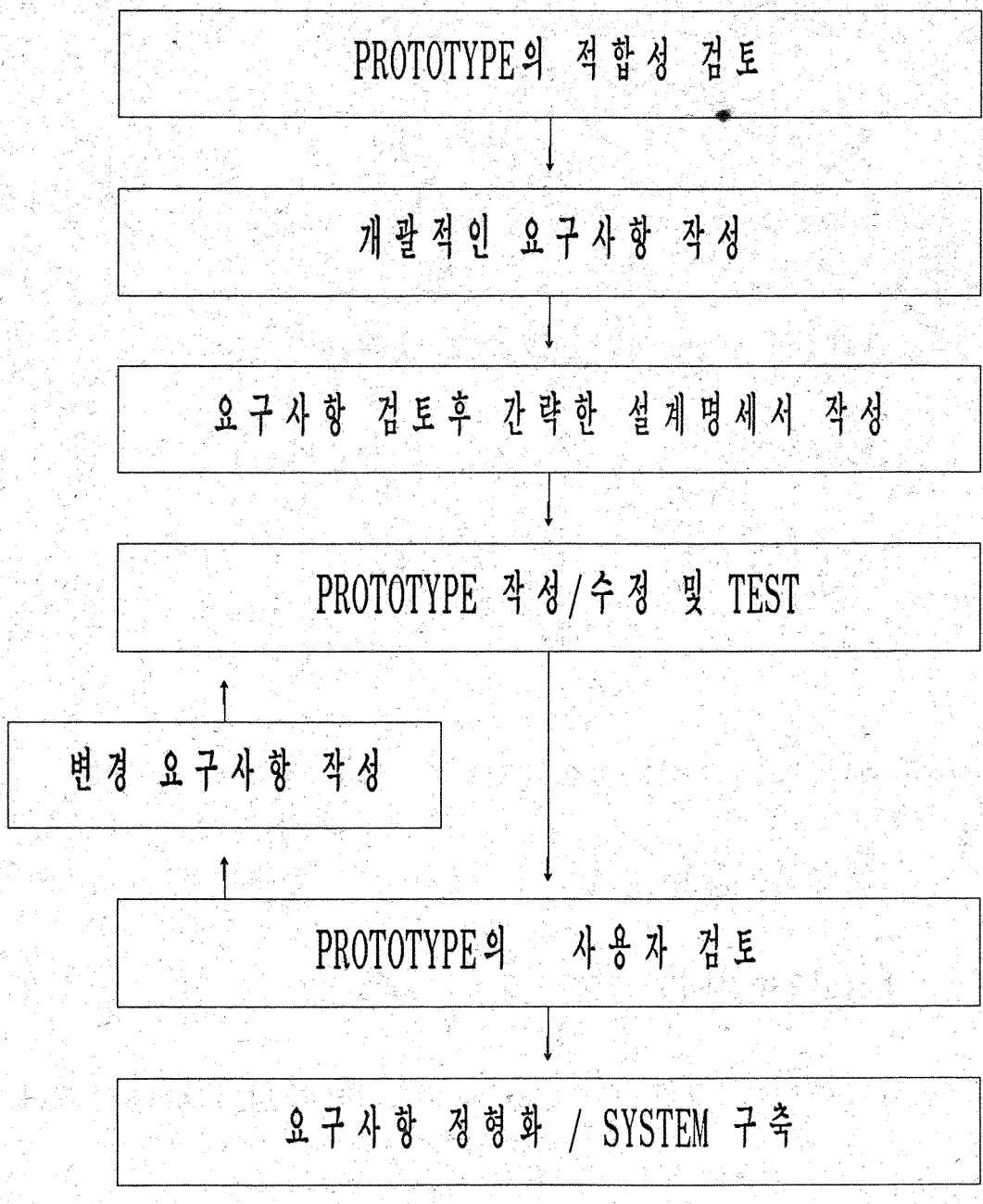
OBJECT-ORIENTED ANALYSIS의 예

CLSS software will receive input information from a bar code reader at time intervals that conform to the conveyor line speed. Bar code data will be decoded into box identification format. The software will do a look-up in a 1000-entry data base to determine proper bin location for the box currently at the reader (sorting station). A FIFO list will be used to keep track of shunt positions for each box as it moves past the sorting station.

OBJECT-ORIENTED ANALYSIS의 예



PROTOTYPING APPROACH



PROTOTYPING 기법 및 도구

▷ FOURTH GENERATION TECHNIQUES

- 4세대 언어를 이용, 쉽게 모형 구축
- MANTIS, CSP, UFO, FOCUS, NATURAL등

▷ REUSABLE SOFTWARE COMPONENTS

- 현존 데이타베이스, 프로그램등을 이용
- 현존 SOFTWARE PRODUCT를 이용

▷ FORMAL SPECIFICATION AND PROTOTYPING

- 요구사항을 정형화된 언어로 명세하여
  수행 가능한 코드로 변환
- SREM/RSL, TAGS/IORL 등

SPECIFICATION PRINCPLES

- FUNCTIONALITY와 IMPLEMENTATION을 분리하라
- PROCESS-ORIENTED SPECIFICATION을 하라
- SOFTWARE를 구성원소로 하는 SYSTEM을 기술하라
- SYSTEM이 작동하는 ENVIRONMENT를 기술하라
- COGNITIVE MODEL로 기술하라
- OPERATIONAL 하도록 기술하라
- SPECIFICATION은 확장이 용이해야 한다
- SPECIFICATION의 내용을 국부화 하라

REPRESENTATION PRINCIPLES

▷ REPRESENTATION FORMAT과 내용을 PROBLEM DOMAIN에 적합하게 기술하라

▷ SPECIFICATION 내의 정보를 중첩되도록 기술하라

▷ 가능한 NOTATIONAL FORM은 수시로 제한하고 사용상 일관성을 유지하라

▷ REPRESENTATION이 수정 용이하도록 하라

요구분석 기법의 평가 요소

- INFORMATION DOMAIN 분석을 위한 MECHANISM
- FUNCTIONAL DOMAIN을 기술하기 위한 접근방법
- INTERFACE의 정의
- PROBLEM PARTITIONING을 위한 MECHANSIM
- ABSTRACTION을 지원하는 MECHANISM
- PHYSICAL VIEW와 LOGICAL VIEW의 기술

요구분석 기법의 분류

▷ DATA-FLOW ORIENTED ANALYSIS

- STRUCTURED ANALYSIS
- STRUCTURED ANALYSIS AND DESIGN TECHNIQUE

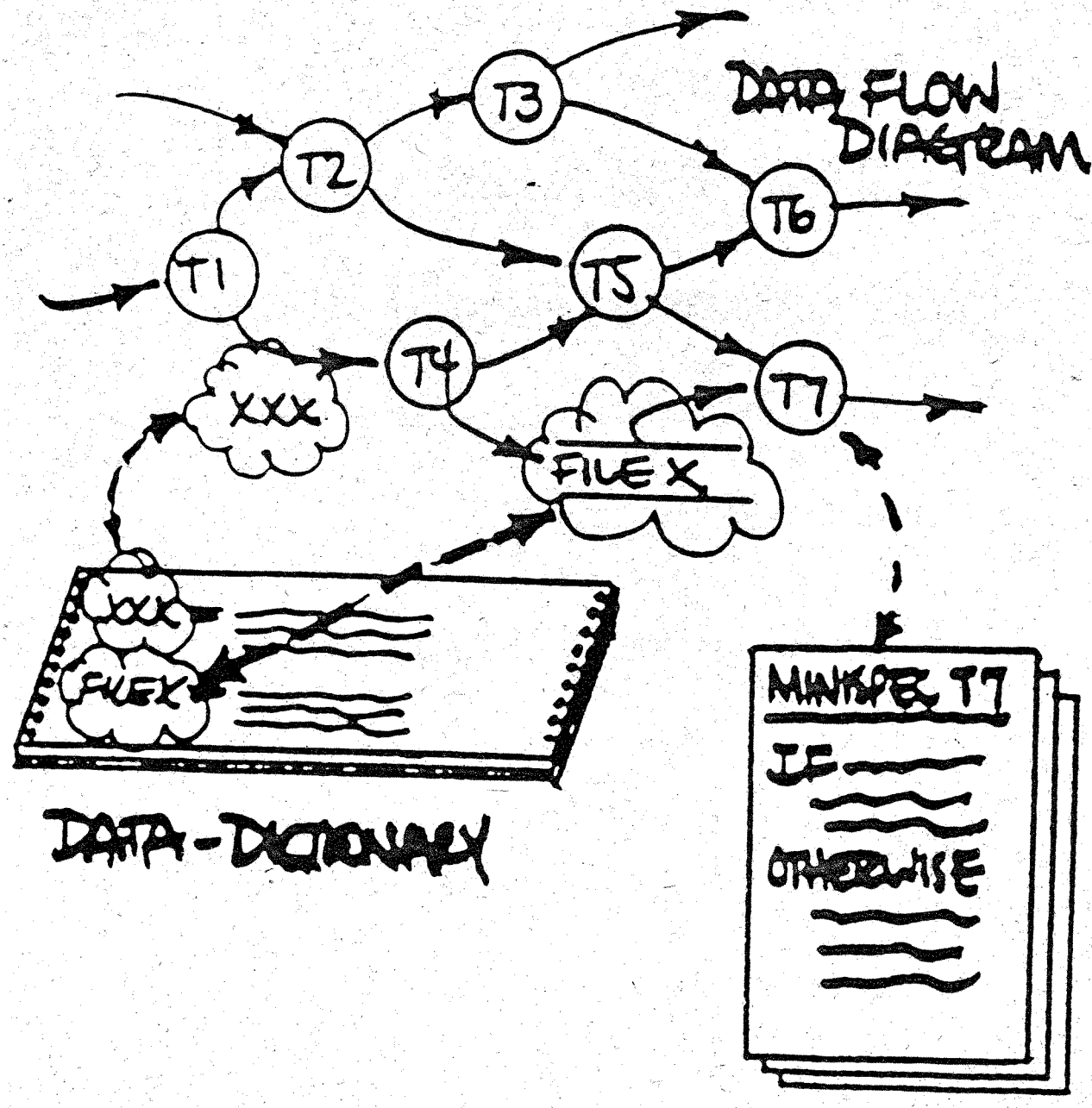
▷ DATA-STRUCTURE ORIENTED ANALYSIS

- DATA-STRUCTURE SYSTEM DEVELOPMENT
- JACKSON SYSTEM DEVELOPMENT

▷ LANGUAGE-BASED FORMAL SPECIFICATION

- PSL/PSA
- SREM/REVS
- TAGS

DATA-FLOW ORIENTED ANALYSIS



DATA-FLOW ORIENTED ANALYSIS의 3가지 도구

▷ DATA FLOW DIAGRAM

- A FUNCTIONAL NETWORK REPRESENTATION OF A SYSTEM

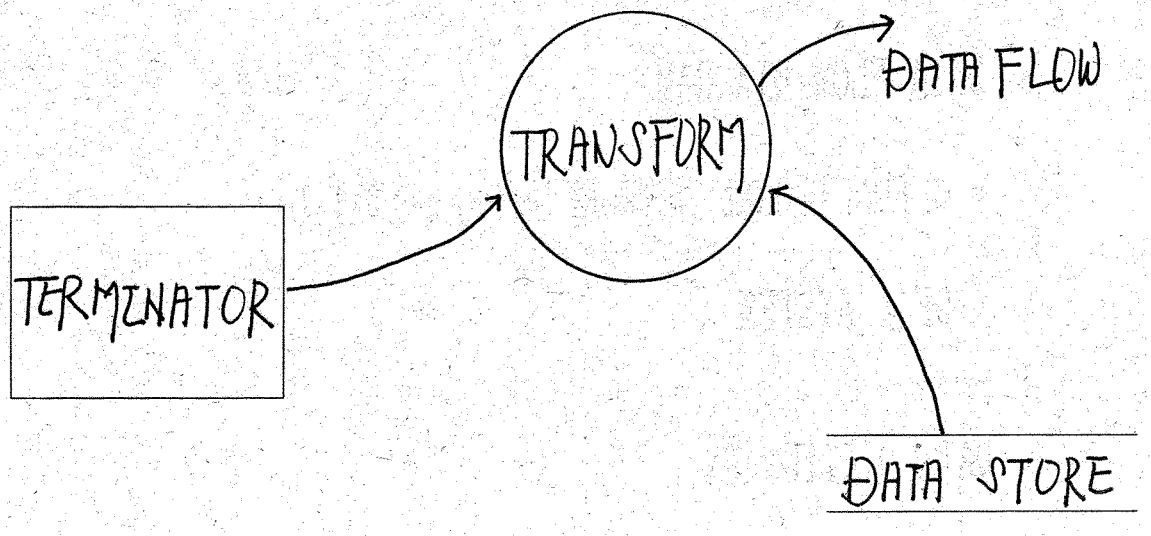
▷ DATA DICTIONARY

- A SET OF DEFINITION OF INTERFACES DECLARED IN THE MODEL

▷ STRUCTURED ENGLISH

- AN ENGLISH LANGUAGE SUBSET THAT UTILIZES A LIMITED VOCABULARY AND A LIMITED SYNTAX

DATA FLOW DIAGRAM의 NOTATION



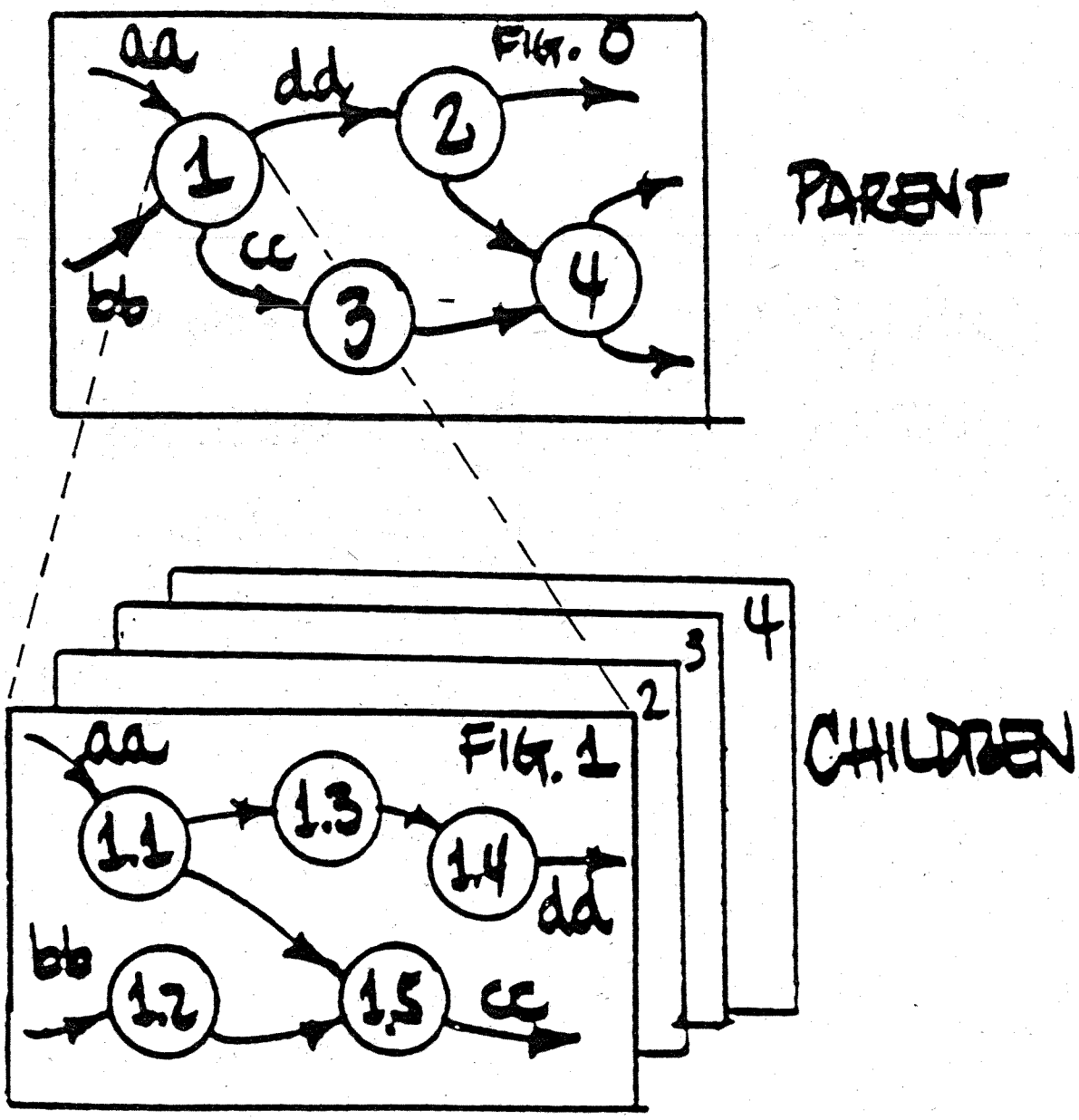
DATA FLOW : DATA가 흐르는 통로

TRANSFORM : 하나 이상의 INCOMING DATA FLOW를
OUTGOING DATA FLOW로 변하는 PROCESS

DATA STORE : 정보의 TIME-DELAYED REPOSITORY

TERMINATOR : SYSTEM DATA의 송신 및 수신처

DATA FLOW DIAGRAM의 LEVELING



DATA DICTIONARY의 NOTATION

DATA CONSTRUCT	NOTATION	MEANING
	=	IS COMPOSED OF
SEQUENCE	+	AND
SELECTION	[|]	EITHER-OR
REPETITION	{ }^{n}	n-REPETITIONS
	( )	OPTIONAL

DATA DICTIONARY의 예

KEYED PHONE NUMBER =

[LOCAL EXTENSION | OUTSIDE NUMBER | 0]

LOCAL EXTENSION =

[2001 | 2002 | ..... | 2999 | CONFERENCE SET]

OUTSIDE NUMBER =

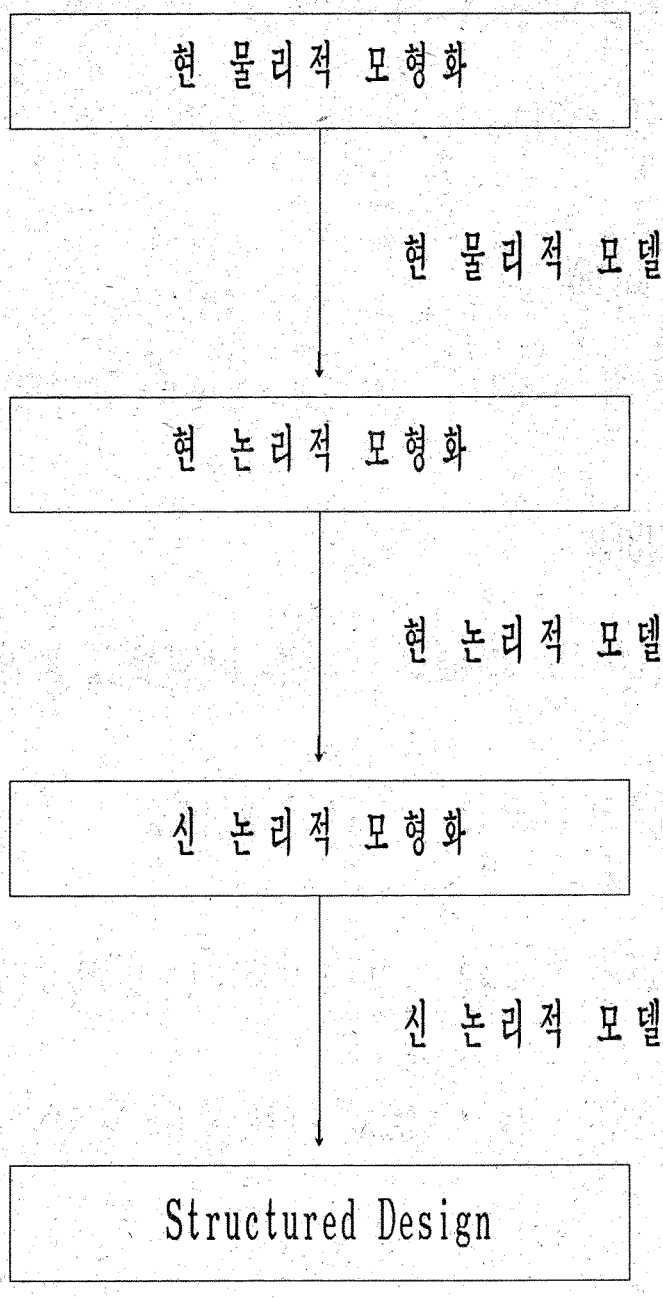
9 + [LOCAL NUMBER | LONG DISTANCE NUMBER]

LOCAL NUMBER = PREFIX + ACCESS NUMBER

LOGAL DISTANCE NUMBER = AREA CODE + LOCAL NUMBER

$$\text{CONFERENCE SET} = \{\# + \text{LOCAL EXTENSION} + \#(\#)\}_{2}^{6}$$

Structured Analysis의 절차



DATA-STRUCTURE ORIENTED ANALYSIS

▷ OBJECT와 OPERATION을 식별

▷ INFORMATION DOMAIN을 분석하여 모델링

▷ SEQUENCE, SELECTION, REPETITION에 의해 INFORMATION STRUCTURE를 표현

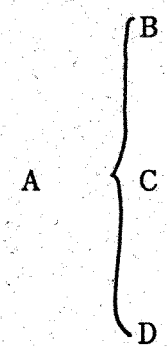
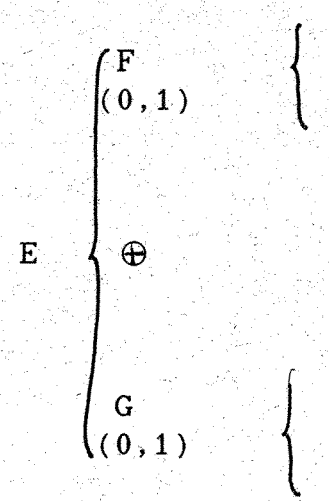
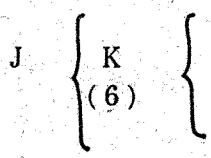
▷ INFORMATION STRUCTURE에서 PROGRAM STRUCTURE로 MAPPING

▷ 대표적 기법

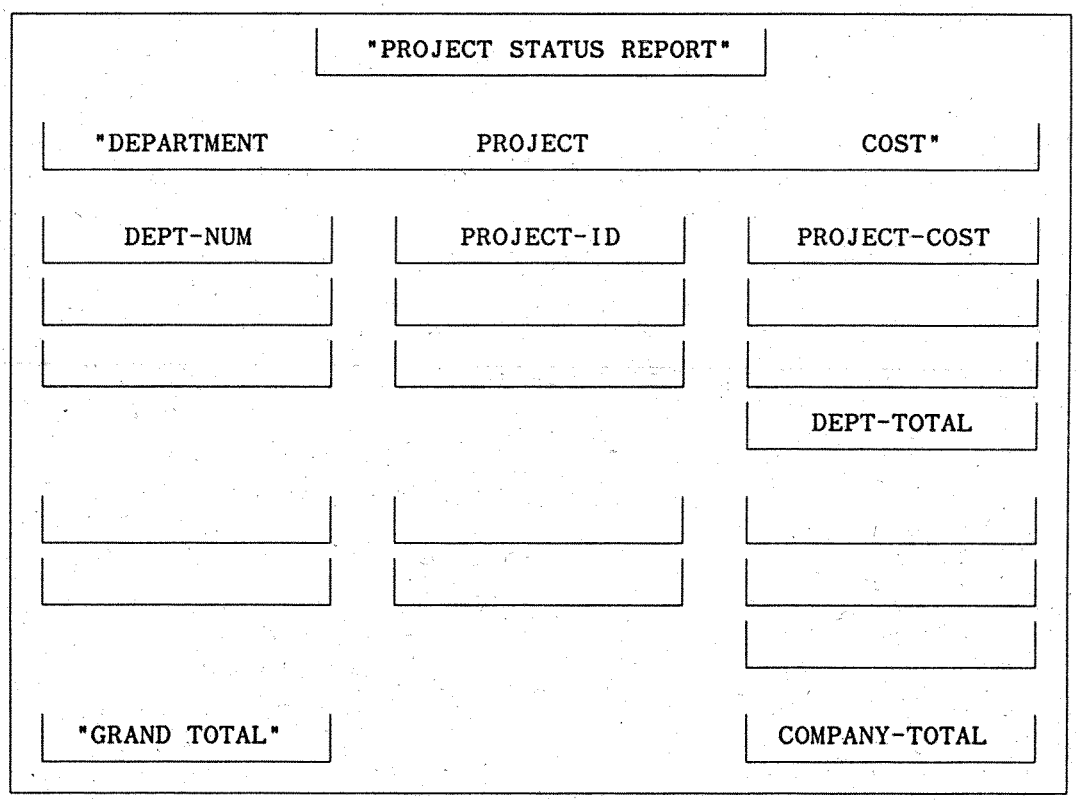
- DATA STRUCTURE SYSTEM DEVELOPMENT (D S S D)
- JACKSON SYSTEM DEVELOPMENT (J S D)

D S S D 에서의 NOTATION

▷ WARNIER DIAGRAM

SEQUENCE	ALTERNATION	REPETITION
		
A consists of B followed by C followed by D	E consists of either F or G but not both	J consists of K repeated 6 times

DATA STRUCTURE의 표현



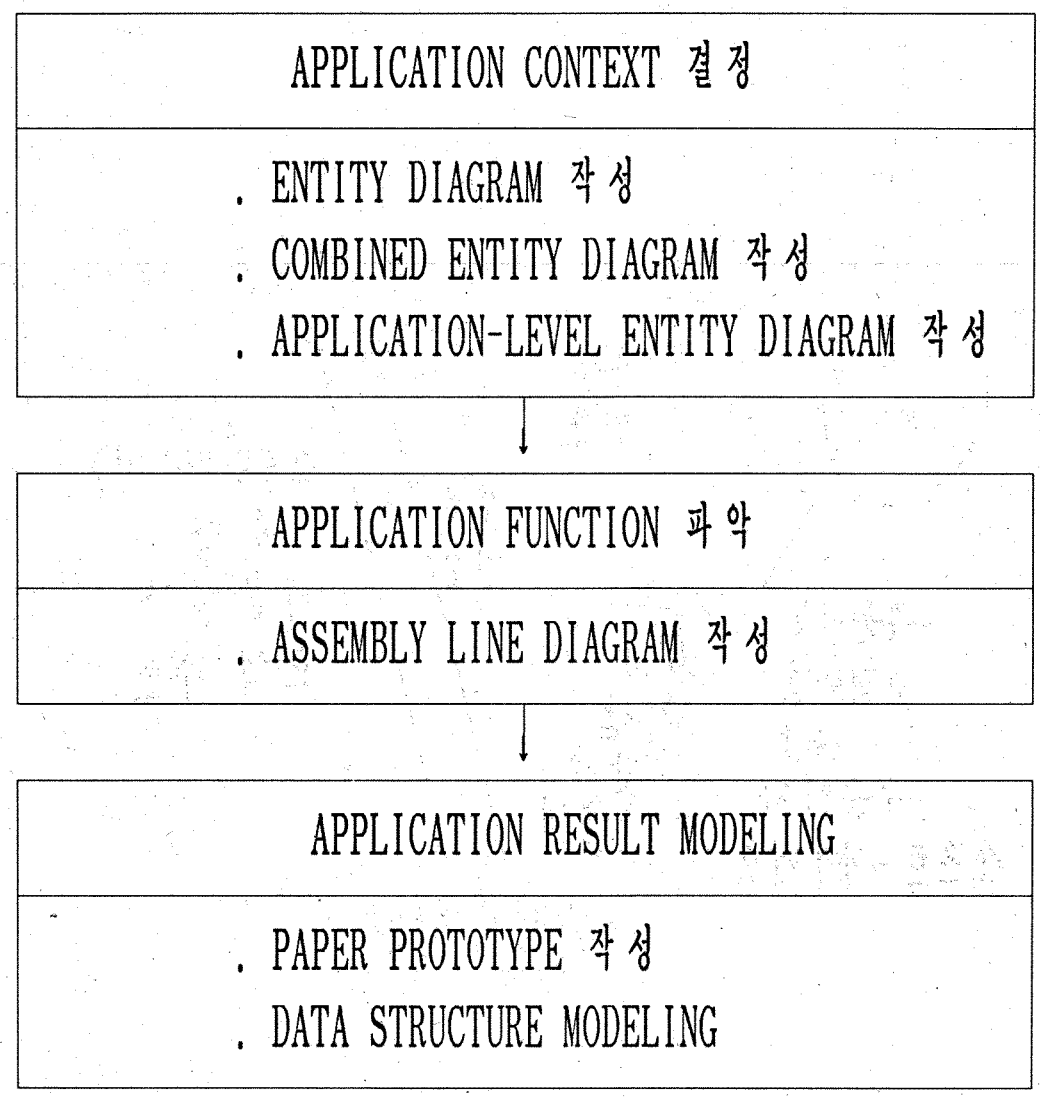

```
                  "PROJECT STATUS REPORT"
                  "DEPARTMENT    PROJECT    COST"
COMPANY  <        DEPT      <   PROJECT   <   DEPT-NUM
                  (1,D)         (1,P)         PROJECT-ID
                                              PROJECT-COST
                                .TOTAL
                  "GRAND TOTAL"
                  .TOTAL
```

PROGRAM STRUCTURE 의 표현

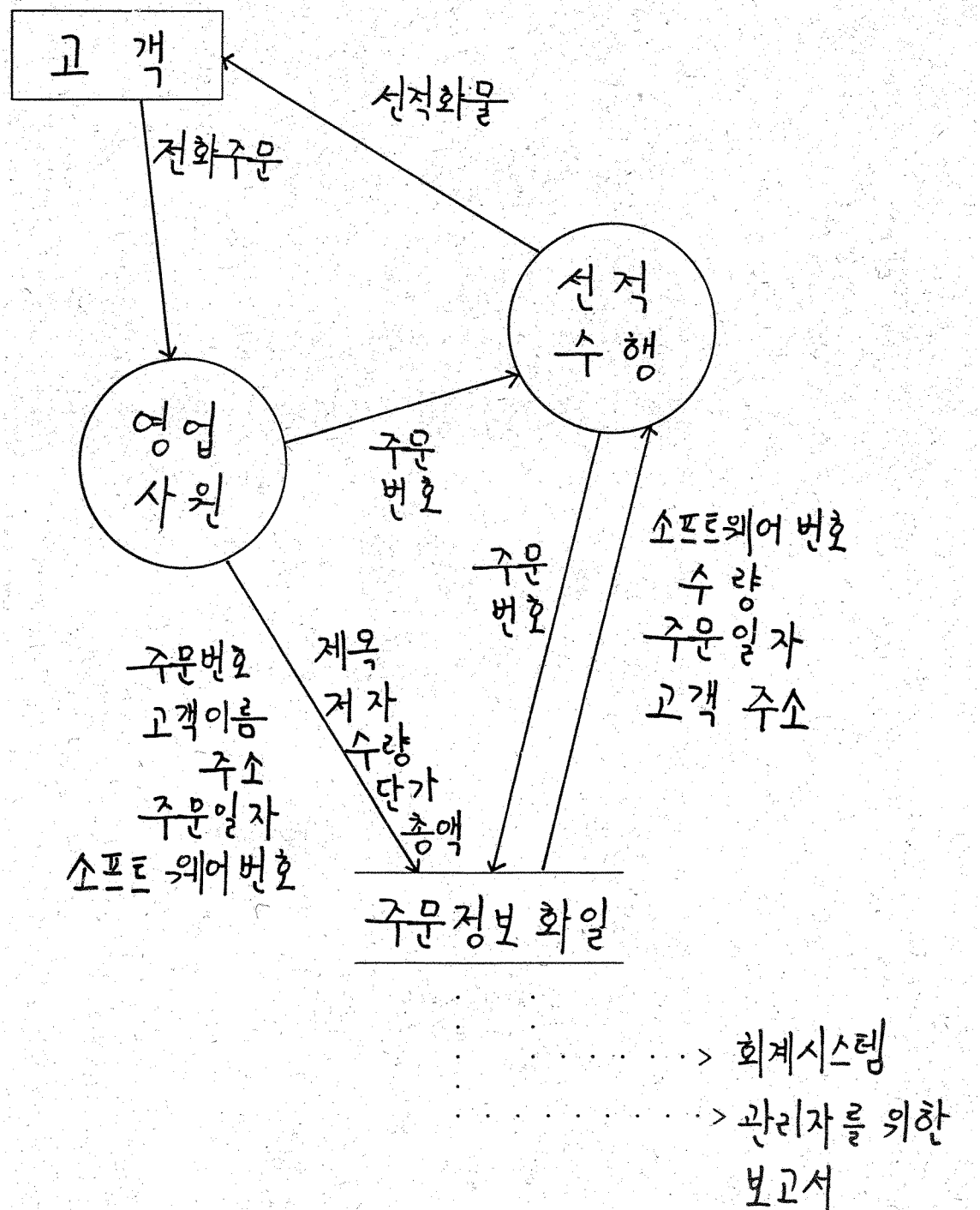
```
        LIST OF EMPLOYEE LOANS

ABEL                      350.00
CUMMINGS                 2625.73
MIKHANOV                  985.08
   .                         .
   .                         .
   .                         .

        END OF LOAN LIST
```

```
                    .BEGIN      DISPLY "LIST OF EMPLOYEE LOANS"
                                OPEN INPUT LOAN-FILE
                                MOVE "NO" TO EOF
                                READ LOAN-FILE
                                  AT END MOVE "YES" TO EOF

                    LOAN        MOVE LOAN-AMOUNT TO EDITED-AMOUNT
                    (0,L)/?1    DISPLAY EE-NAME,EDITED-AMOUNT
                                READ LOAN-FILE
                                   AT END MOVE "YES" TO EOF
COMPANY

                    .END        DISPLAY "    END OF LOAN LIST"

                                CLOSE LOAN-FILE

                ?1/ UNTIL EOF = "YES"
```

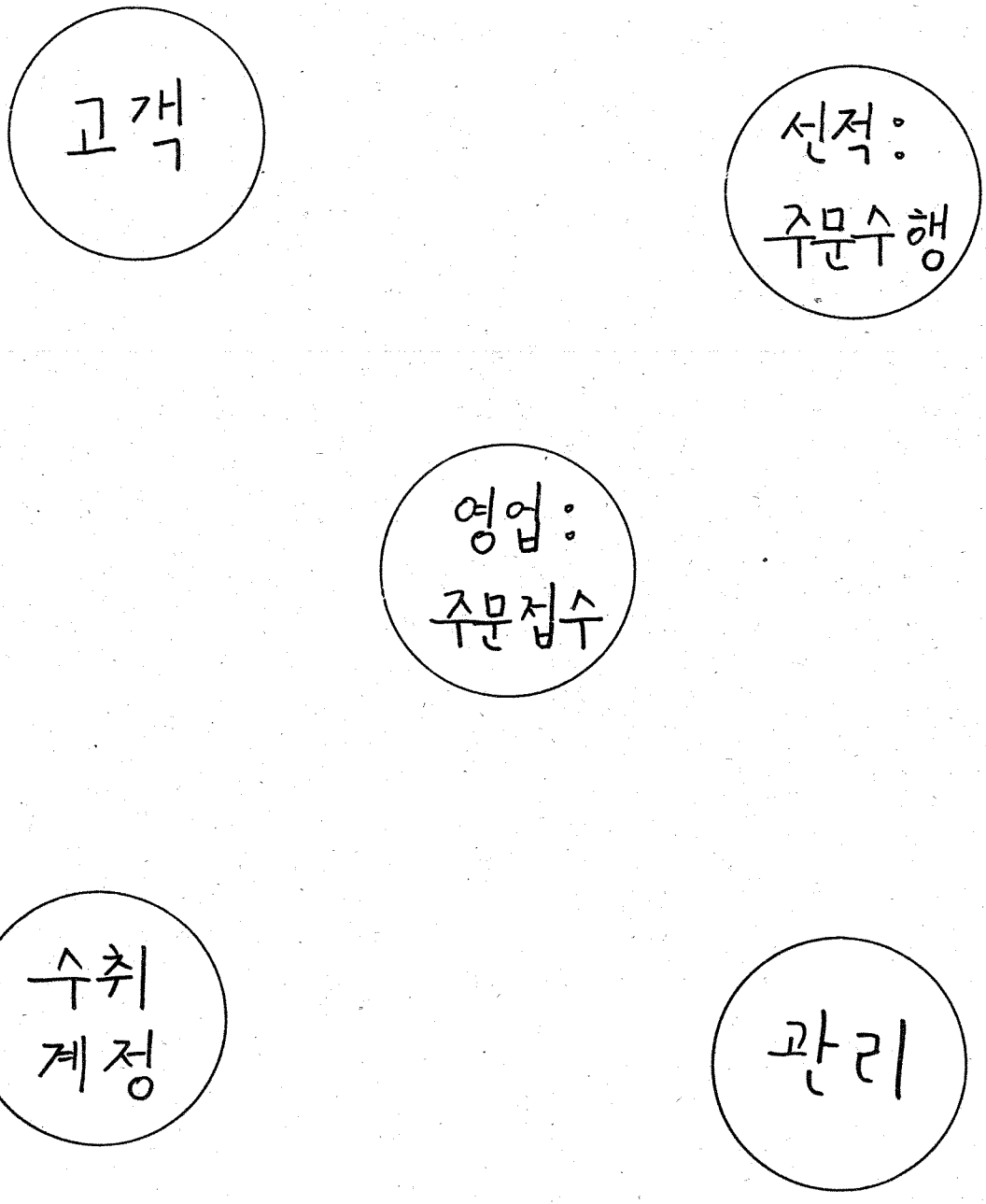
D S S D 의 분석 절차


APPLICATION CONTEXT 결정
. ENTITY DIAGRAM 작성
. COMBINED ENTITY DIAGRAM 작성
. APPLICATION-LEVEL ENTITY DIAGRAM 작성
APPLICATION FUNCTION 파악
. ASSEMBLY LINE DIAGRAM 작성
APPLICATION RESULT MODELING
. PAPER PROTOTYPE 작성
. DATA STRUCTURE MODELING

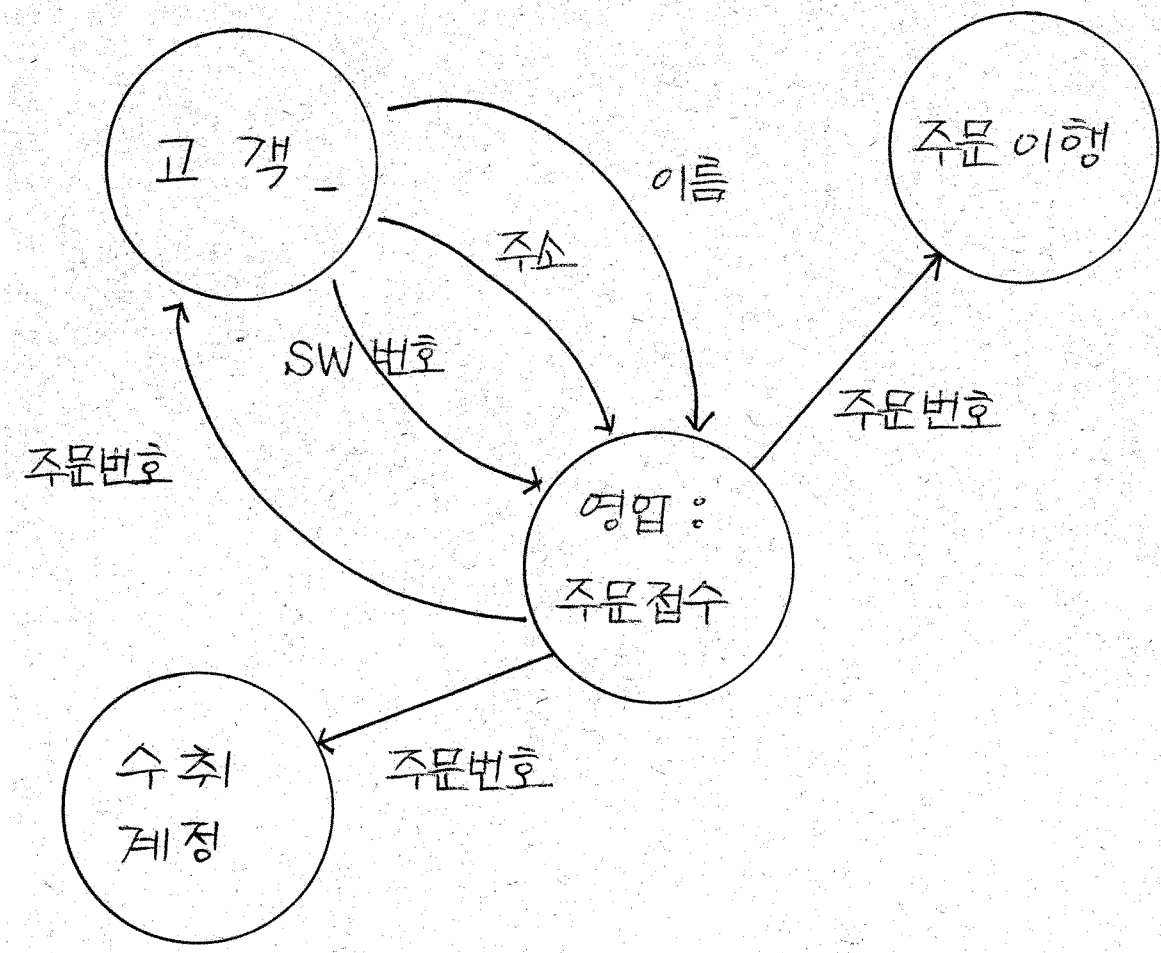
D S S D EXAMPLE



ENTITY의 식별

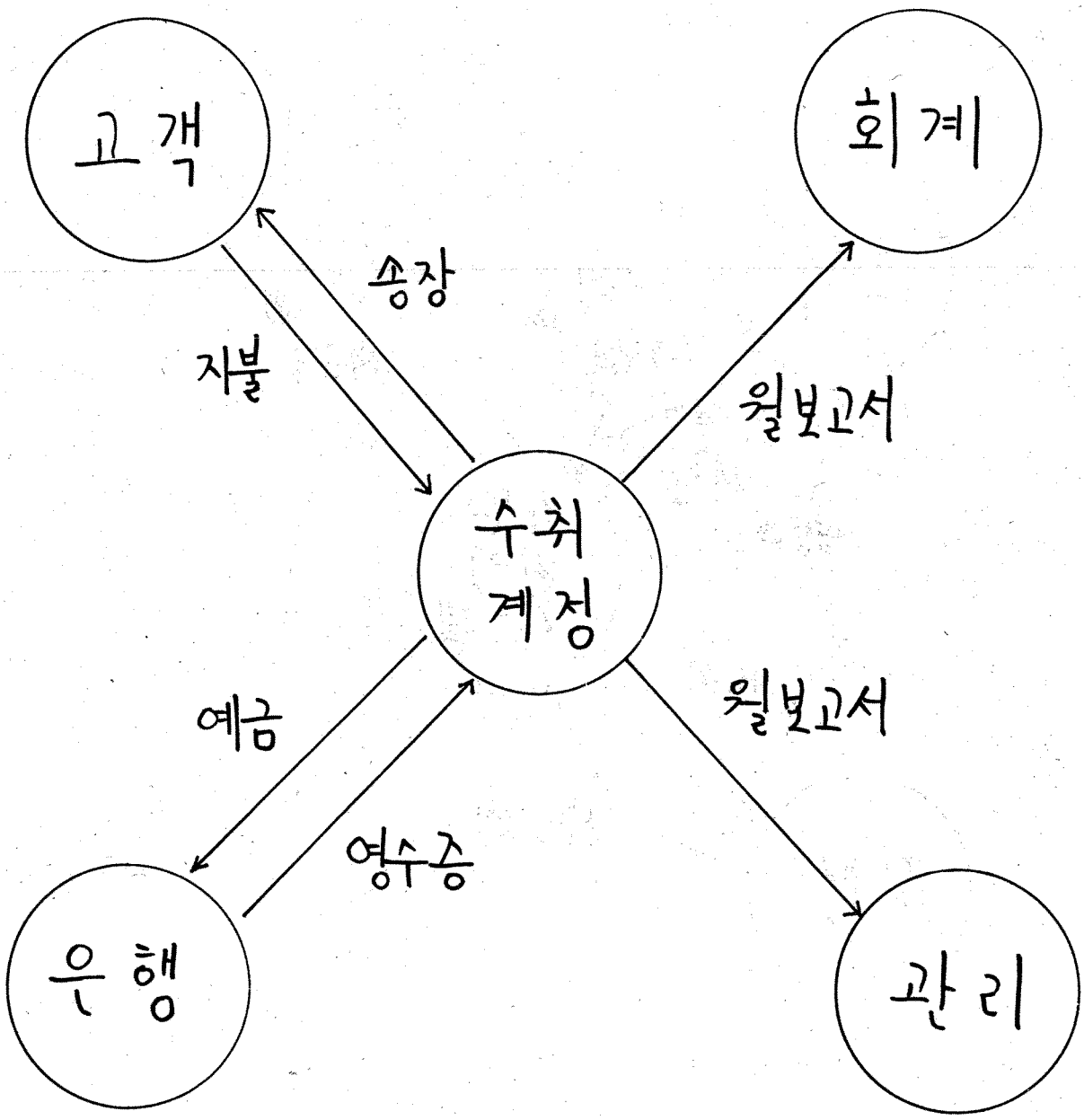


주문접수 ENTITY DIAGRAM의 작성

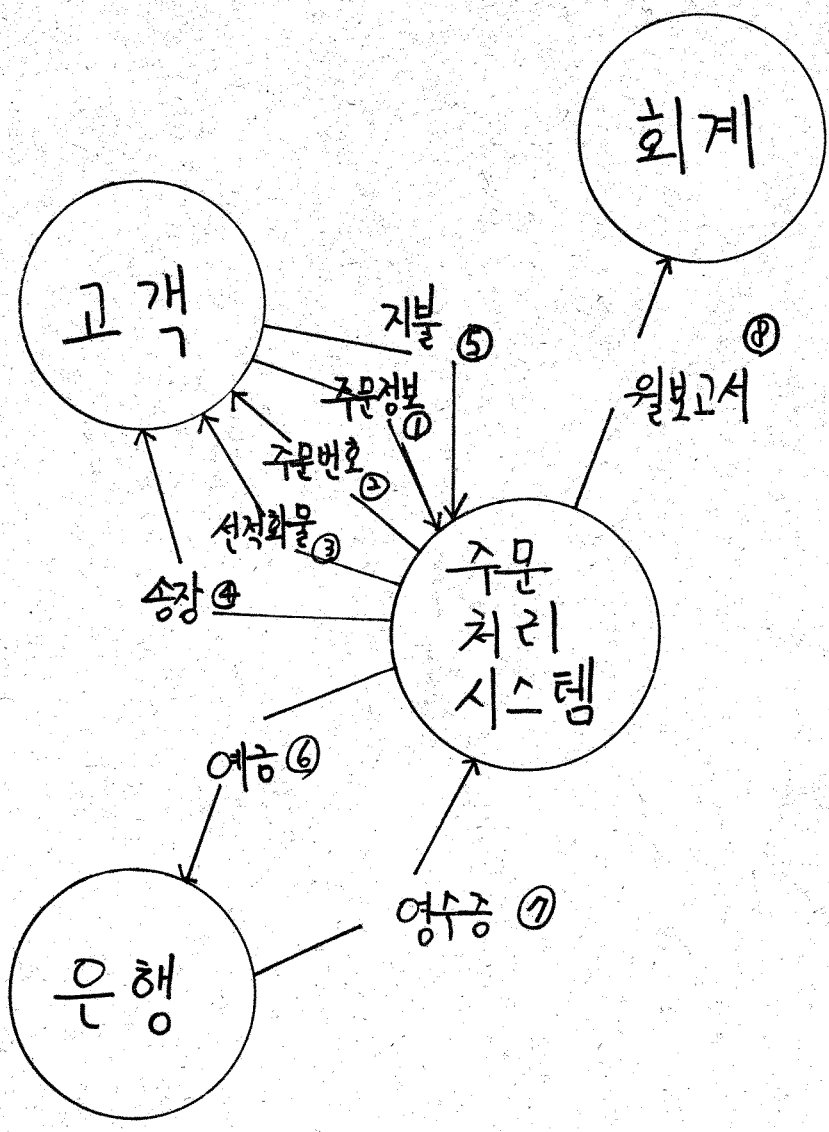


주문정보 = 이름 +
주소 +
SW번호

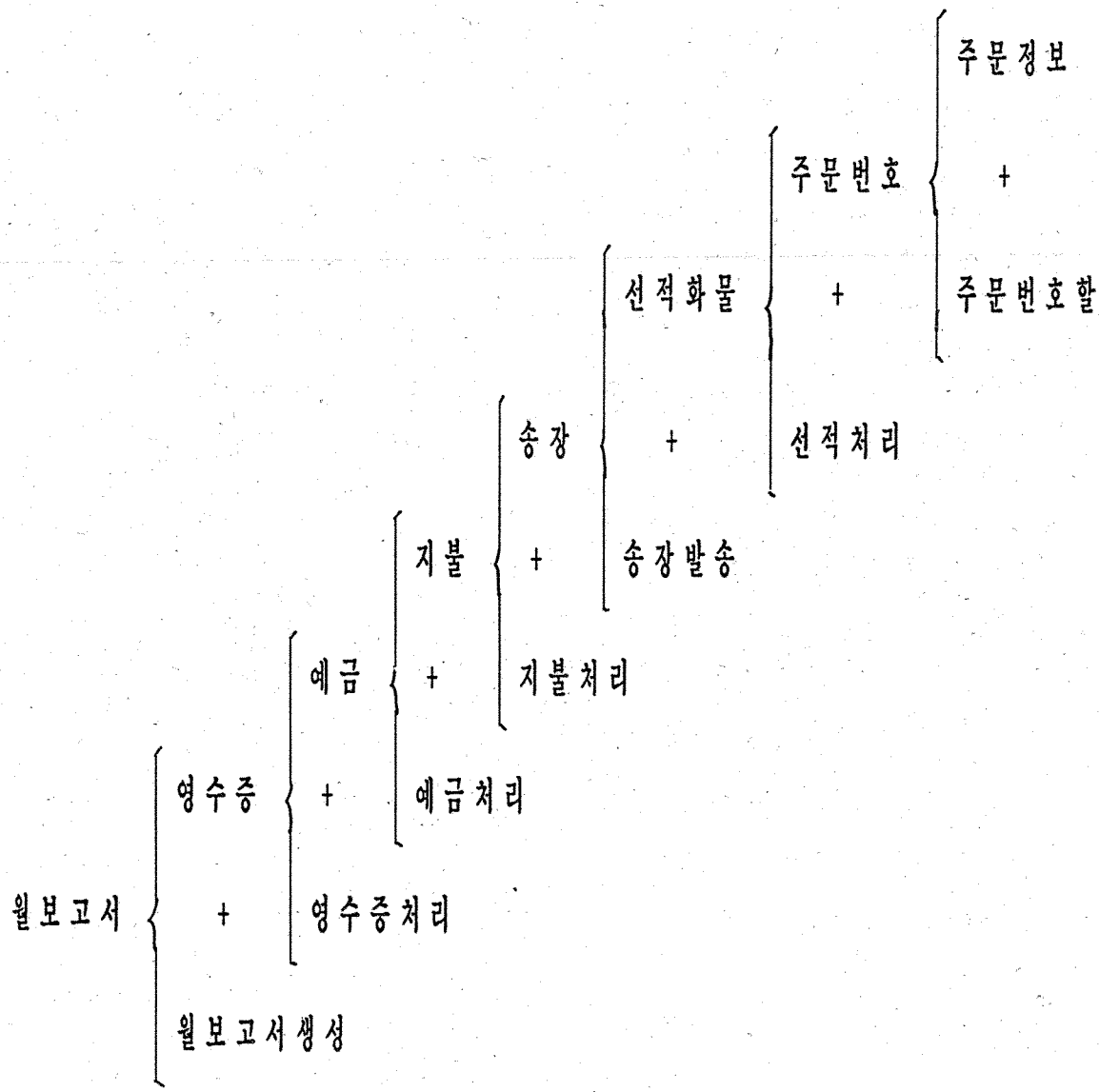
수취계정 ENTITY DIAGRAM의 작성



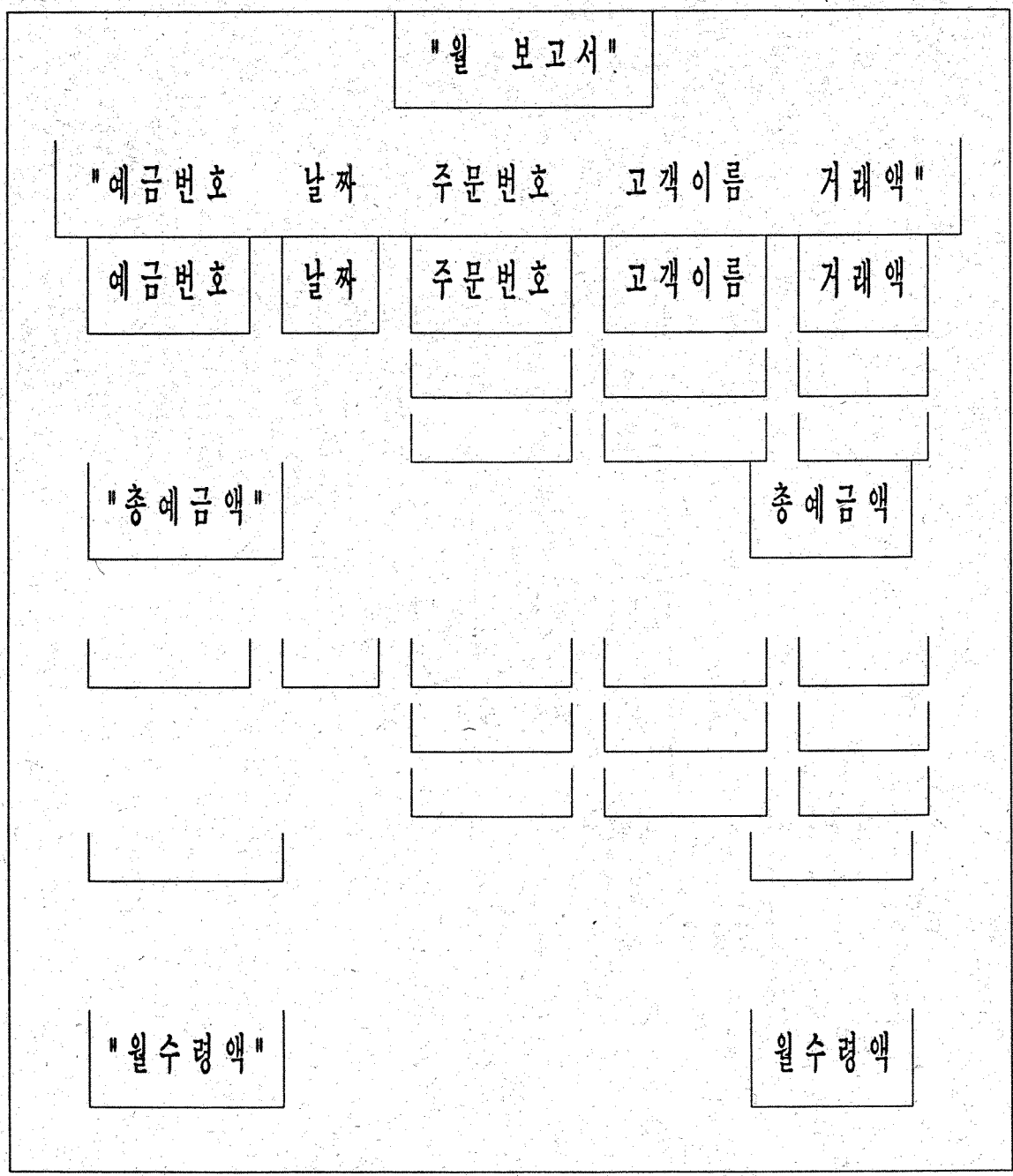
APPLICATION-LEVEL ENTITY DIAGRAM의 작성



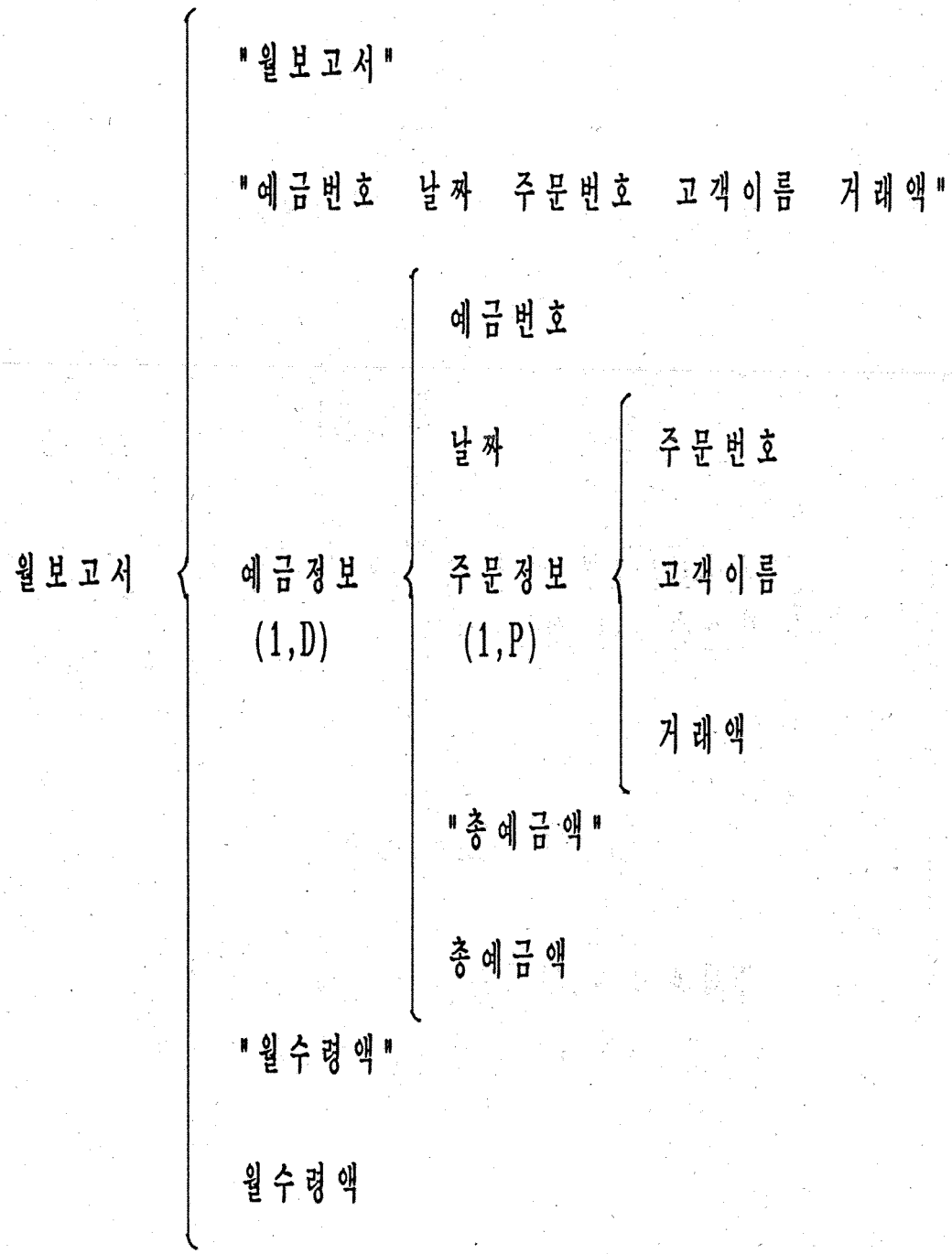
ASSEMBLY LINE DIAGRAM의 작성



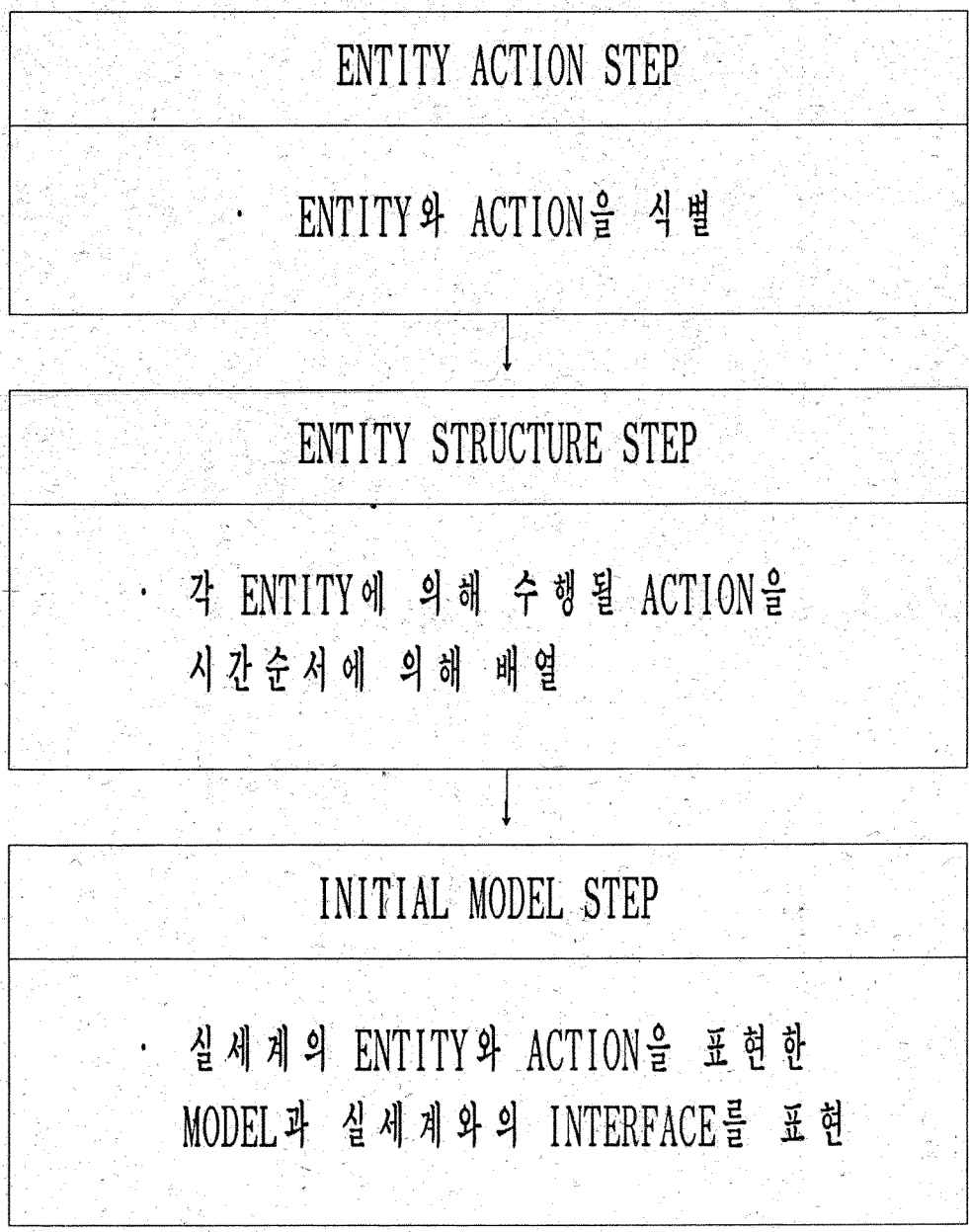
PAPER PROTOTYPE의 작성



DATA STRUCTURE MODELING



J S D 의 분석 절차


ENTITY ACTION STEP
· ENTITY와 ACTION을 식별
ENTITY STRUCTURE STEP
· 각 ENTITY에 의해 수행될 ACTION을
시간순서에 의해 배열
INITIAL MODEL STEP
· 실세계의 ENTITY와 ACTION을 표현한
MODEL과 실세계와의 INTERFACE를 표현

Entity와 Action

▷ Entity

- 시간 순서로 Action을 수행하는 사물
- 실세계에 존재
- 개별적인 사물로 간주가 가능해야 함

▷ Action

- 시스템 외부의 실세계에서 발생
- 특정 시점에 발생
- Subaction으로의 분할 불가능해야 함

ENTITY ACTION STEP

▷ READER

SUBSCRIBE : 1년분 구독료를 지불함으로써 정규고객이 되는 것

ENTER : 시합을 위한 등록서류를 만들고 이를 회사에 보내는 것

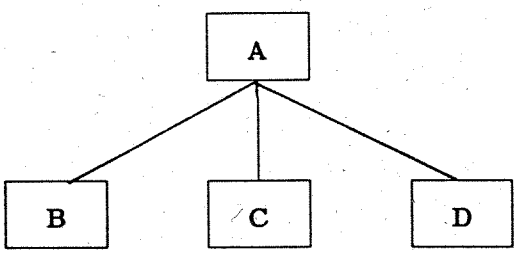
▷ PANEL

AWARD : 참가자에게 상을 수여

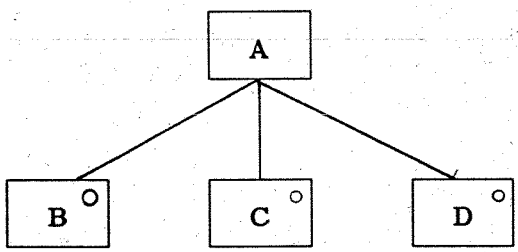
MEET : 1주에 한번 함께 모여 참가서류를 받고 판정

DISPERSE : 보고서를 산출하고 다음 모임을 위해 해산

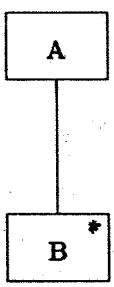
STRUCTURE DIAGRAM과 TEXT



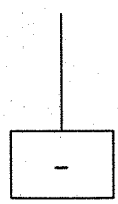

```
A  s e q
   B ;
   C ;
   D ;
A  e n d
```



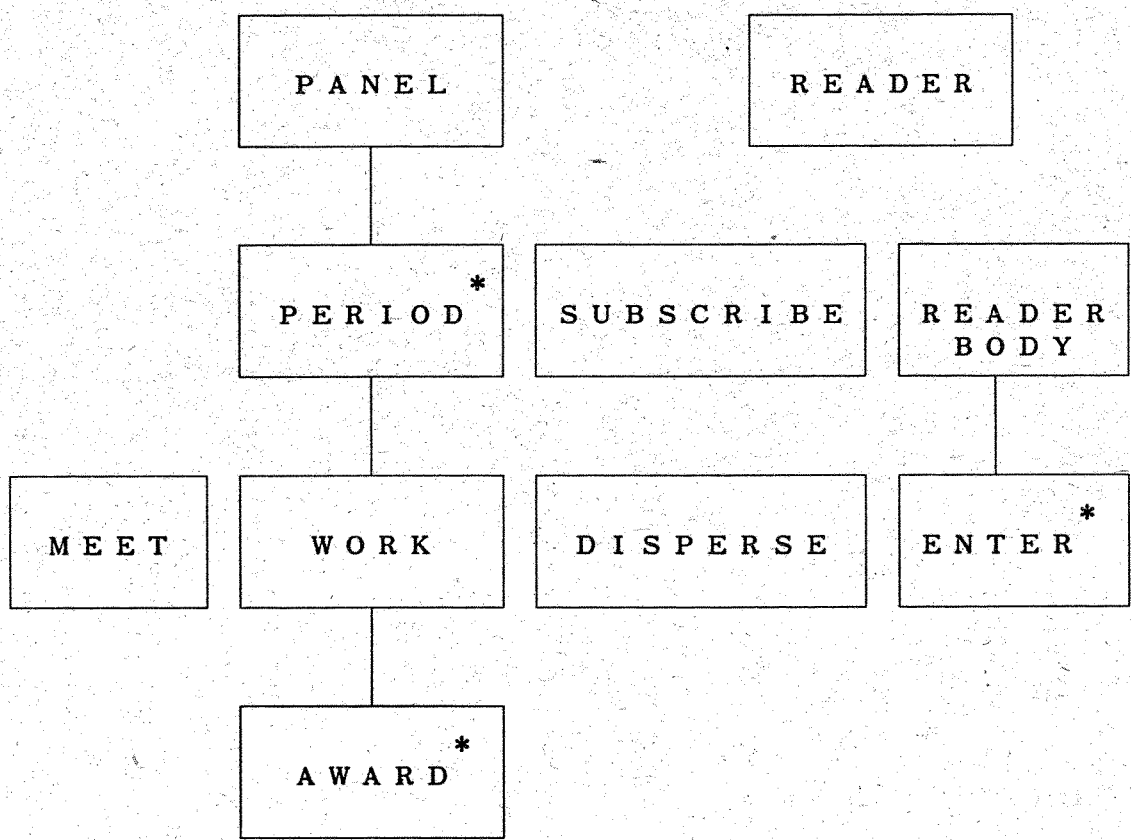

```
A  s e l ( c o n d - B )
   B ;
A  a l t ( c o n d - C )
   C ;
A  a l t ( c o n d - D )
   D ;
A  e n d
```




```
A  i t r  w h i l e ( c o n d - B )
   B ;
A  e n d
```



ENTITY STRUCTURE STEP

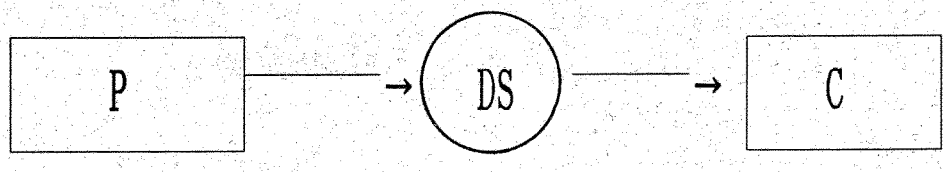


ENTITY STRUCTURE STEP

```
PANEL itr
    PERIOD  seq
        MEET;
        WORK itr
            AWARD;
        WORK end
        DISPERSE;
    PERIOD end
PANEL end
```

```
READER seq
    SUBSCRIBE;
    READER-BODY itr
        ENTER;
    READER-BODY end
READER end
```

Data Stream Connection



▷ P는 메세지 또는 레코드의 순서화된 집합으로 구성된 Data Stream을 Write하고, C는 이 Stream을 읽는다.

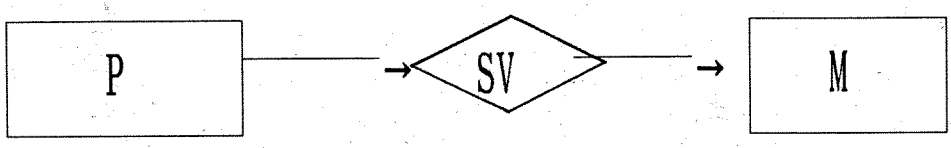
▷ P의 Structure Text에

write RP to DS ; 첨가

▷ C의 Structure Text에

read DS ; 첨가

State Vector Connection



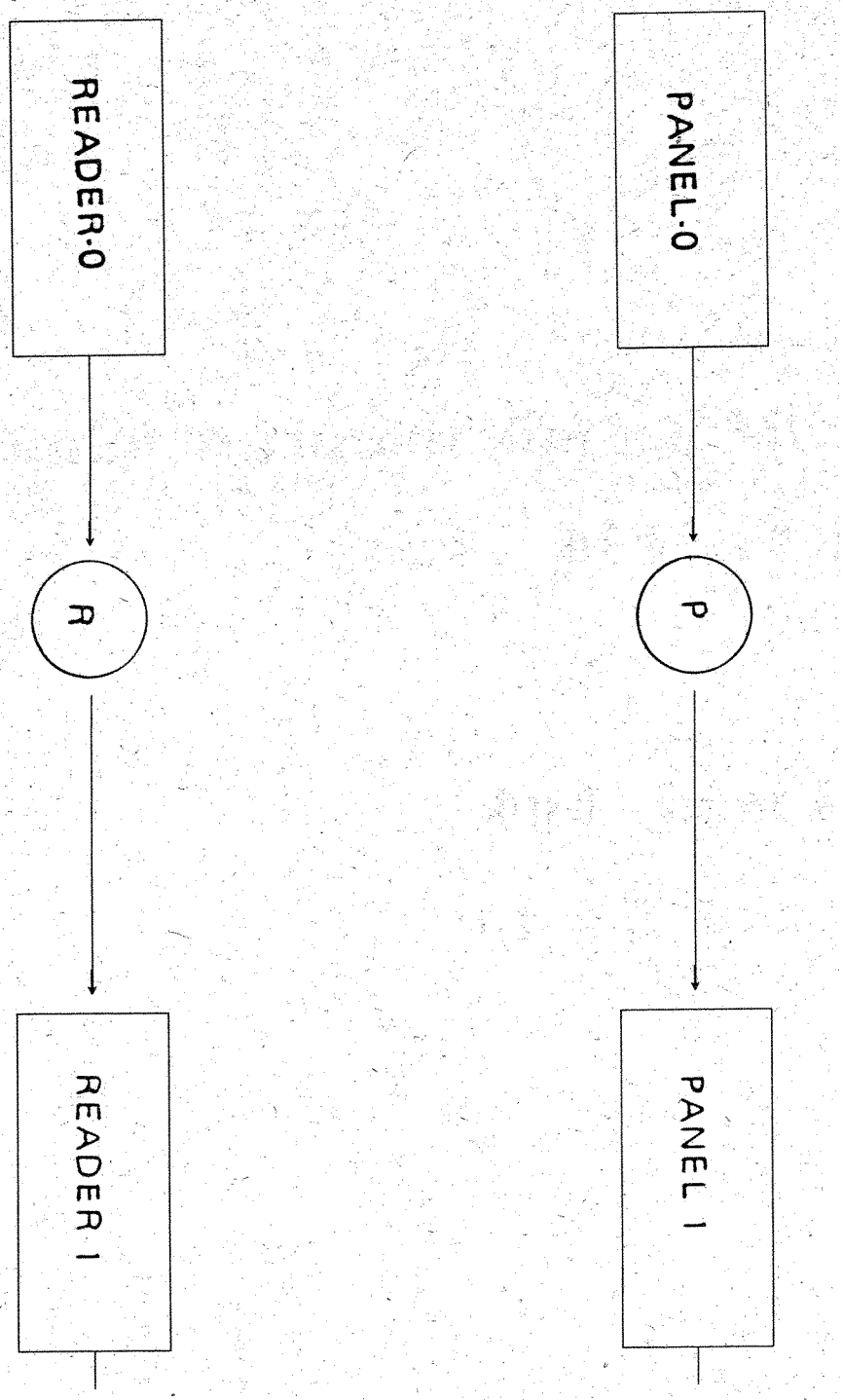
▷ M은 P의 State Vector (Internal Local Variable)

를 직접 관찰한다.

▷ M의 Structure Text에

getsv SV ; 첨가

INITIAL MODEL STEP

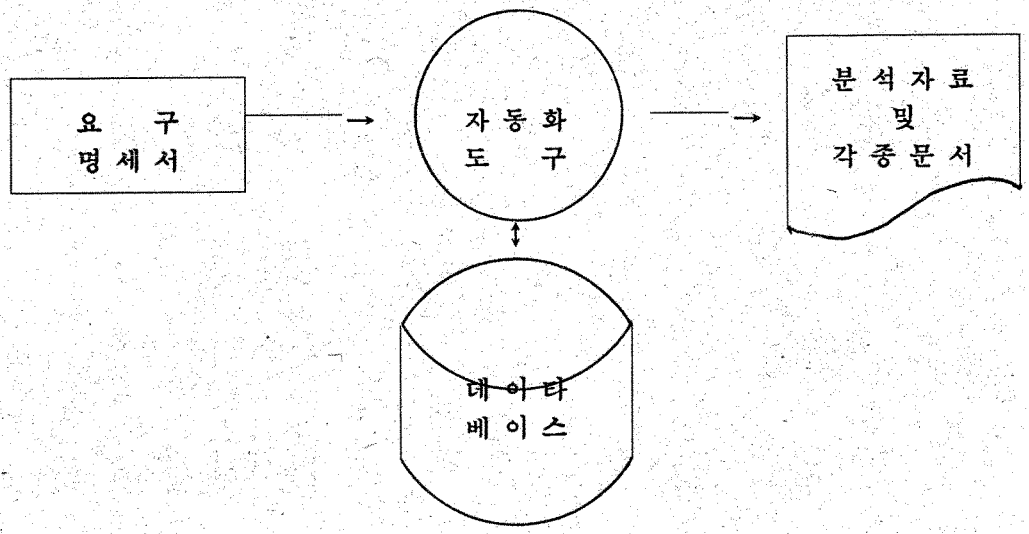


INITIAL MODEL STEP

```
PANEL-1 itr
    read P;
    PERIOD seq
        MEET;read P;
        WORK itr
          AWARD;read P;
        WORK end
        DISPERSE;
    PERIOD end;
PANEL-1 end
```

```
READER-1 seq
    read R;
    SUBSCRIBE;read B;
    READER-BODY itr
      ENTER;read R;
    READER-BODY end
READER-1 end
```

요구분석용 자동화 도구



▷ 표준화 및 보고서 생성을 통한 문서의 질 향상

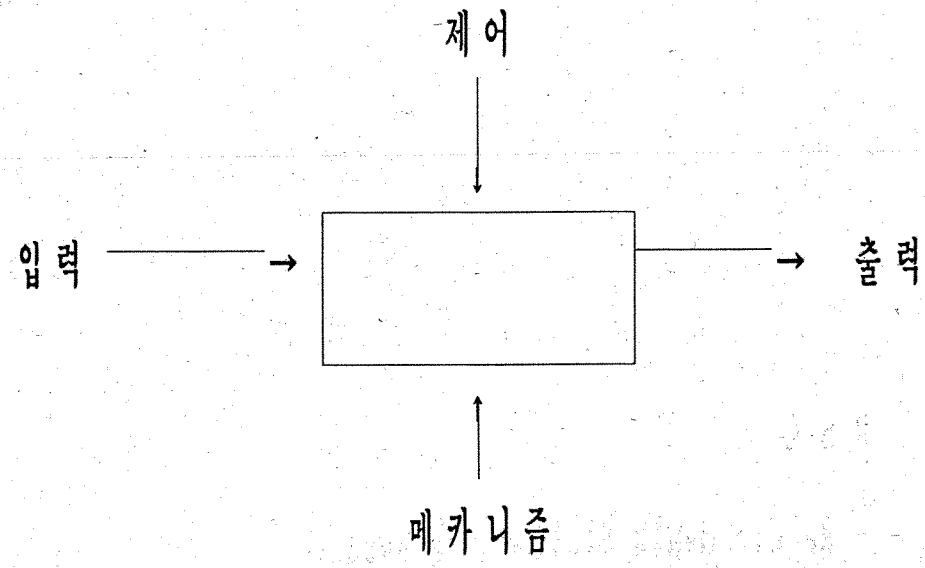
▷ 데이타베이스를 통한 분석가간의 용이한 협조

▷ 불일치성 및 불완전성의 용이한 파악

▷ 명세서의 유지보수 비용 절감

▷ 수정에 의해 영향을 받는 사항들을 쉽게 추적

▷ Box와 Arrow로 표현되는 명세서

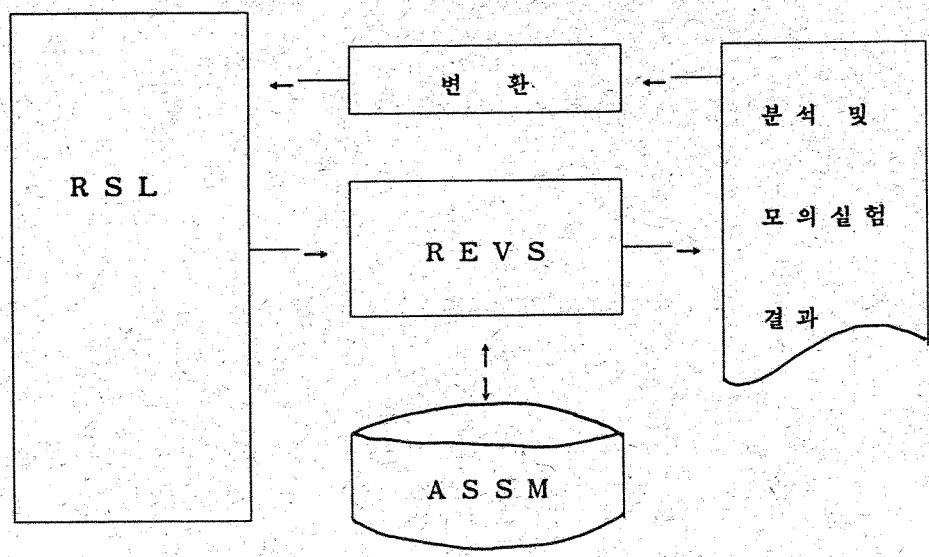


Actigram과 Datagram

▷ 명세서 작성을 위한 방법론

- ICOM CODE
- SA Tie Process
- Support Arrow와 Call Arrow

Software Requirements Engineering Methodology



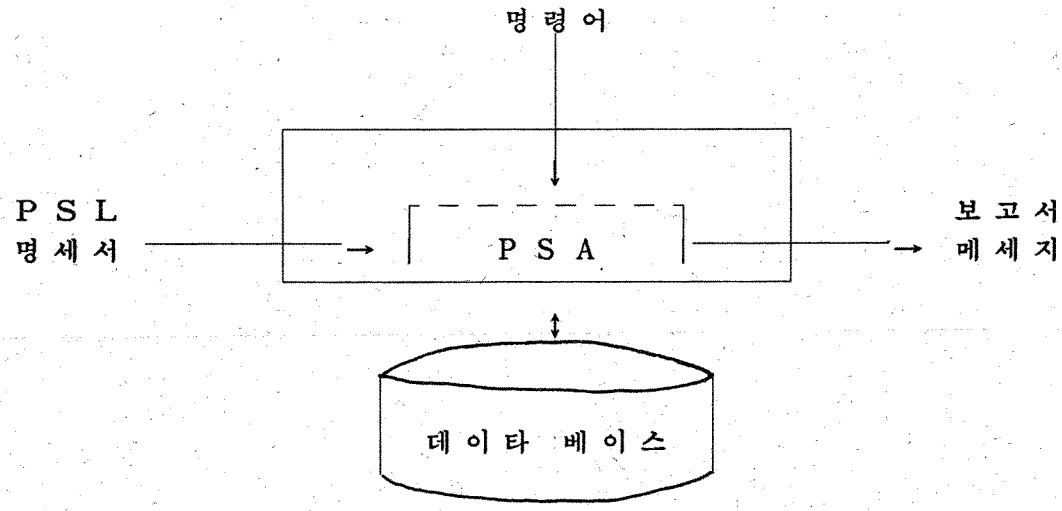
▷ R S L

- Requirements Statement Language
- 유한상태 기계를 기초로한 명세언어

▷ R E V S

- Requirements Engineering Validation System
- RSL의 내용을 ASSM에 수록하고, 이를 통해 분석및 모의 실험을 수행

P S L/P S A



▷ P S L

- Problem Statement Language

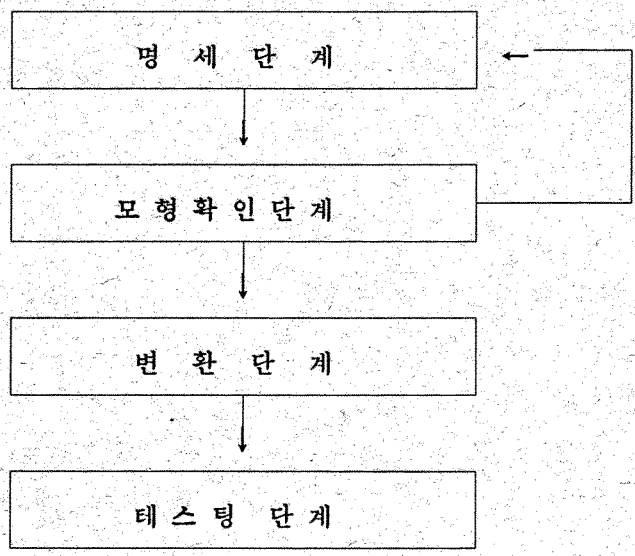
- 데 이 타 흐 름 모 델 을 기 초 로 한 명 세 언 어

▷ P S A

- Problem Statement Analyzer

- PSL의 내 용 을 데 이 타 베 이 스 에 저 장 하 고, 이 를 통 해 각 종 분 석 및 보 고 서 를 산 출

TAGS



▷ IORL

- Input Output Requirements Language
- 도형과 테이블로 시스템을 기술하는 언어

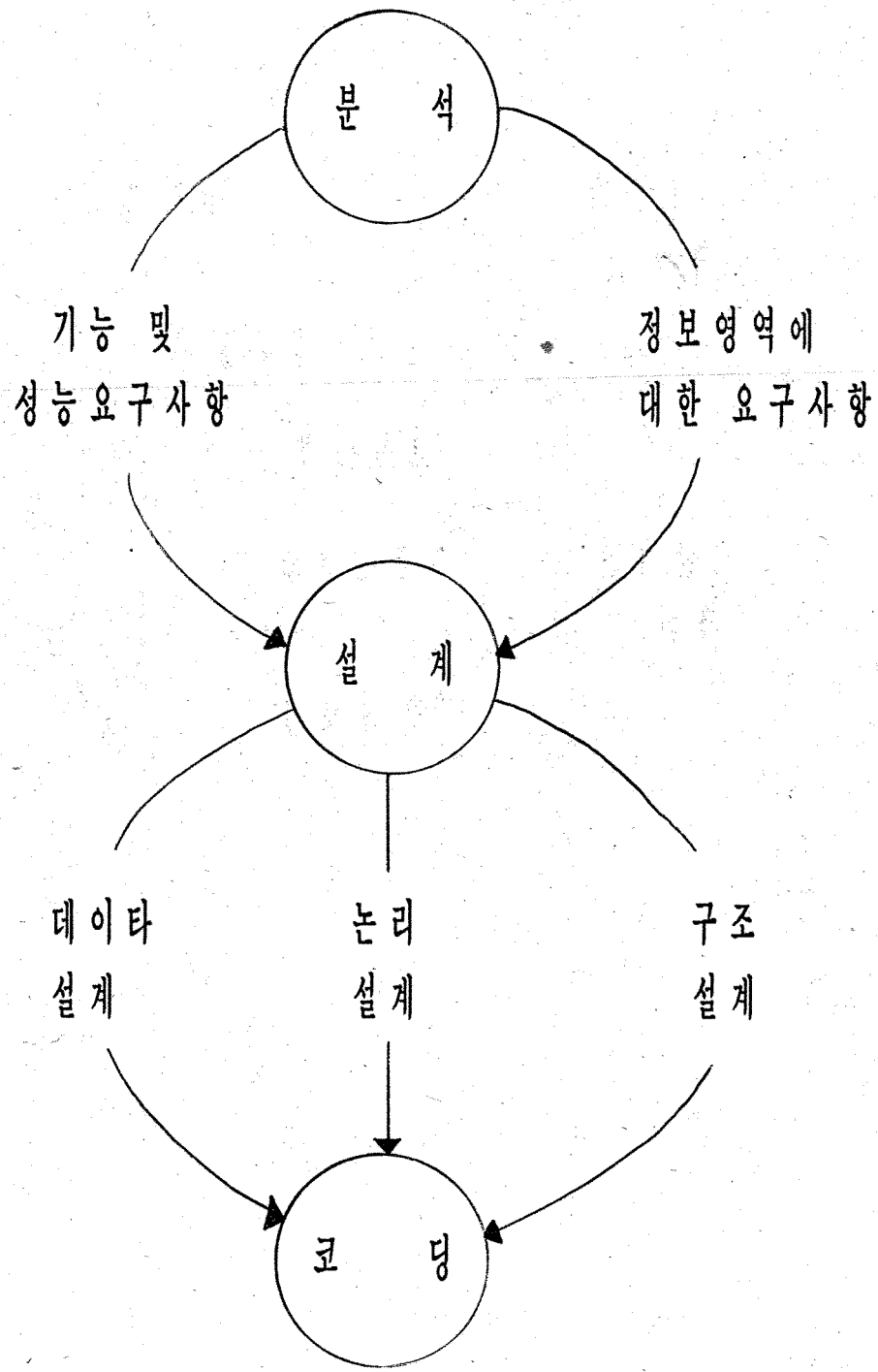
▷ 소프트웨어 도구

- IORL의 내용을 저장 및 검색토록하며,
  ADA언어를 생성하여 모의실험을 수행

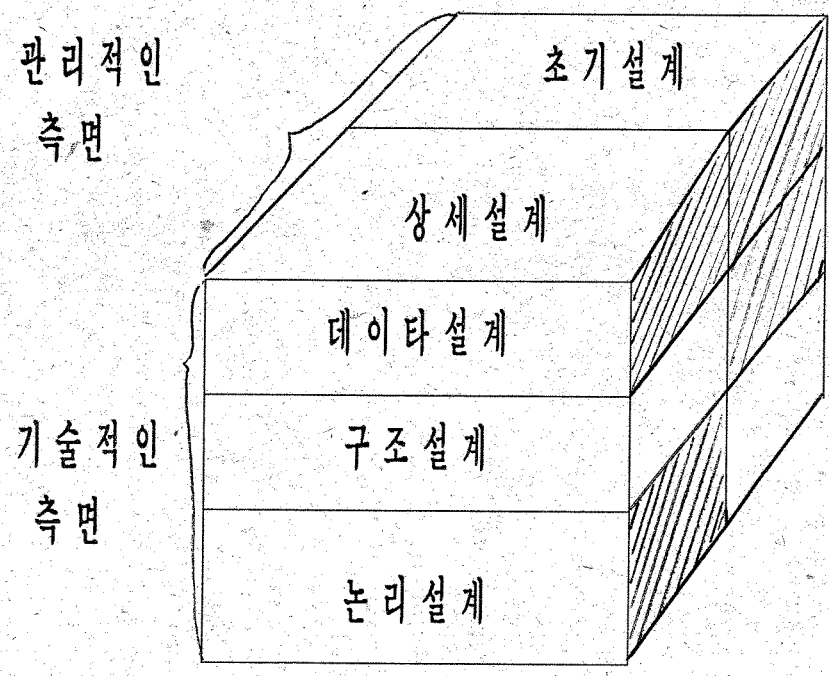
소프트웨어 설계

Software Design

소프트웨어 설계



소프트웨어 설계



양질의 설계를 위한 가이드라인

▷ 소프트웨어 구성요소간의 제어 관계를 잘 표현한 계층구조를 산출하라

▷ 소프트웨어를 단일 기능을 수행하는 구성요소들로 분할하라

▷ 프로시듀어와 데이타는 분리하여 기술하라

▷ 모듈은 서로 독립적인 기능을 갖도록 유도하라

▷ 요구분석에서 산출된 정보를 사용하여 설계할 수 있는 연계 가능한 방법을 사용하라

설계기법의 공통적 특성

- 정보의 표현을 설계의 표현으로 변환하기 위한 메카니즘
- 소프트웨어의 기능적 구성요소와 그들간의 인터페이스를 기술하기 위한 표기법
- Stepwise Refinement 와 Partition을 위한 경험적 방법
- 품질 평가를 위한 가이드라인

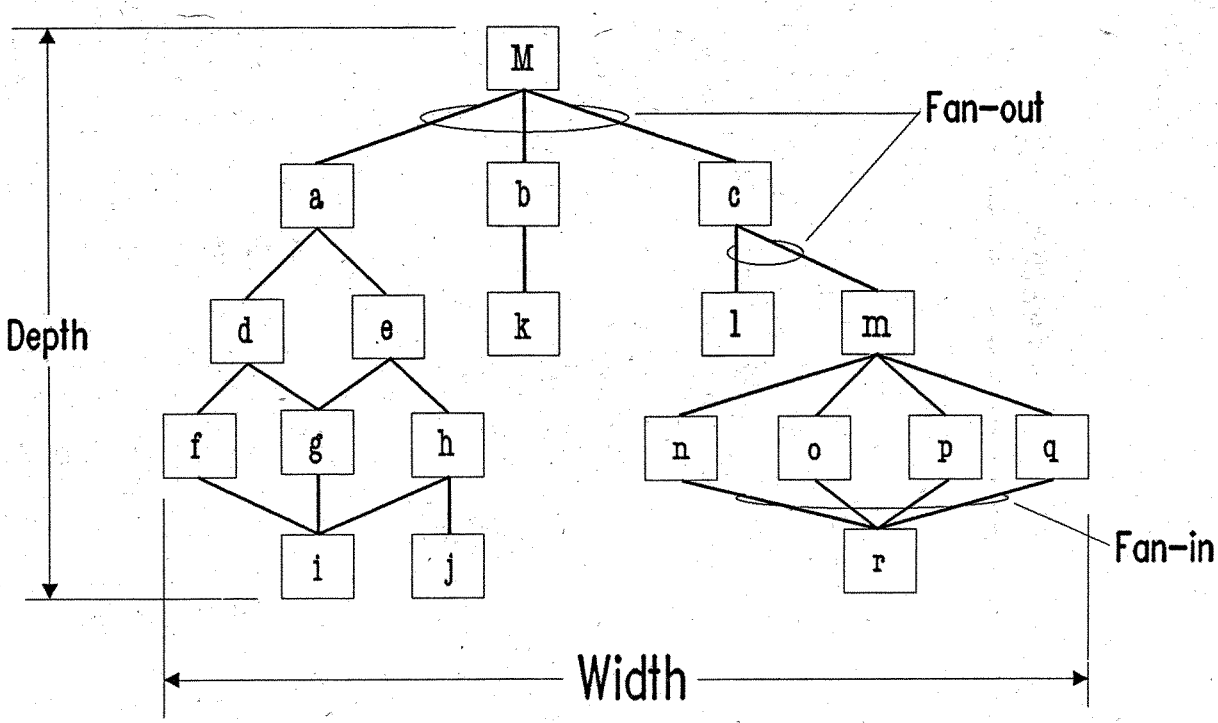
프로그램 구조 설계

▷ 프로그램 구성요소간의 제어 구조를 표현

▷ Structure Chart

Warnier-Orr Diagram

Jackson Diagram



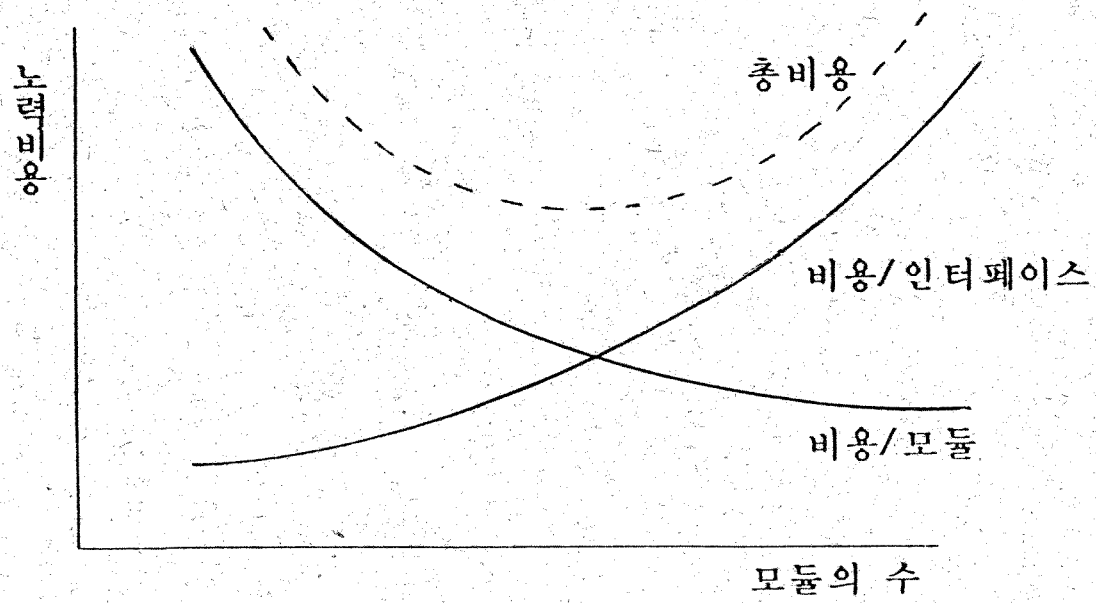
모듈성

▷ 문제 P의 복잡성을 C(P) 라하고, 이를 해결하기 위한 시간을 E(P)라 할때

C(P1) > C(P2) ---> E(P1) > E(P2),

C(P1+P2) > C(P1) + C(P2)라 하면

E(P1+P2) > E(P1) + E(P2)



추상화

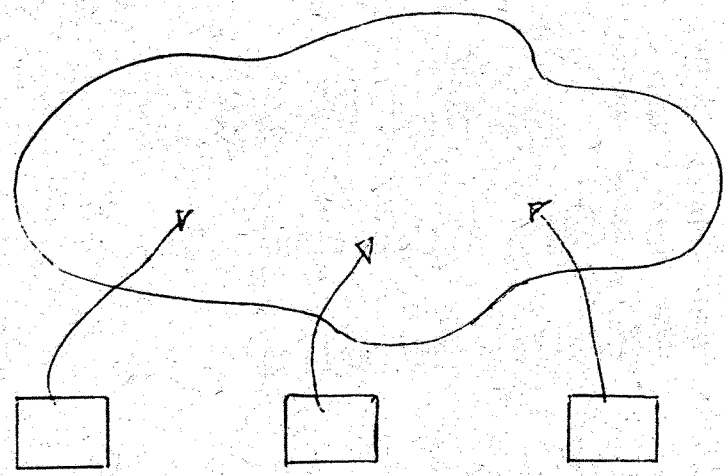
▷ 시스템의 특정한 사항 및 특성을 강조하고,
다른 사항을 억제함으로써 시스템을 간단히
기술하는 방법

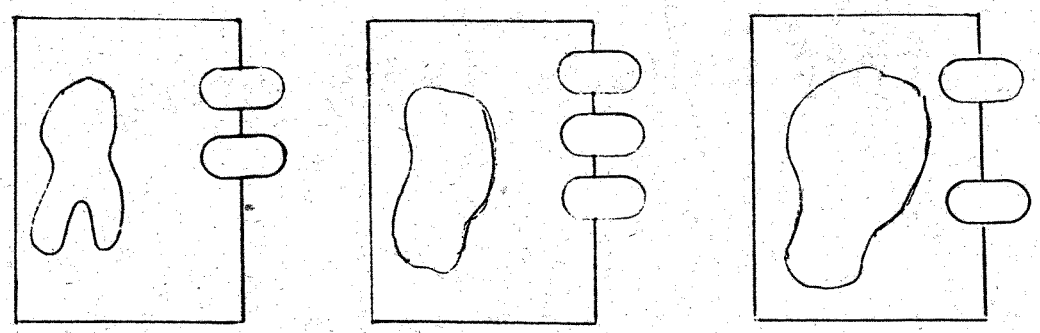
▷ 절차적 추상화(Procedural Abstraction)
데이타 추상화(Data Abstraction)
제어 추상화(Control Abstraction)

▷ 분류(Classification)/실례화(Instantiation)
집성(Aggregation)/분할(Decomposition)
일반화(Generalization)/특수화(Specialization)

정보은닉

▷ 다른 시스템 구성요소가 필요하지 않은 정보는 접근할 수 없도록 기술하는 방법





응집도

- ▷ 우연적 응집(Coincidental Cohesion)
- ▷ 논리적 응집(Logical Cohesion)
- ▷ 시간적 응집(Temporal Cohesion)
- ▷ 절차적 응집(Procedural Cohesion)
- ▷ 통신적 응집(Communication Cohesion)
- ▷ 순차적 응집(Sequential Cohesion)
- ▷ 기능적 응집(Functional Cohesion)

▷ 내용 결합(Content Coupling)

▷ 공유 결합(Common Coupling)

▷ 외부 결합(External Coupling)

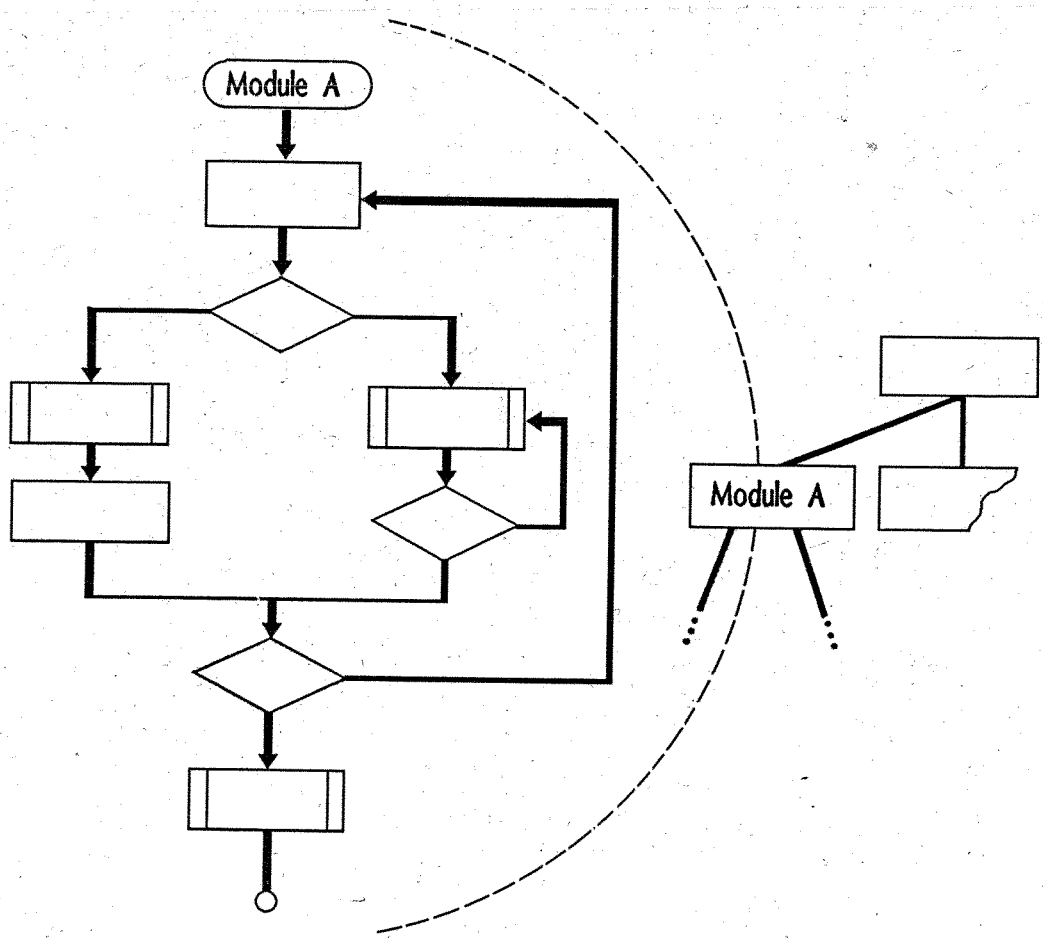
▷ 제어 결합(Control Coupling)

▷ 스탬프 결합(Stamp Coupling)

▷ 데이타 결합(Data Coupling)

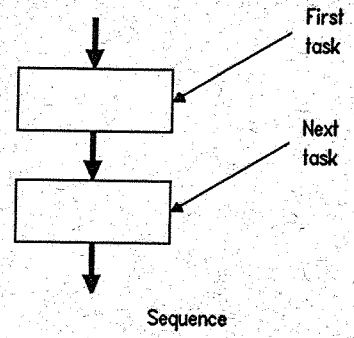
논리설계

▷ 각 모듈의 세부사항을 설계



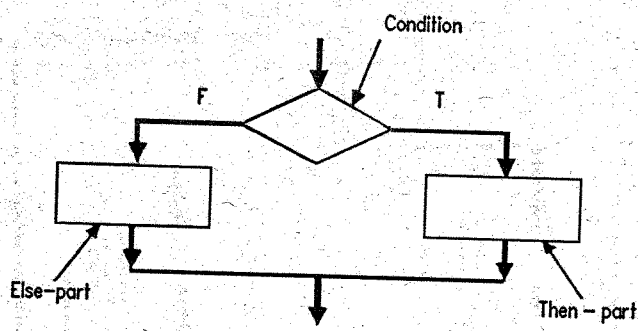
Flow Chart

▷ 순차



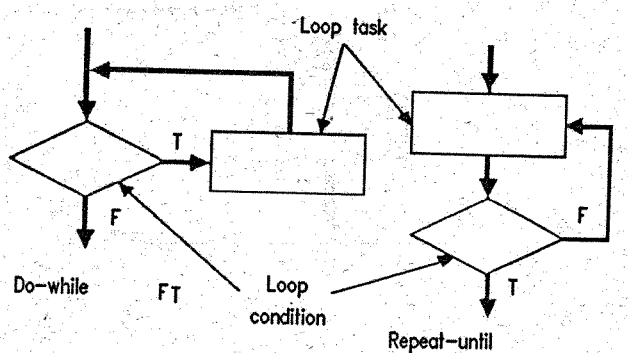
Sequence

▷ 택일



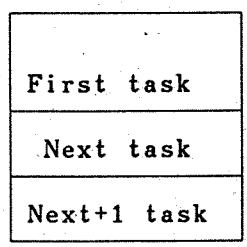
If-then-else

▷ 반복



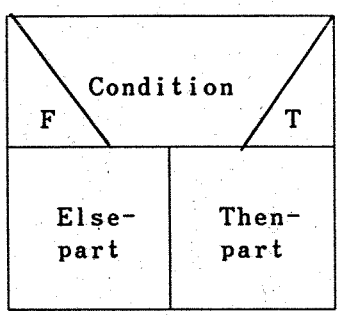
Box Diagram

▷ 순차

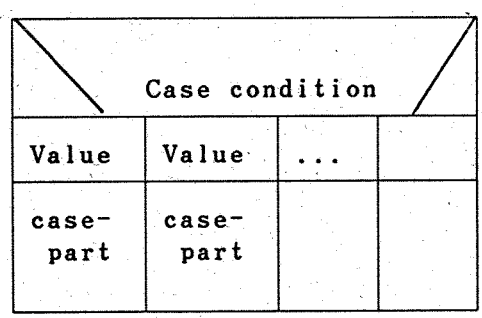


Sequence

▷ 택일

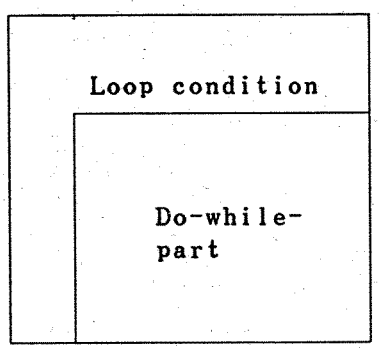


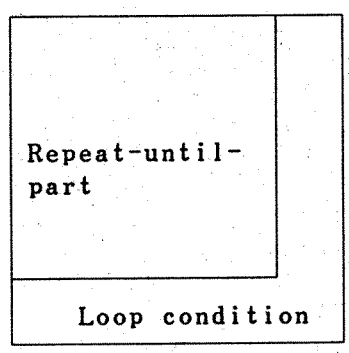
If-then-else



Selection

▷ 반복





Repetition

Decision Table

	조건들의 조합	
조건들의 집합		
작용들의 집합		

Program Design Language

▷ 기 정의된 구문구조와 논리를 기술하는 자연어로 구성

▷ 정의 되어야할 기본적인 PDL 구문구조

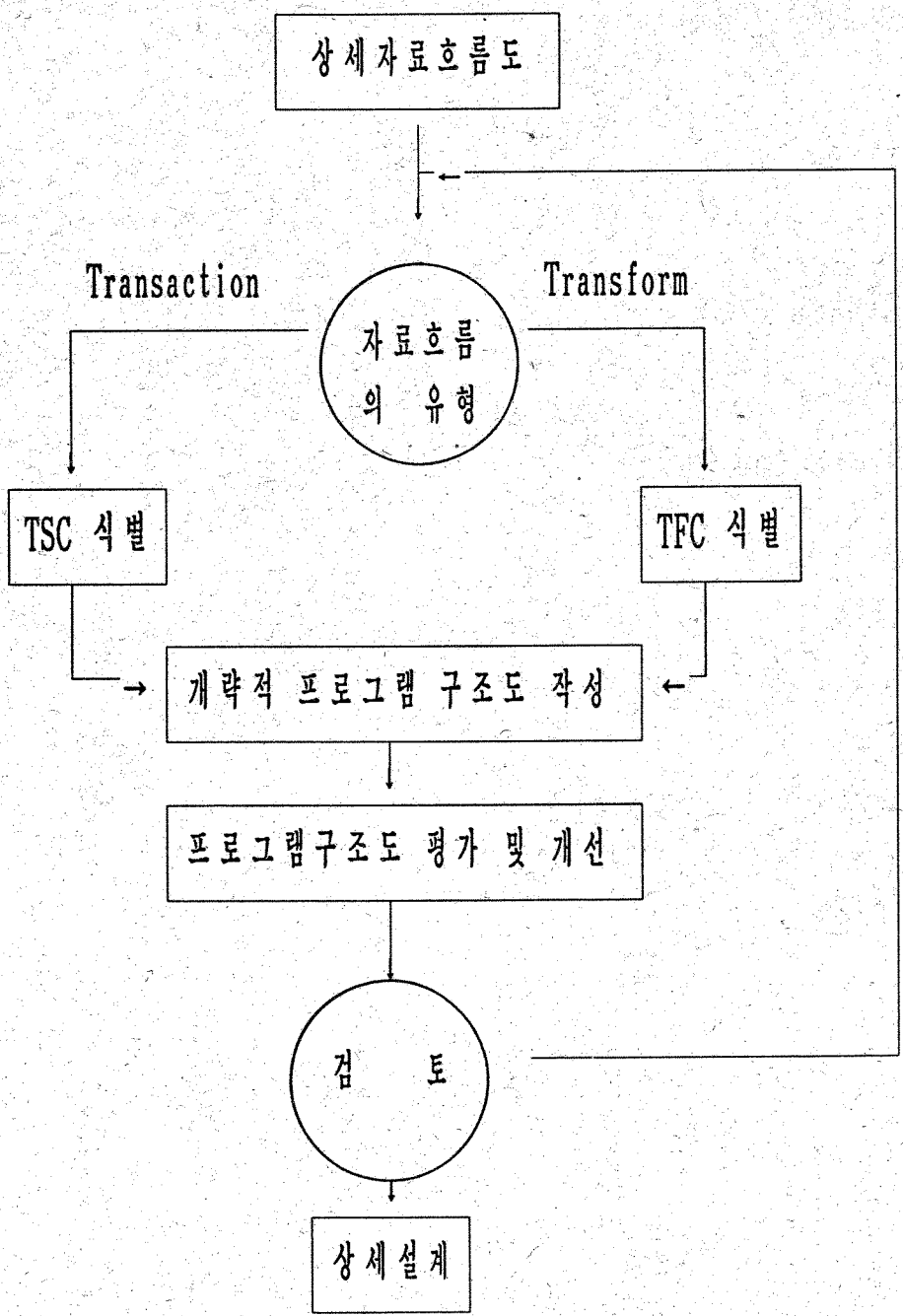
- 서브 프로그램
- 프로그램간의 인터페이스
- 데이타 선언 및 형
- 블럭 구성을 위한 구조
- 순차제어 구조
- 택일 제어 구조
- 반복 제어구조
- 입/출력 구조

논리설계 도구의 비교기준

- 모듈성(Modularity)
- 간편성(Simplicity)
- 수정용이성(Ease of Editing)
- 기계인식성(Machine Readability)
- 유지보수용이성(Maintainability)
- 구조성 강도(Structure Enforcement)
- 자동화 정도(Automatic Processing)
- 데이타의 표현(Data Representation)
- 논리 검증(Logic Verification)
- 코드로의 전환성(Code-to Ability)

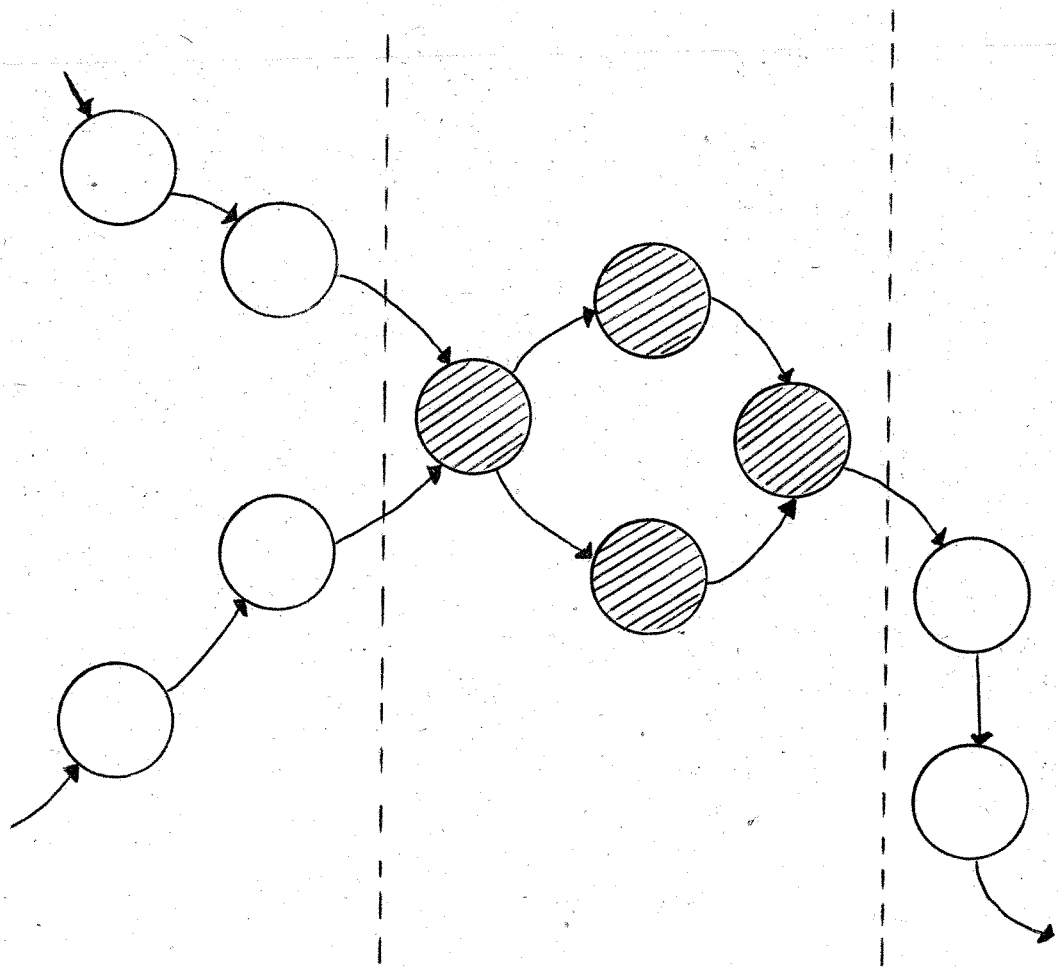
- Scalar Item
- Array
- Linked List
- Stack and Queue
- Tree
- Set
- Graph

데이타 흐름 중심 설계 절차



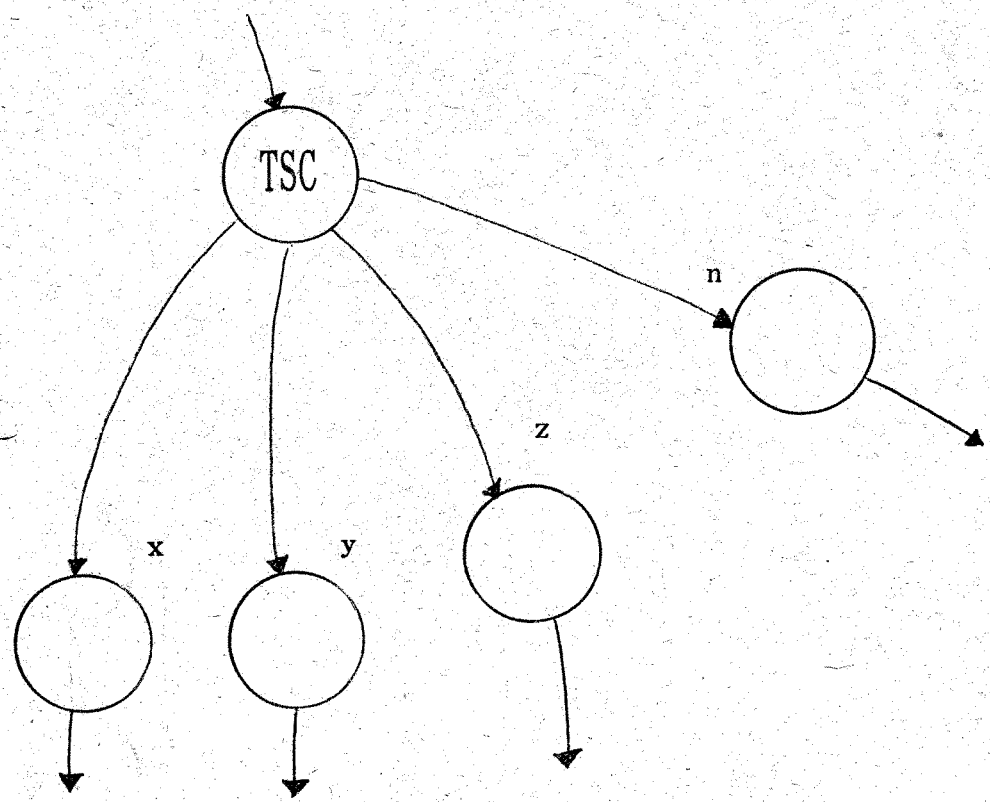
Transform Center

▷ 가장 내부적인 자료를 처리하는 상세 자료흐름도상의 프로세스 집합



Transaction Center

▷ 값에 따라 여러 다른 처리를 수행토록 야기하는 프로세스



개략적 프로그램 구조도 산출

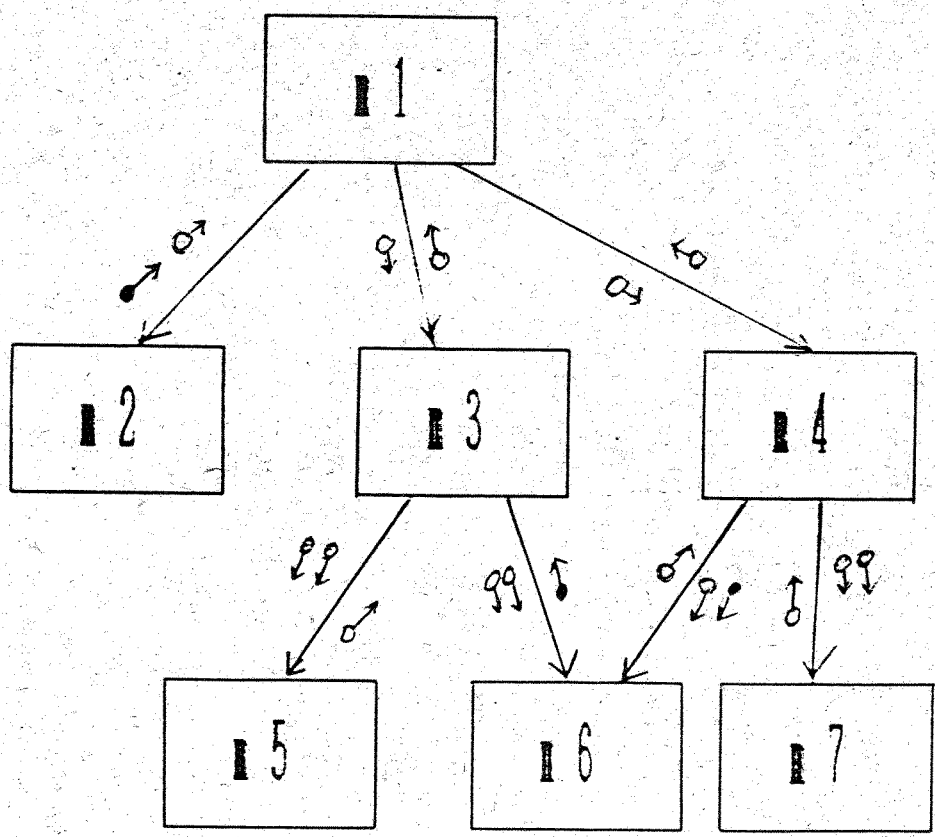
▷ Transform Center 의 적합한 한 프로세스를
상위 모듈로 간주하여 나머지를 하위 모듈로 연결

▷ 가상의 상위 모듈을 생성하여 Transform Center 를
중심으로 모든 프로세스를 가상의 상위 모듈에
하위 모듈로 연결

▷ Transaction Center 를 상위 모듈로 간주하여 나머지를
하위 모듈로 연결

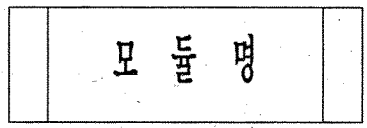
개략적 프로그램 구조도 산출

▷ 프로그램의 각 구성요소 간의 제어관계를 계층적으로 표현한 도표

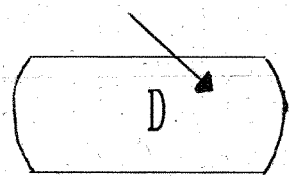


그 밖의 구조도 심볼

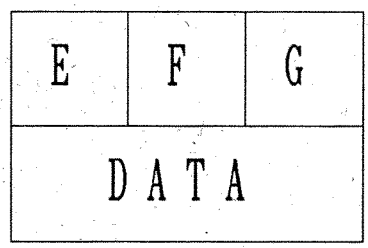
▷ Predefined Module



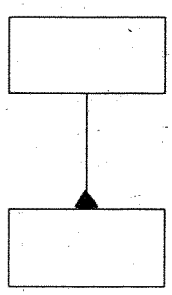
▷ Grobal Data Area



▷ Information Cluster

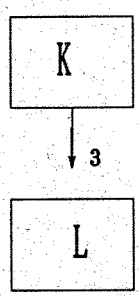


▷ Hat Symbol



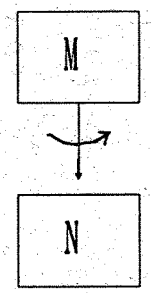
그 밖의 구조도 심볼

▷ Interface Table

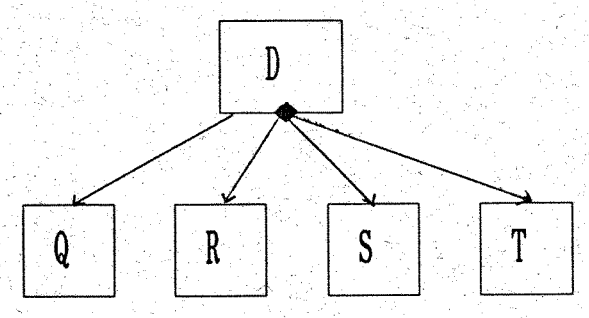


번 호	입 력	출 력
3	a , b	c

▷ 반복 호출



▷ Transaction Center



프로그램 구조도 평가및 개선지침

▷ 응집도는 강하고 결합도는 약하게 하라

▷ 모듈을 인수화하라

▷ 인식부분과 수행부분을 분리하지 말라

▷ 시스템의 형태를 균형있게 하라

▷ 에러를 발견하고 그 에러가 무엇인지 아는 모듈로
부터 에러메세지를 산출하라

▷ 자료는 간단한 사항부터 확인하라

▷ 가능한한 State Memory 를 회피하라

▷ 데이타구조와 일치하는 프로그램구조를 설계하라

▷ 공유결합과 스템프결합을 회피하기 위해
Information Cluster를 사용하라

▷ 너무 제한적이거나 일반적으로 설계하지 말아라
Fan-in을 높이고 Fan-out을 적절히 하라

데이타구조 중심 설계기법

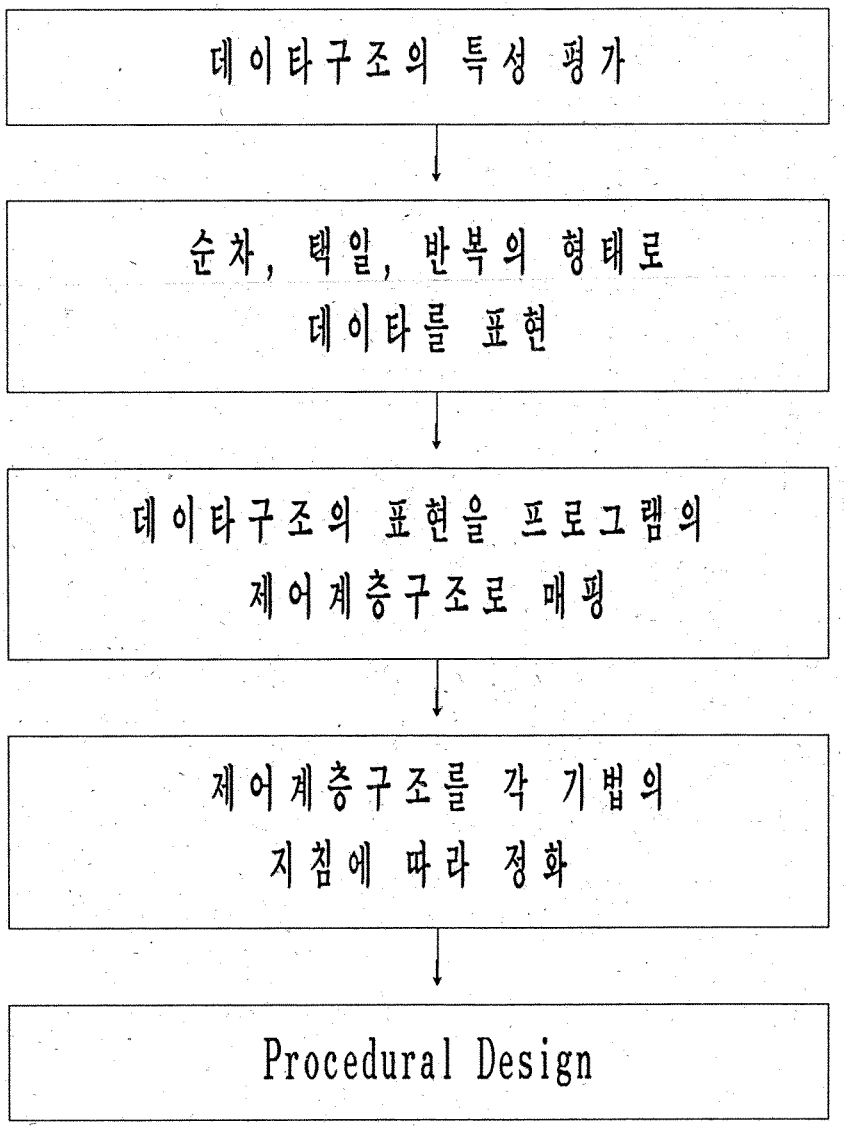
▷ Jackson System Development (J S D)

▷ Jackson System Programming (J S P)

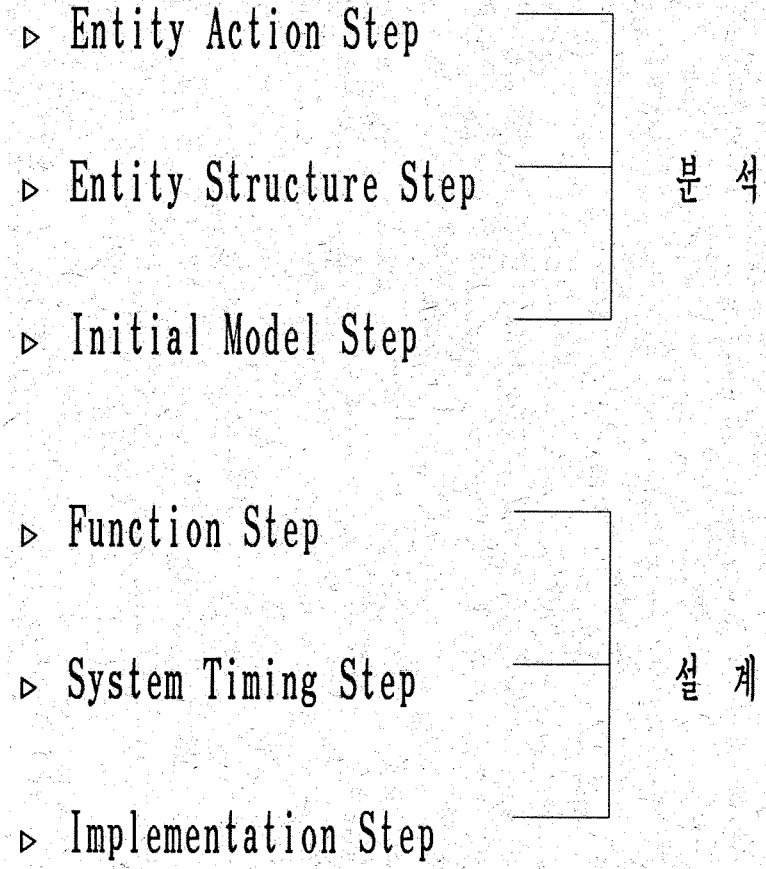
▷ Logical Construction of Program (L C P)

▷ Data Structured System Development (D S S D)

데이타구조 중심 설계절차



JSD의 설계 절차


▷ Entity Action Step
▷ Entity Structure Step
▷ Initial Model Step
분 석
▷ Function Step
▷ System Timing Step
▷ Implementation Step
설 계

Function Step

▷ 새로이 정의된 Function Process를 Data와 Vector Stream에 의해 Model Process와 연결함으로써 System Specification Diagram을 확장

- Embedded Function

  Model Process 구조문에 Operation을 추가

- Imposed Function

  Model Process의 State Vector나 Data Stream을 read하여 Output결과를 생성하는 Function Process를 추가

- Interactive Function

  Model Process의 State Vector나 Data Stream을 Read하여 Model Process와 관련된 Data Stream에 Write하고 Output결과를 생성하는 Function Process를 추가

System Timing Step

▷ Timing Constraints를 조사하여 Implementation Step에서 참고하도록 Informal Note로 문서화 하거나 SSD를 확장

▷ Timing Constraints에 관한 요구사항 예

- System Output의 Processing Time
- getsv Operation의 Frequency
- State Vector의 갱신주기
- Time Grain Marker가 읽혀져야할 시점
- Rough Merge의 동기화 문제

Implementation Step

▷ System Specification Diagram을

System Implementation Diagram으로 변환

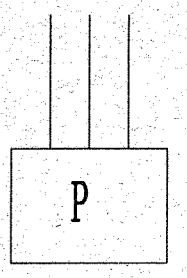
▷ 이 단계에서 고려해야할 사항

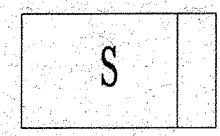
- 어떤 Input에 대해 만족할 만한 Response Time
- 많은 양의 Data를 가능한한 적은 저장장소에 기록
- 주기억 장치의 양을 최소
- 현존 소프트웨어를 가능한한 많이 사용

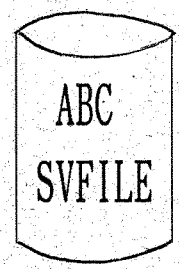
▷ 모든 System에 공통적인 고려사항

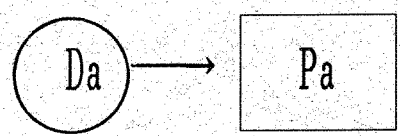
많은 Process들간에 Processor를 공유하도록 설계

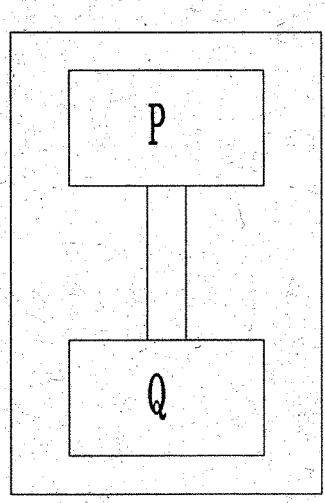
System Implementation Diagram

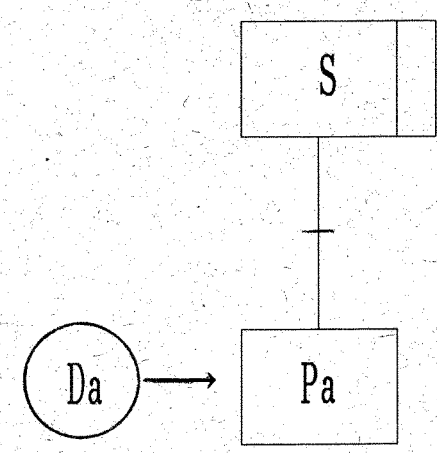




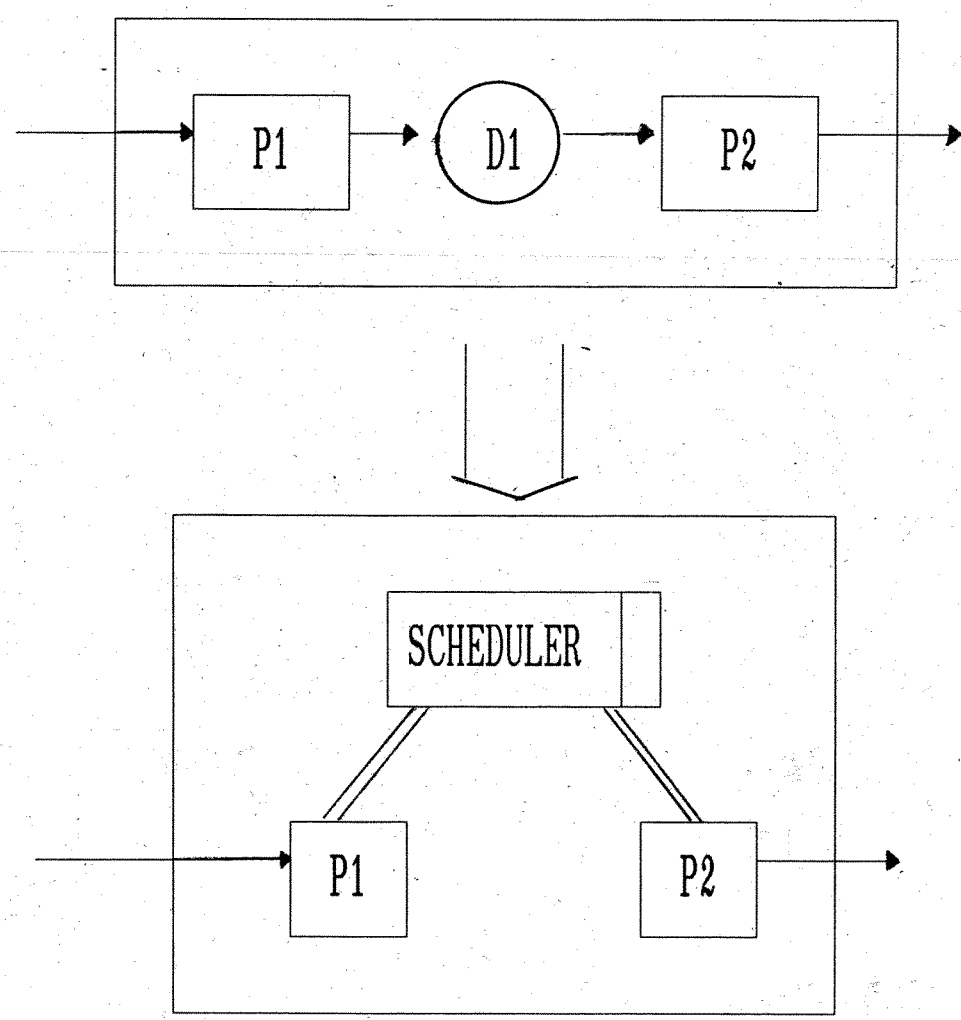




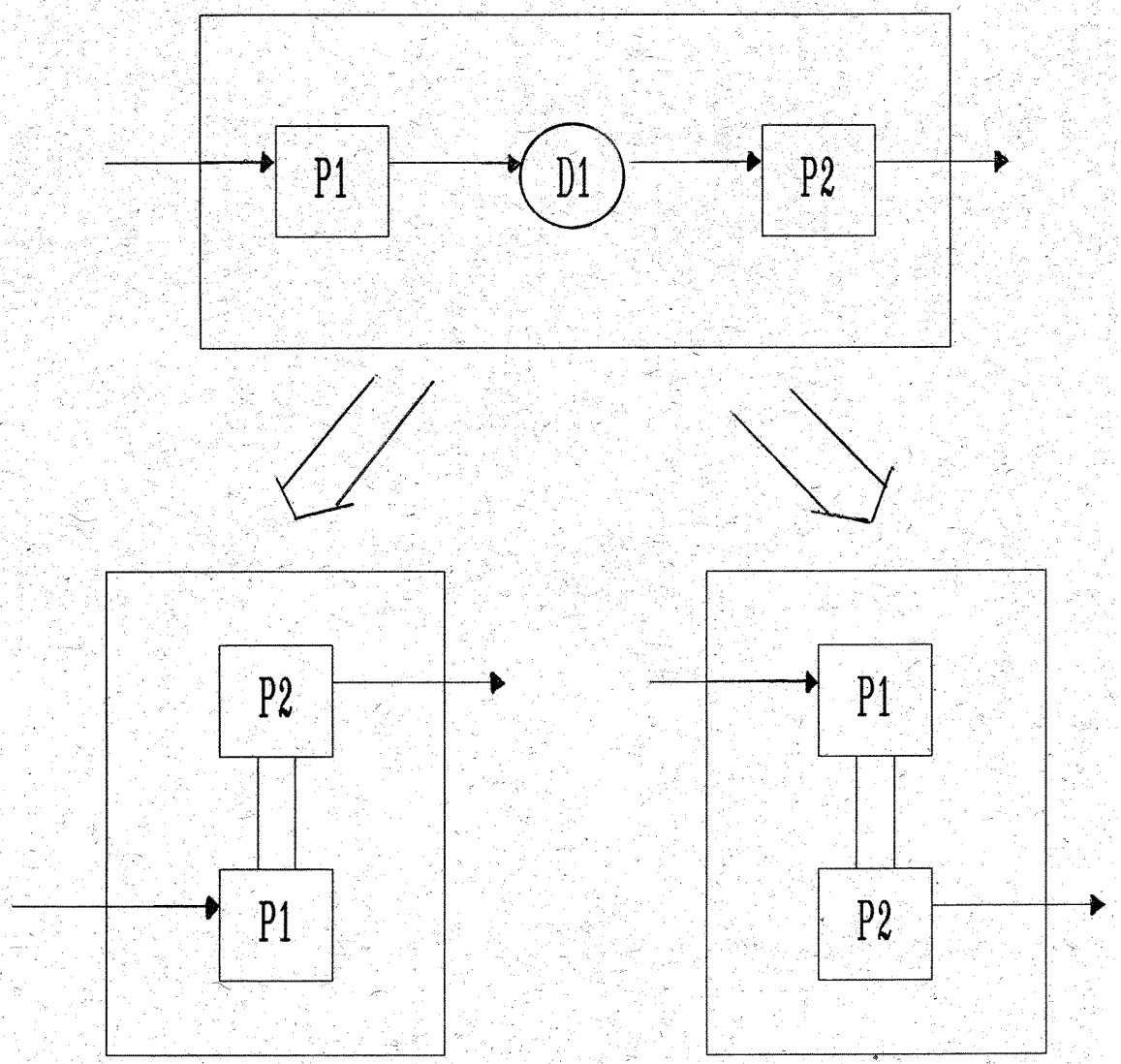




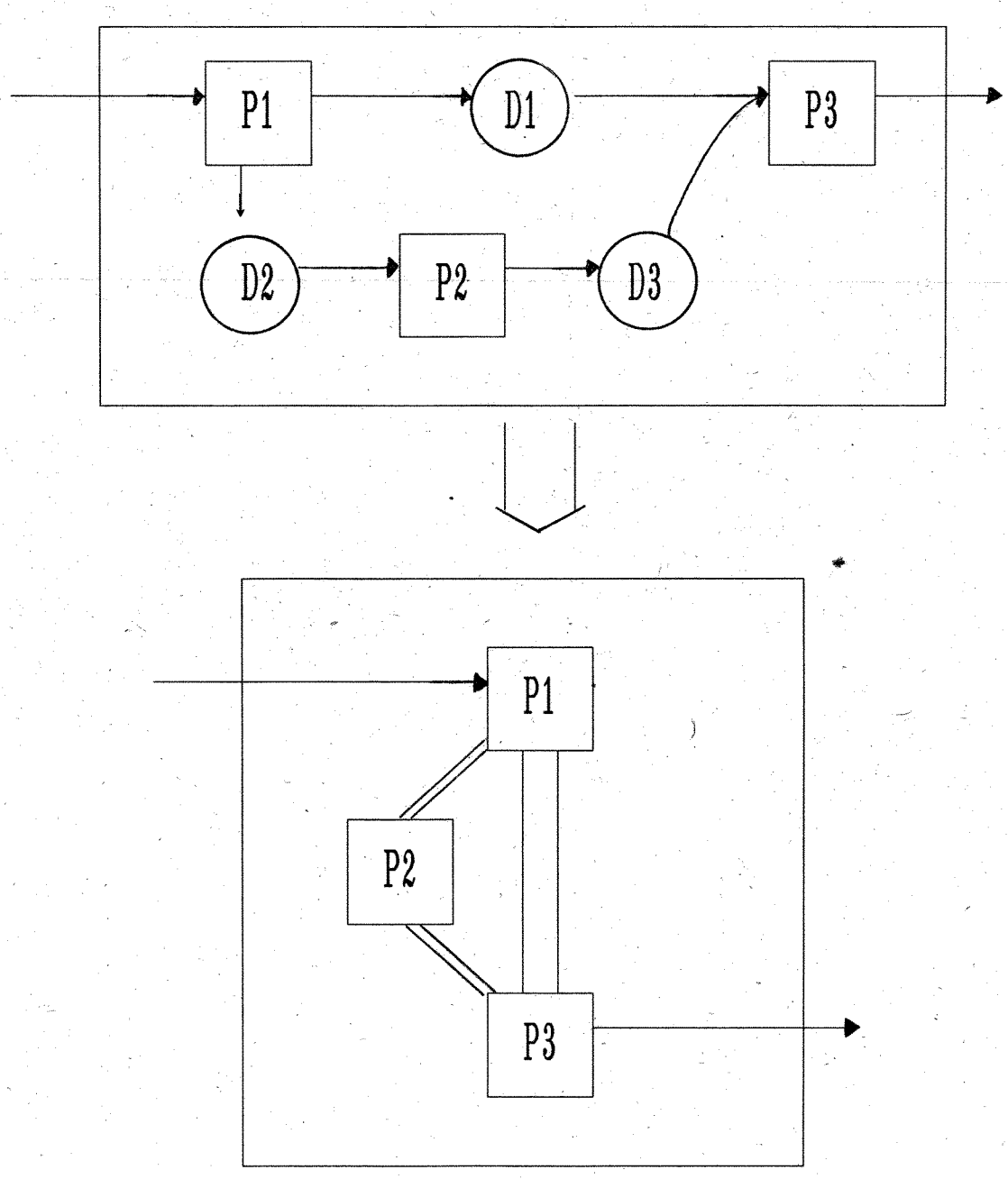
Scheduler에 의한 Scheduling



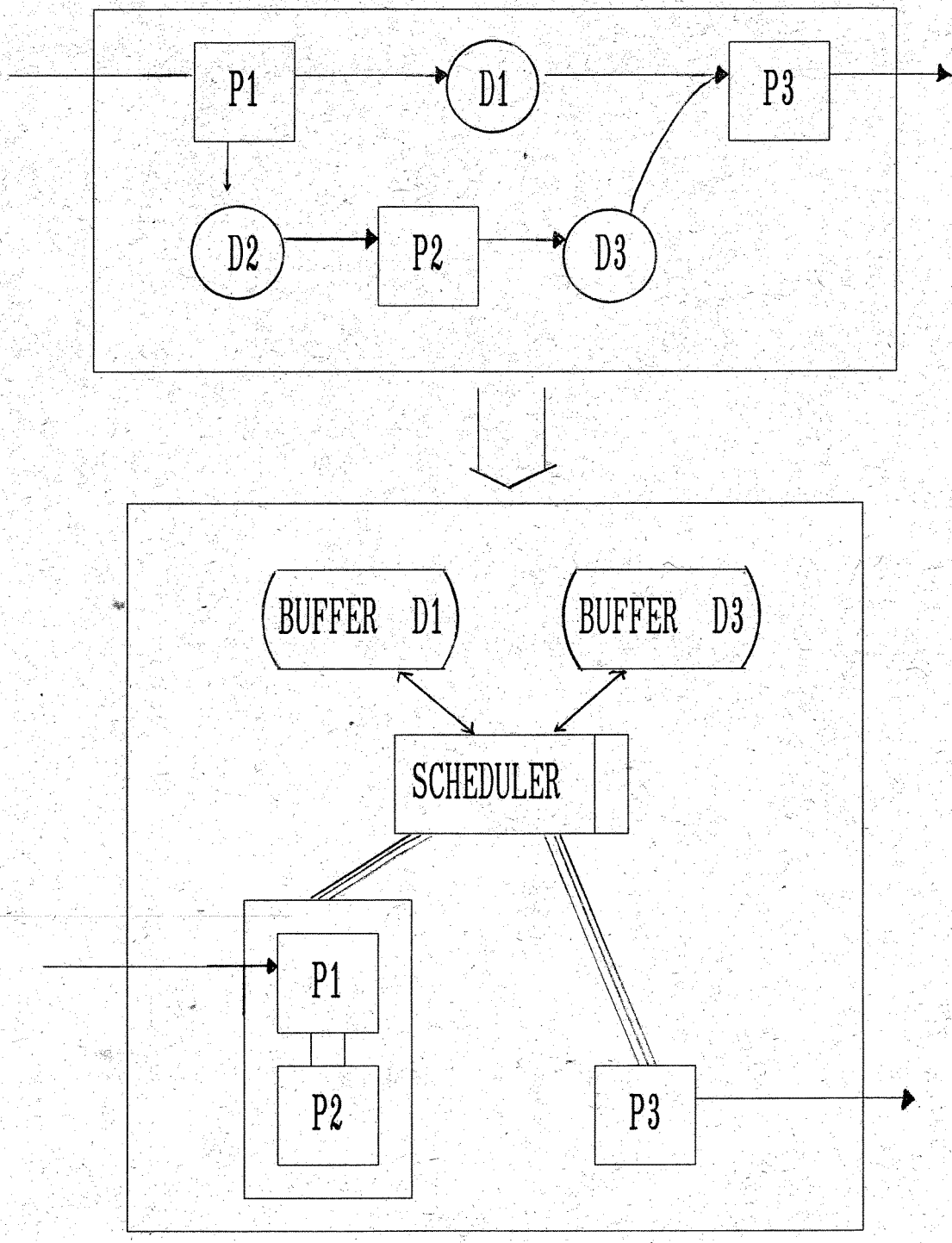
Scheduler를 사용치 않는 Scheduling



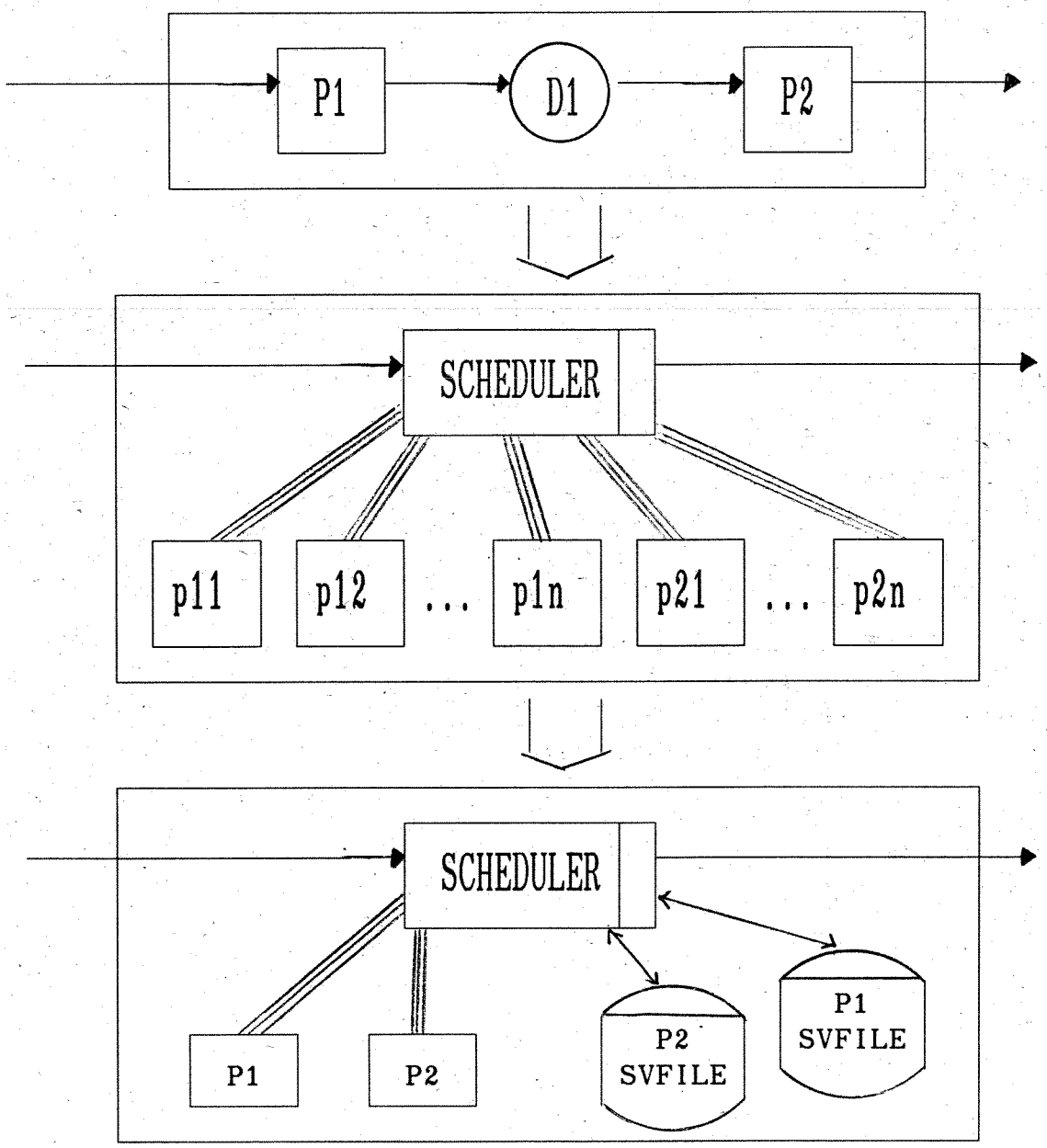
Rough Merge의 Inversion



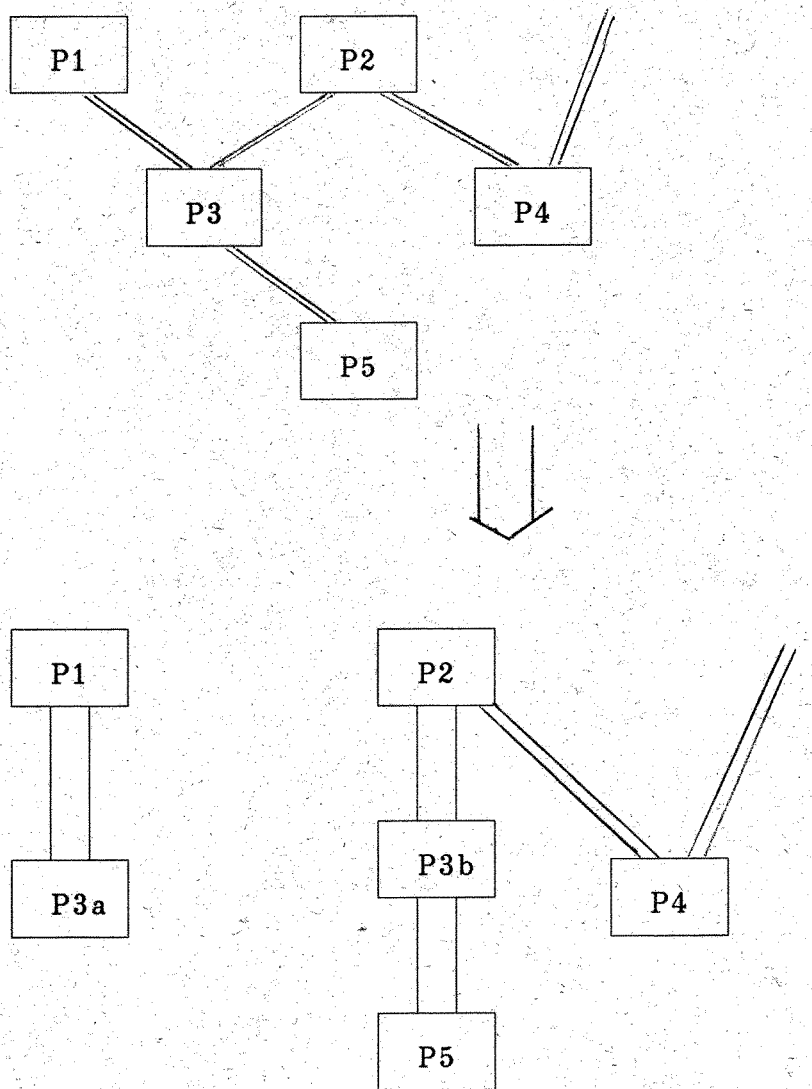
Buffer에 의한 Scheduling



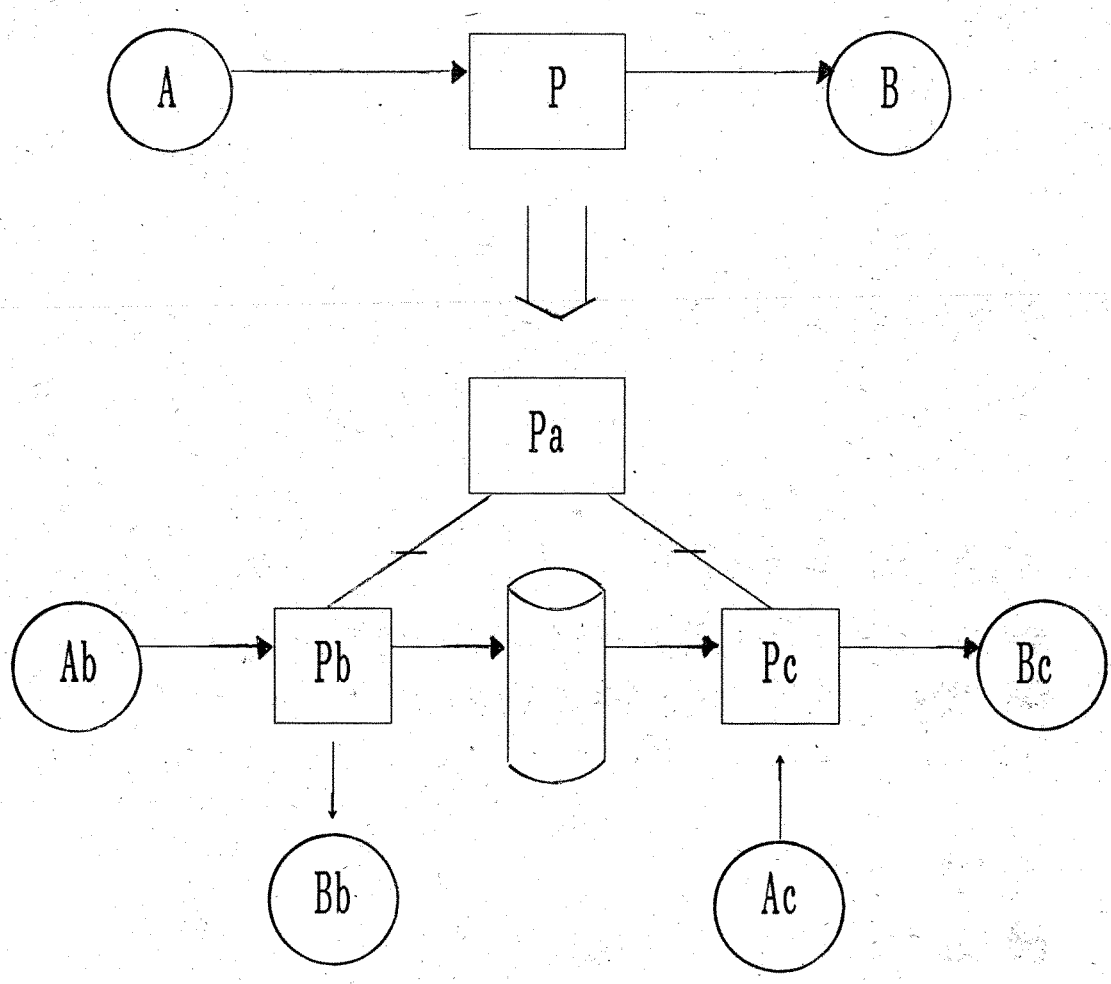
State Vector의 분리



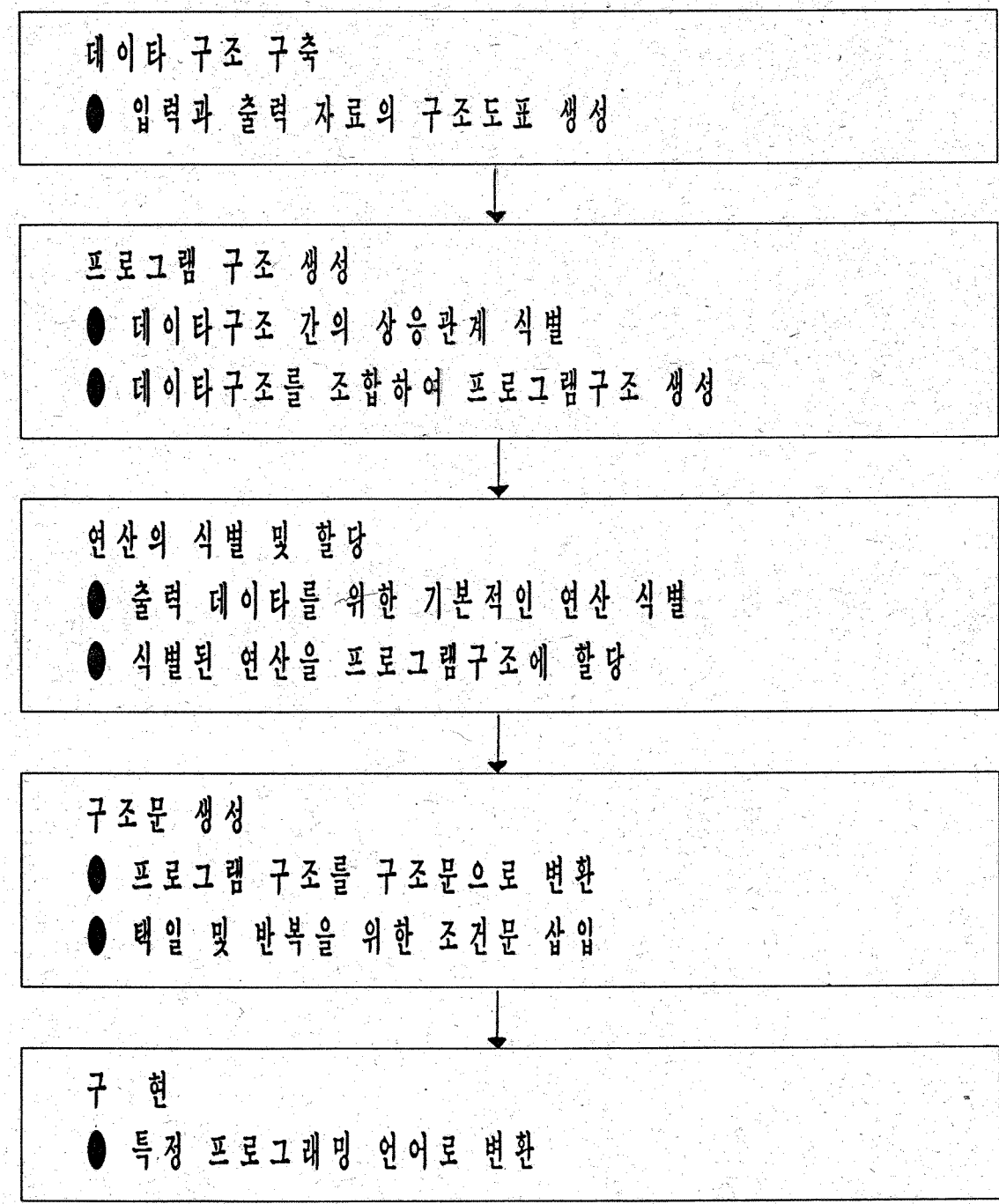
Process Dismembering



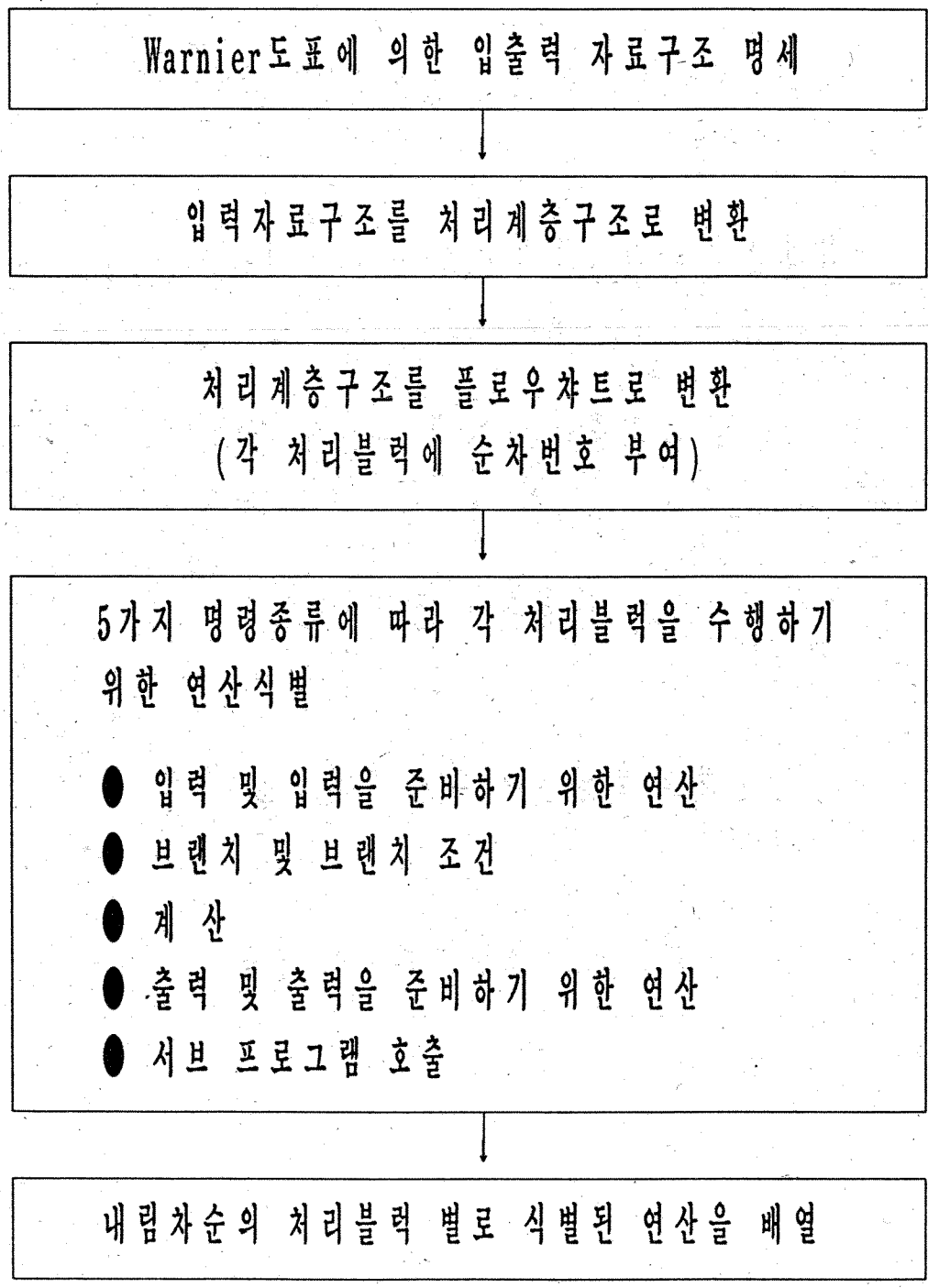
Process Dismembering



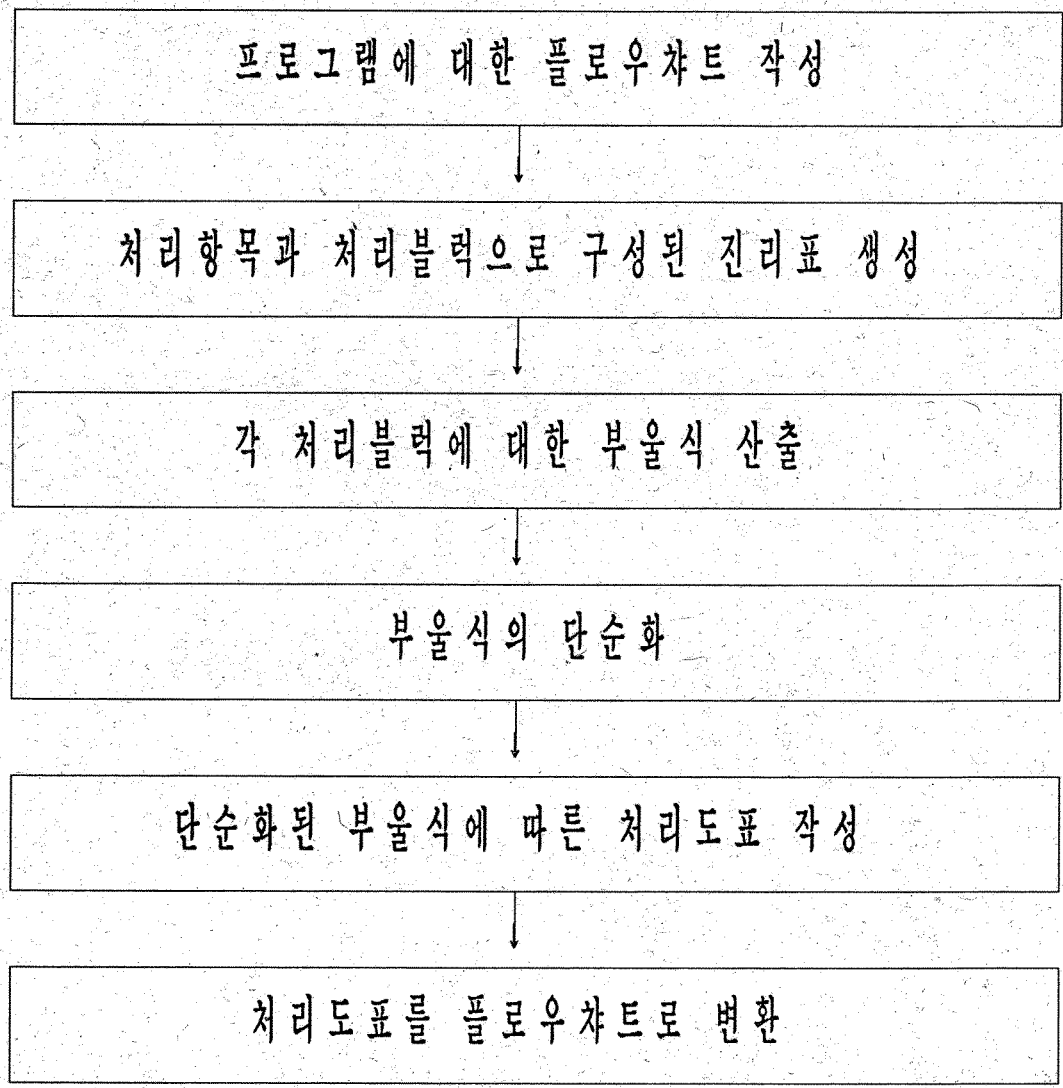
Jackson Structured Programming



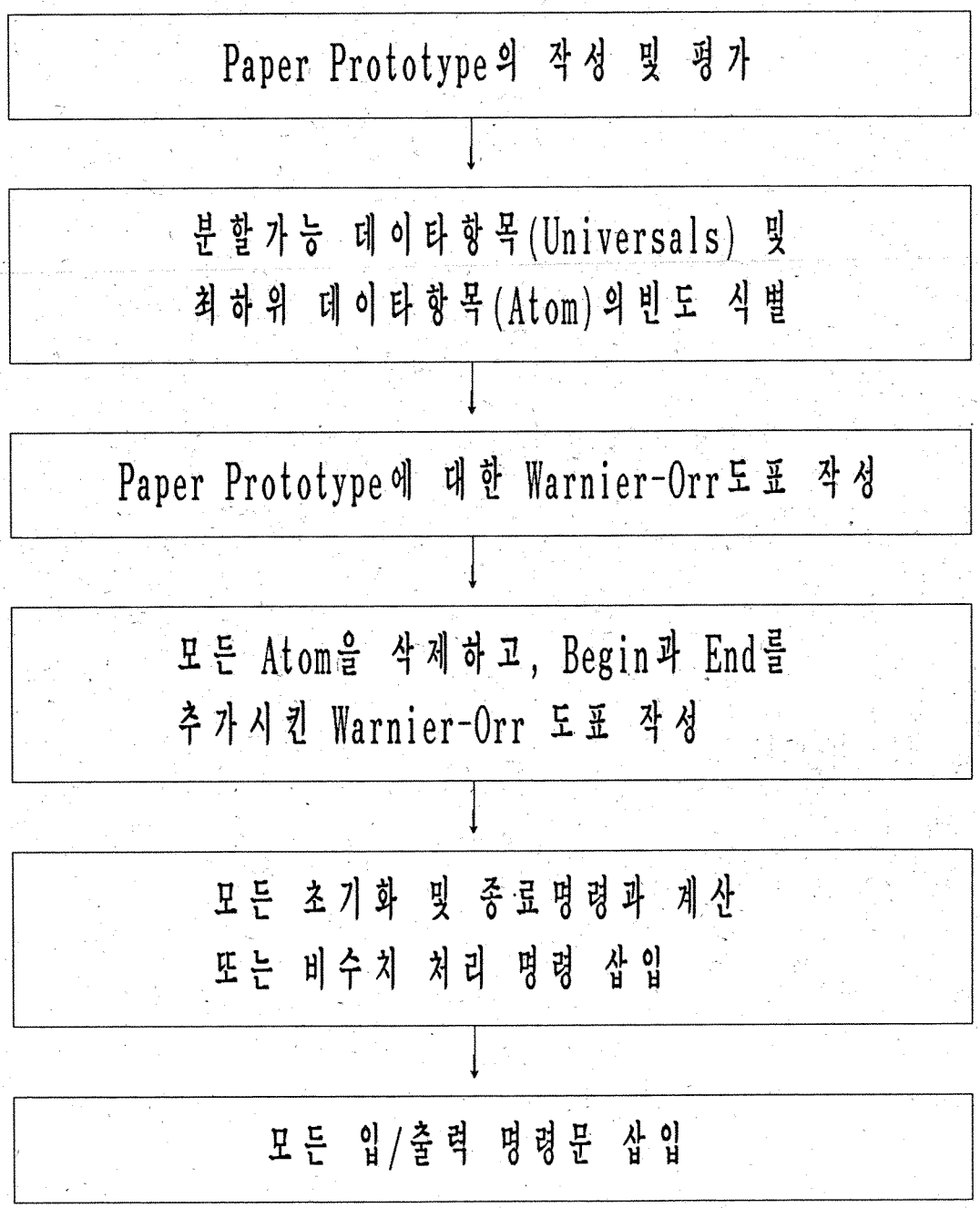
Logical Construction of Program



Logical Structure의 단순화



Data Structured System Development



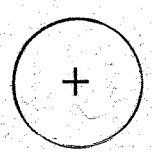
복잡한 처리를 위한 논리표현 방법

▷ 산술 및 논리연산을 위한 연산자의 표현

▷ 조건을 서술하기 위한 Footnote 심볼

?1 ?2 ?3 .....

▷ IF-THEN-ELSE 조건문

조건문과 

Structured Programming

▷ 타당한 시간내에 계산작업을 이해용이한 표현으로 기술하기 위해 사람의 사고를 조직화하는 행위

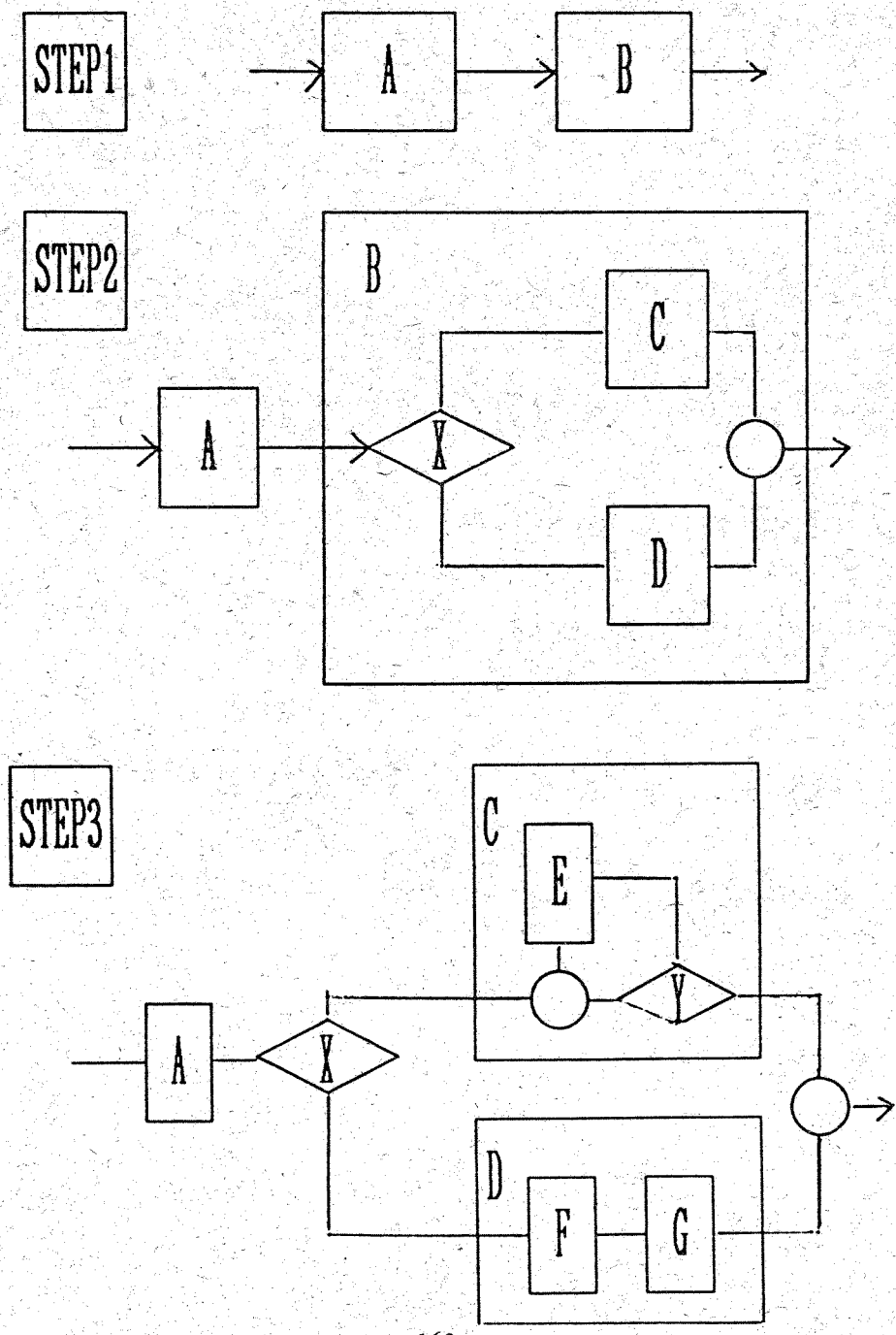
▷ Structured Programming Control Structure를 사용

- Sequence Structure
- Repetition Structure
- Selection Structure

▷ Program Segment가 작고 관리용이 하도록 유지

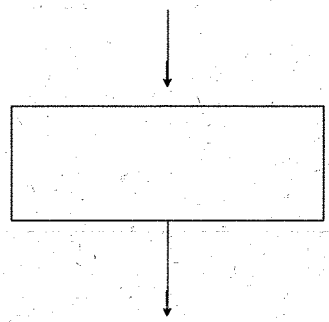
알고리즘을 계층적으로 구성

알고리즘의 계층적 구성

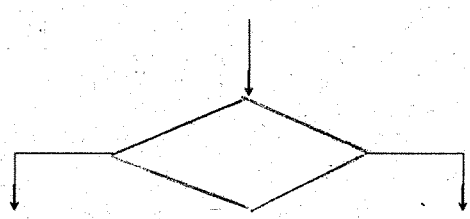


Specialized Flowchart Format

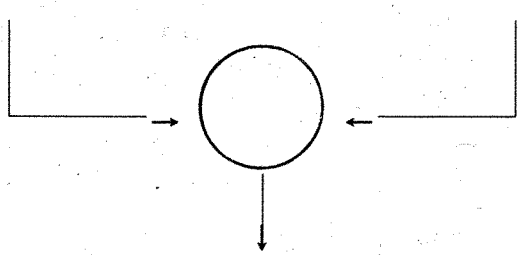
▷ Function Node



▷ Predicate Node



▷ Collecting Node

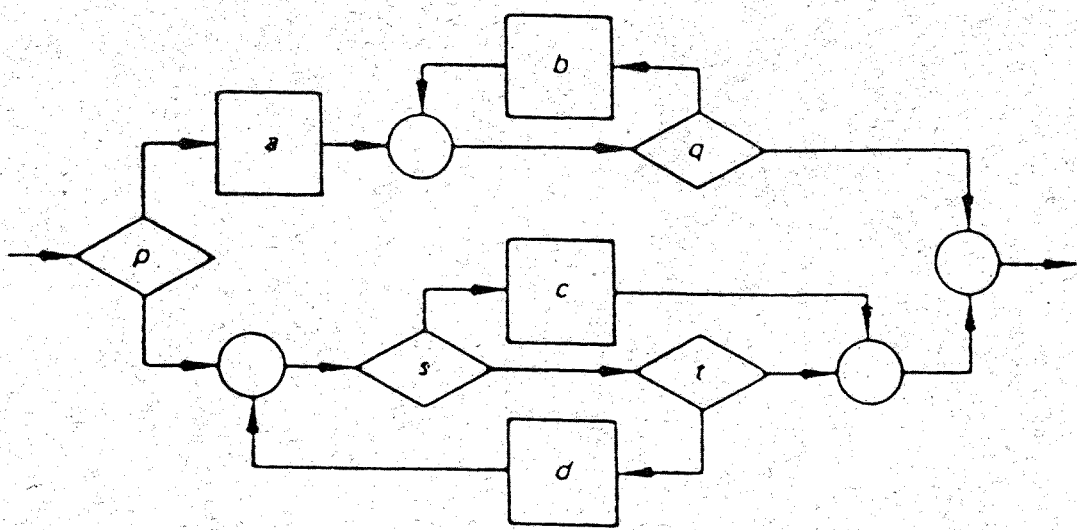


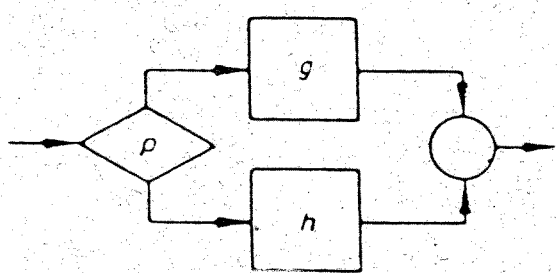
Proper Program

▷ 하나의 입구와 하나의 출구를 갖고

각 Node에 대해서 입구로 부터 그 Node를

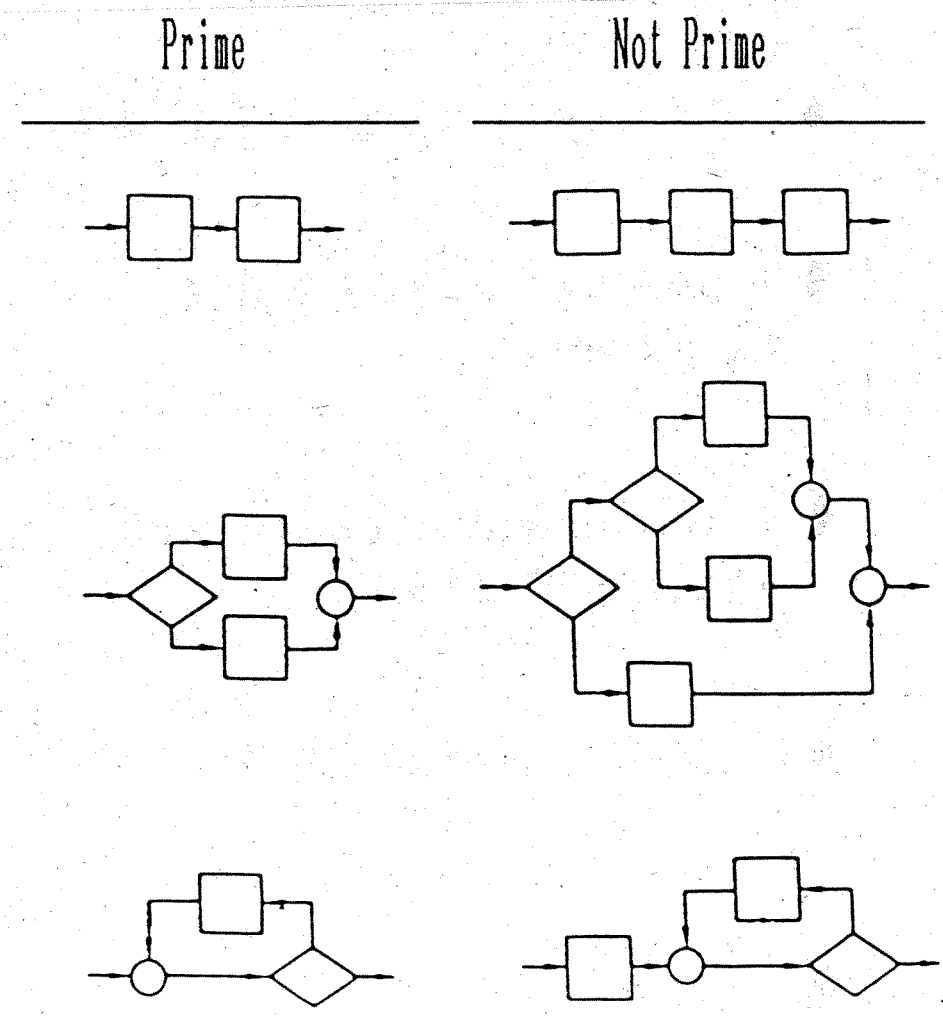
통하여 출구로 나가는 Path를 갖는 프로그램



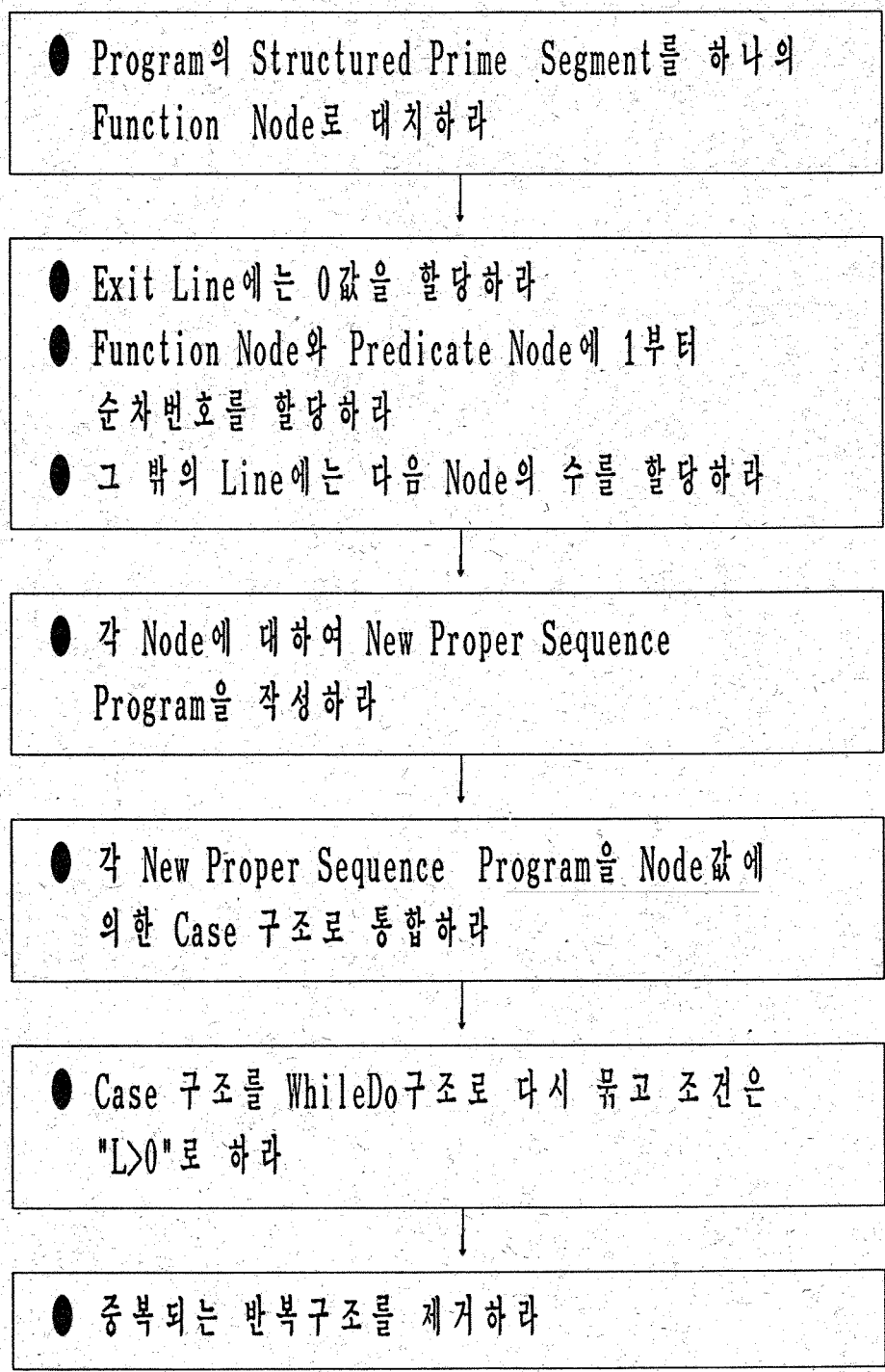


Prime Program

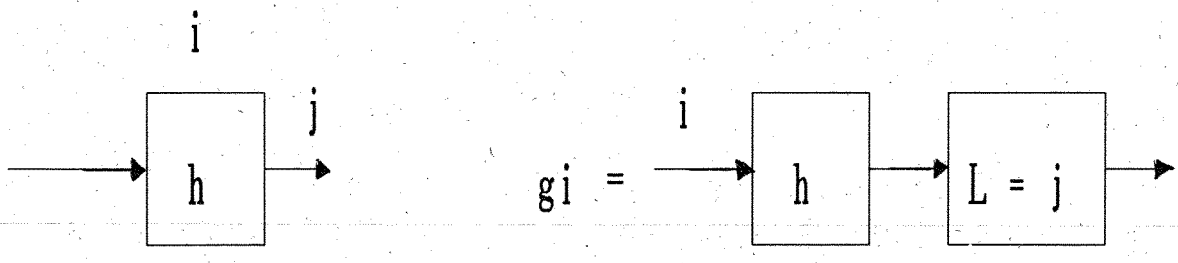
▷ 둘이상의 Node로 구성된 Proper Program으로서 Program의 어떠한 Segment도 Proper Program이 아닌 프로그램

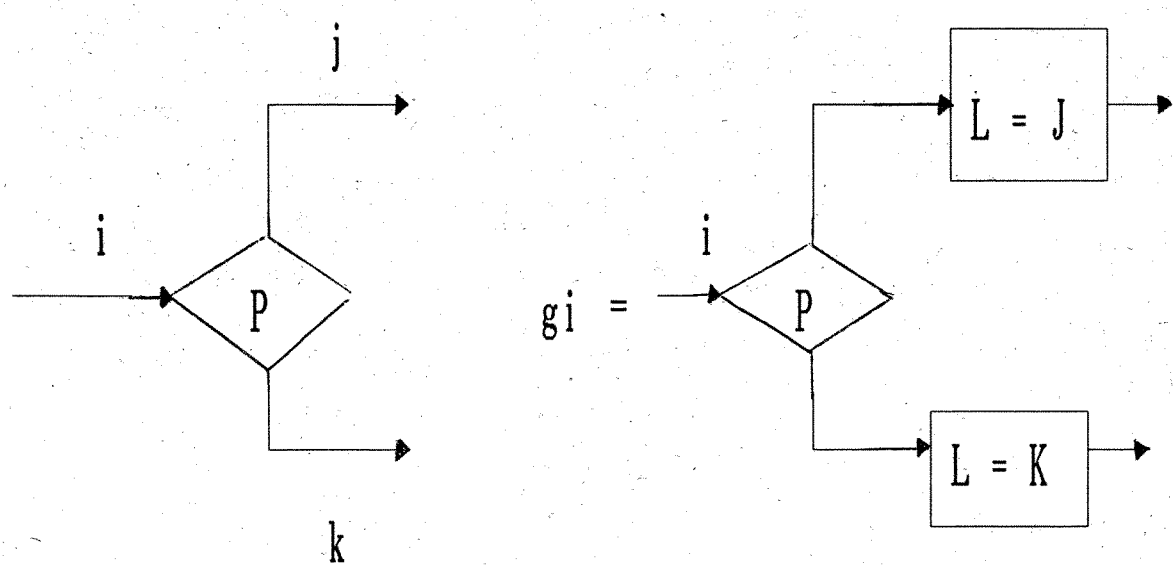


Structured Program으로의 변환



New Proper Sequence Program의 구축

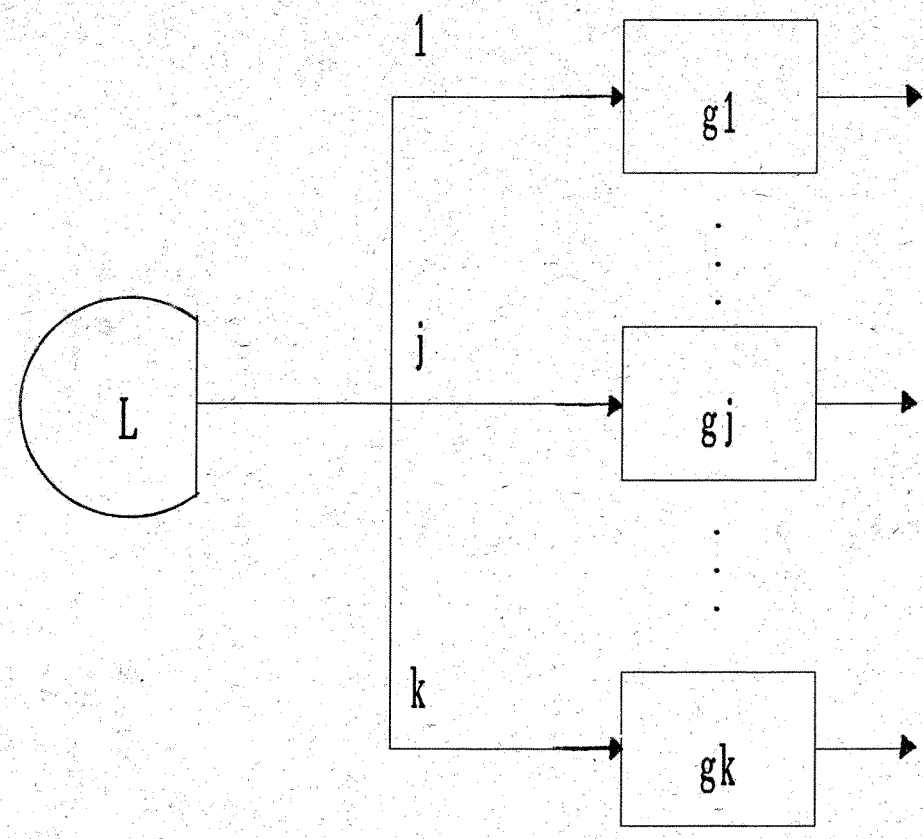




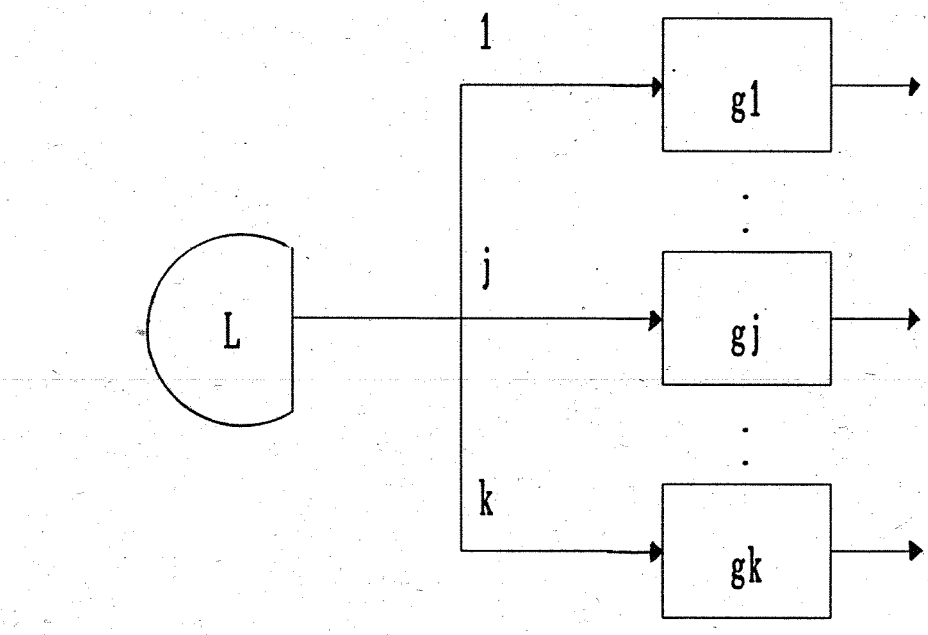
Case 구조의 구축

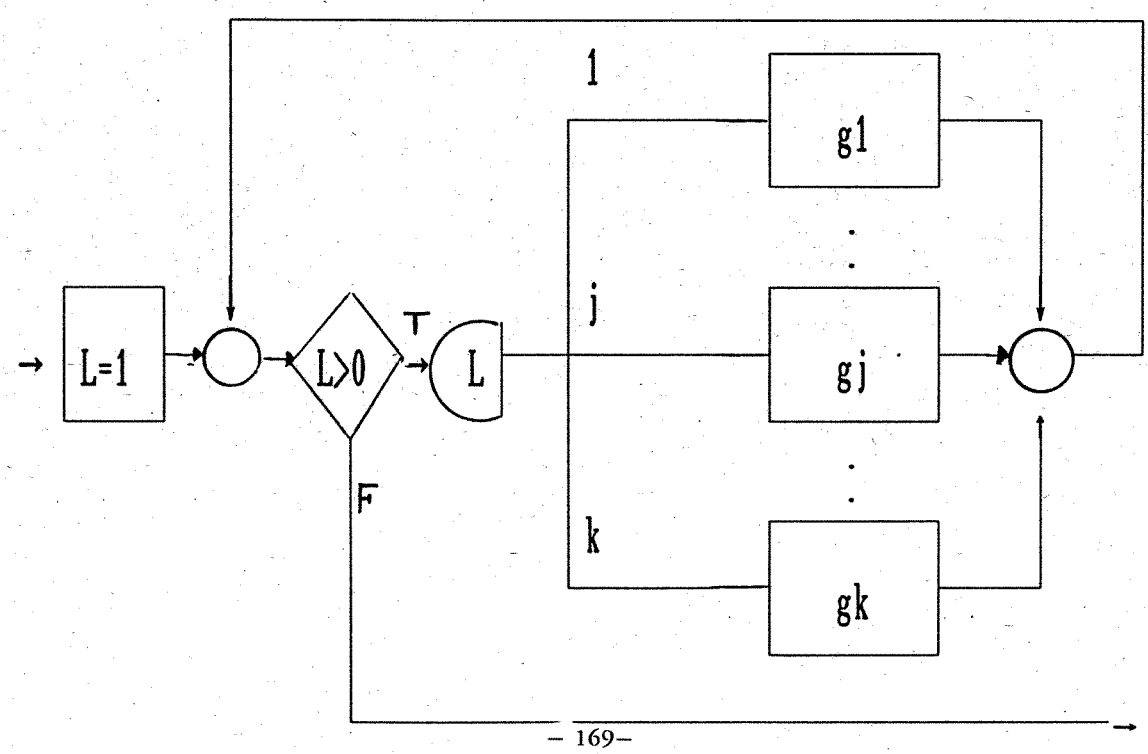
New Proper Program Sequence Program 들을

g1, ..., gj, ... , gk라 할때



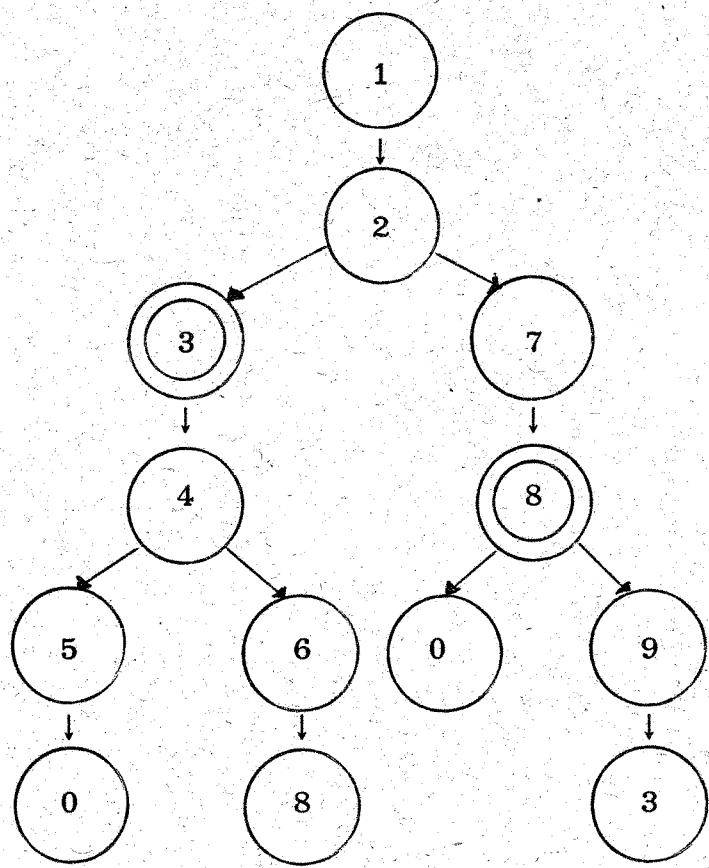
While-Do 구조의 구축





중복되는 반복구조의 제거

▷ L의 상태변이도를 작성하여 순환되는 값과 "1"만으로 분기되는 Case 구조로 변환



소프트웨어 공학에 미치는 언어의 특성

▷ 심리적 특성

- Uniformity(균일성)
- Ambiguity (모호성)
- Compactness(조밀성)
- Locality(국부성)
- Tradition(전통성)

▷ 공학적 특성

- 설계에서 코딩으로의 변환 용이성
- 컴파일러의 효율성
- 원시코드의 이식성
- 개발도구의 유용성
- 유지보수의 용이성

언어 선택시 고려사항

▷ 공학적, 심리적 특성

▷ 일반적인 응용 분야

▷ 알고리즘 및 계산상의 복잡성

▷ 소프트웨어가 수행될 환경

▷ 성능요구 사항

▷ 자료구조의 복잡성

▷ 소프트웨어 개발진의 지식

▷ 양질의 컴파일러 유용성

프로그래밍 언어의 발전

▷ 1950년 - 1960년대초

- 기계어 수준의 코딩
- 기계어, 어셈블리어

▷ 1950년대말 ~ 1960년대말

- High-Level Language의 출현
- Fortran, Cobol, Algol, Basic

▷ 1960년대중 ~

- 강력한 프로시듀어와 데이타 구조
- PL/1,Pascal,Modula-2,C,Ada
- Lisp, Prolog, Smalltalk,APL, Forth

▷ 1970년대말 ~

- Query Language
- Program Generator, 결심지원 언어, 4GL

▷ Expression (표현식)

▷ Control Structure (제어구조)

▷ Input / Output (입출력)

▷ Documentation (문서화)

Expression

▷ 의미하는 것을 간단히, 그리고 직접 코딩하라

▷ 라이브러리 Function 을 사용하라

▷ Temporary Variable을 회피하라

▷ 명확하게 코딩하라

▷ 반복되는 표현식은 Common Function으로 대치하라

▷ 이해도를 증가하기 위한 심볼들을 사용하라

▷ 불필요한 분기는 삼가하라

▷ 언어의 좋은 특성은 최대한 이용하라

▷ 논리 표현식을 Conditional Branch로 해결하지 말아라

Control Structure

- DO-END의 Block Structure를 사용하라
- Indentation을 수행하라
- IF-THEN-ELSE구조를 사용하라
- DO-UNTIL 또는 WHILE-DO 구조를 사용하라
- 프로그램이 하향식으로 읽혀질 수 있도록 기술하라
- 조건문과 관련된 기능은 가능한 조건문 바로 다음에 기술하라.
- 부정적인 조건문은 삼가하라
- 간단한 제어구조를 유지하라

Input/Output

▷ 입력의 타당성을 테스트하라

▷ 입력자료의 끝을 나타내는 식별자를 사용하라

▷ 입력자료를 준비하기 용이하도록 하라

▷ 출력은 이해용이 하도록 하라

▷ 입력자료의 형태는 균일 하도록 유지하라

Documentation

▷ 코드와 설명문은 일치하도록 하라

▷ 설명문은 코드를 반복 설명하지 말아라

▷ 의미있는 변수명을 사용하라

▷ 의미있는 레이블명을 사용하라

▷ 쉽게 이해할수 있도록 프로그램을 편집하라

▷ 복잡한 데이타에 대해서도 설명을 하라

▷ Over-Comment하지 말아라

▷ 데이타는 표준화된 순서에 의해 선언하라

프로그래밍 언어와 코딩

Programming Language and Coding

프로그래밍 언어의 기초개념

- Type Checking (형검사)
- Separate Compilation (분리가능 컴파일)
- User Defined Data Type (사용자정의 데이타형)
- Generic Facility (포괄적 프로그램 단위)
- Scope Rule
- Exception Handling (예외처리)
- Concurrency Mechanism (병렬 메카니즘)

Type Checking

▷ Typeless

선언문이 존재치 않음

▷ Automatic Type Conversion

다른 데이타형 간의 연산시 자동적으로 형 변환

▷ Mixed Mode

유사한 데이타형간에만 형 변환

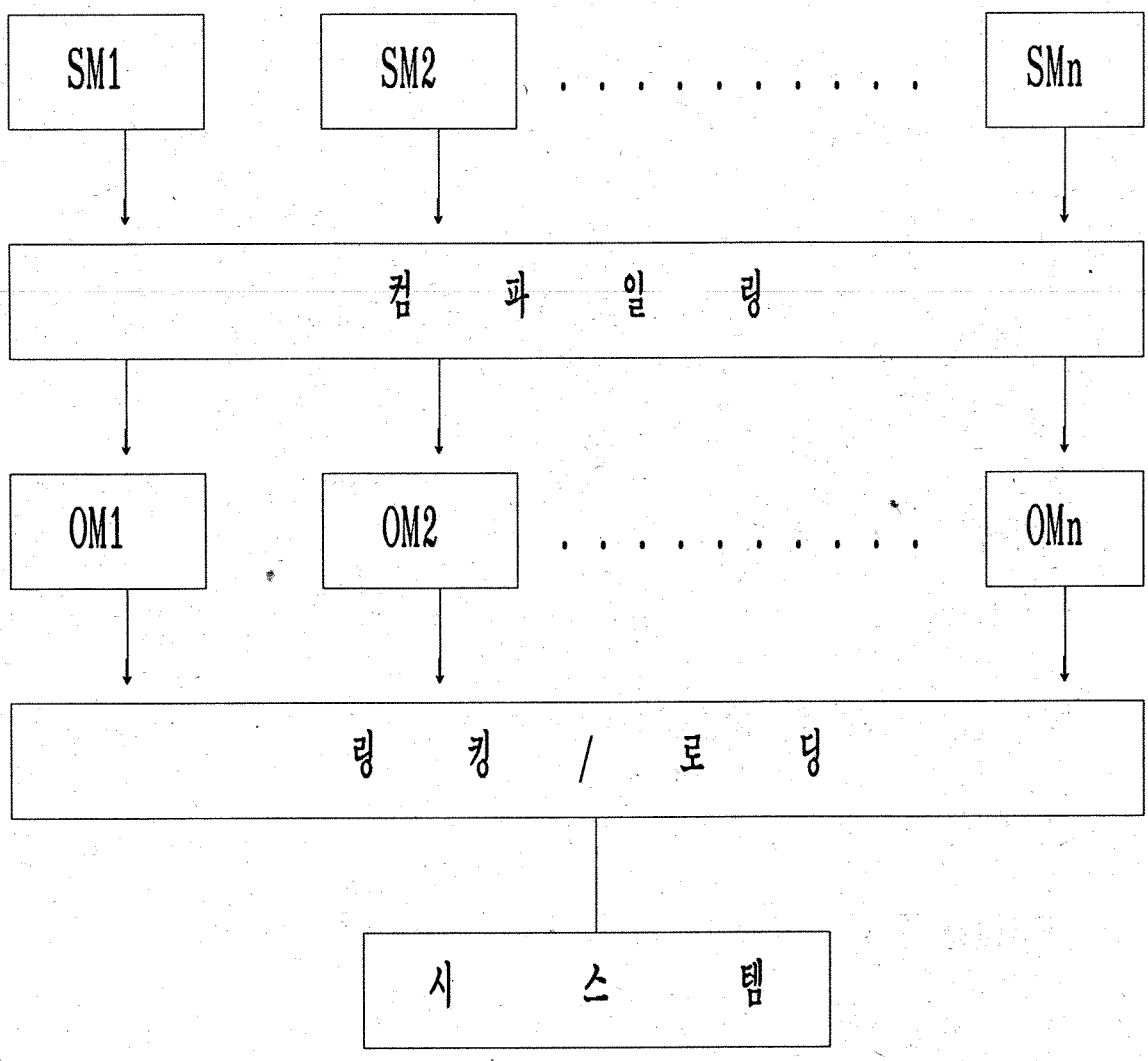
▷ Strong Type Checking

동등한 데이타형 간에만 연산이 가능

▷ Psedo-Strong Type Checking

로드 및 수행시 형검사를 않함

Separate Compilation



User Defined Data Type

▷ SubType

기 정의된 데이타형이 가질수 있는 값의 집합 내에서

정의된 데이타형

▷ Derived Type

기정의된 데이타형이 가질수 있는 값의 집합 내에서

새롭게 정의된 데이타형

▷ Enumeration Type

데이타형이 가질수 있는 값의 집합을 나열하여 정의하는 데이타형

▷ Variant/Disciminant Record Type

구조가 가변적인 레코드를 정의하는 데이타형

▷ Pointer Type

다른 데이타형을 엑세스하는 데이타형

▷ Abstract Data Type

데이타와 이에 적용될수 있는 연산까지 정의하는 데이타형

Generic Facility

▷ 프로그램 내의 특정 구성요소를 매개변수화 하여 포괄적 프로그램을 작성하고 매개변수화된 구성요소에 대해 변경사항 만을 선언해 줌으로써 새로운 프로그램을 작성토록 하는 수단

▷ Reusability(재사용성)를 직접적으로 재고하는 개념

Scope Rule

▷ Global Scope

모든 변수가 모든 모듈에서 공용

▷ Fortran Scope

COMMON문에서 선언된 변수외에는 선언된 모듈 내에서만 공용

▷ Nested Scope

중첩된 프로그램에서 상위 모듈에 선언된 변수는 해당 모듈 및 하위 모듈에서 공용

Exception Handling

▷ Resumption Model

- Exception(예외사항)이 발생하면 Exception에 대한 적합한 처리를 수행하고 Exception발생 지점 이후부터 끝까지 수행한 후 호출한 상위 모듈로 리턴
- PL/I

▷ Termination Model

- Exception이 발생하면 Exception에 대한 적합한 처리를 수행한 후 바로 호출한 상위 모듈로 리턴
- Ada

Concurrency Control

▷ 자원의 공유

- Mutual Exclusion(상호배반)을 해결하기 위한 메카니즘

☞ Semaphore,Monitor

▷ 메세지 전달

- Synchronization(동기화)를 해결하기 위한 메카니즘

☞ Asynchronous Message Passing

Synchronous Message Passing

- Symetric Message Passing

- Asymetric Message Passing

소프트웨어 품질보증

Software Quality Assurance

소프트웨어 품질보증

Software Quality Assurance

소프트웨어 품질보증

▷ 소프트웨어의 품질

명확하게 기술된 기능및 성능 요구사항,
문서화된 개발표준안, 그리고
모든 소프트웨어에 기대하는 내재적인 특성에
대한 일치도

▷ 소프트웨어 품질보증

소프트웨어 품질이 적합한 신뢰구간을 유지하도록
수행하는 모든 활동의 집합

소프트웨어 품질보증을 위한 활동

- 기술적방법 및 도구의 적용
- Formal Technical Review(공식적 기술검토) 수행
- 소프트웨어 테스트 수행
- 표준화 유지
- Change Control(변경통제)
- 소프트웨어 품질 측정
- 품질보증 활동 결과 기록 및 보고

Formal Technical Review

▷ Formal Technical Review

공식적인 절차에 의해 개발 기간중 산출되는 산출물의 결점을 발견하기 위한 활동

▷ 목 적

- ☞ 기능,논리,구현 상의 가능한 에러를 조기에 발견
- ☞ 소프트웨어가 표준안에 따라 구축되도록 보장
- ☞ 관리 용이한 사업을 수행

Formal Technical Review의 중요성

▷ 개발 기간중 발견되는 총 에러의 50-60%가 코딩단계 이전에서 발견

▷ 후기 단계에서 발견된 에러를 수정하는 비용이 초기단계에서 발견된 에러를 수정하는 비용보다 상대적으로 높음

☞ 설계시 발견된 에러를 수정하는 비용이 "1"이라면 코딩시 발견된 에러를 수정하는 비용은 "6.5", 테스트시 발견된 에러를 수정하는 비용은 "15", 인도후 발견된 에러를 수정하는 비용은 "67" 정도의 비용이 요구됨

Formal Technical Review를 위한 회의

▷ 구성 인원

-검토회 리더, 발표자, 3-5명의 검토자

▷ 진행절차

-생산자가 프로젝트 리더에게 회의 소집을 요구

-프로젝트 리더는 검토회 리더와 접촉

-생산자는 검토할 산출물을 검토자에게 회의 소집전 배부

-검토리더는 회의시 일정을 발표하고 발표자는 간단한 브리핑 수행

-문제점 제기

-문제점 및 발견된 에러를 기록

-최종 결정(수락,조건부수락,취소)

-참석자 서명

검토사항 기록및 보고

▷ Review Issue List 작성

-검토 대상물 내에 존재하는 문제점 기술

-산출자가 수정시 참조

▷ Review Summary Report 작성

-무엇을 검토했는가

-누가 검토했는가

-결론은 무엇인가

검토시 가이드 라인

▷ 산출자를 검토하지 말고 산출물을 검토하라

▷ 일정을 세우고 그것을 유지하라

▷ 논쟁과 반박을 삼가하라

▷ 검토의 목적은 문제의 발견 임을 잊지 말아라

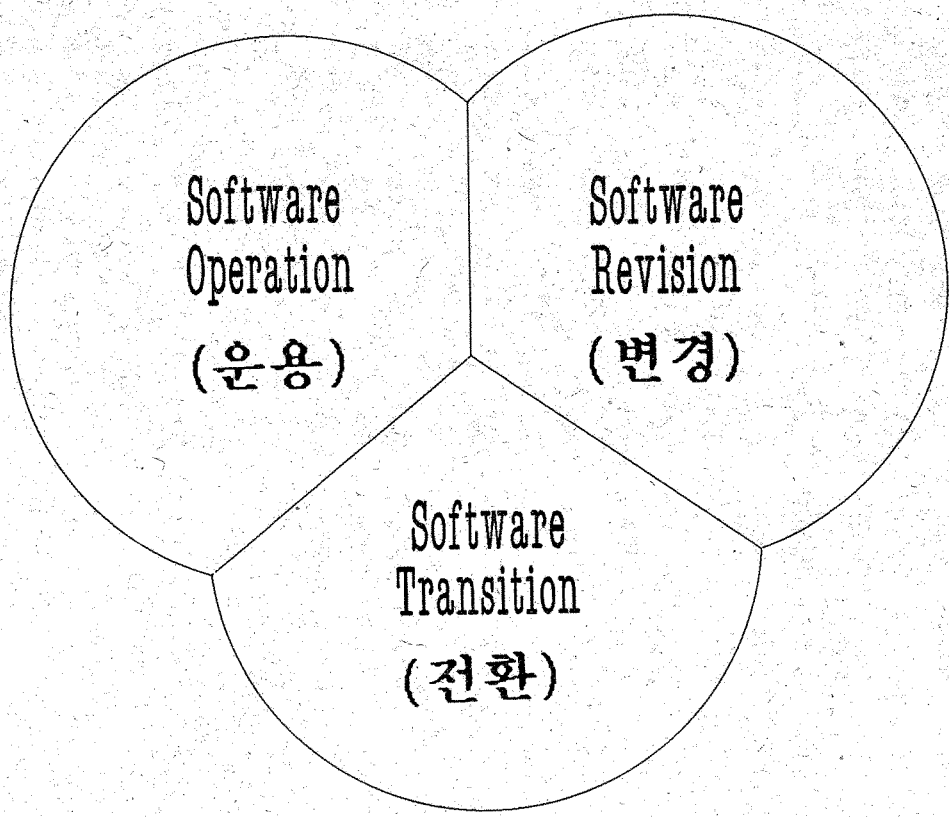
▷ 칠판등에 검토자 들의 의견을 기록하라

▷ 검토자의 수는 통제하고 문제점은 가능한 서면으로 작성하라

▷ 검토할 각 산출물에 대한 검토사항을 작성하라

▷ 검토를 위해 자원과 시간을 할당하라

▷ 모든 검토자에 대해 의미있는 훈련을 실시하라


Software
Operation
(운용)
Software
Revision
(변경)
Software
Transition
(전환)

운용측변에서의 품질요소

▷ Correctness(정확도)

소프트웨어가 요구사항을 만족하는 정도

▷ Reliability(신뢰도)

운용시간 중 오류의 발생으로 정지된 시간의 비율

▷ Efficiency(효율성)

소프트웨어 운용중 필요한 Computing Resource의 양

▷ Integrity(무결성)

인가되지 않은 접근을 통제할 수 있는 정도

▷ Usability(유용성)

입력을 준비하고, 출력을 해석하기 위해 요구되는

노력의 정도

변경측면에서의 품질 요소

▷ Maintainability(유지보수용이성)

오류를 해결하기 위해 요구되는 노력의 정도

▷ Flexibility(유연성)

소프트웨어를 수정하기 위해 요구되는 노력의 정도

▷ Testability(테스트용이성)

기대하는 기능을 수행토록 보장하기 위해 소프트웨어를 테스트하는데 요구되는 노력의 정도

전환측면에서의 품질요소

▷ Portability(이식성)

하드웨어나 소프트웨어 환경의 변화로 프로그램을 변환하는데 요구되는 노력의 정도

▷ Reusability(재사용성)

다른 응용에 재사용 하기위해 요구되는 노력의 정도

▷ Interoperability(연동성)

다른 시스템과 상호 운용되기 위해 요구되는 노력의 정도

소프트웨어 품질 측도

- 감리용이성(AUDITABILITY)
- 정밀도(ACCURACY)
- 통신일반성(COMMUNICATION COMMONALITY)
- 완전성(COMPLETENESS)
- 복잡도(COMPLEXITY)
- 간결성(CONCISENESS)
- 일관성(CONCISTENCY)
- 데이타의 일반성(DATA COMMONALITY)
- 에러의 수용도(ERROR TOLERANCE)
- 수행 효율성(EXECUTION EFFICIENCY)
- 확장성(EXPANDABILITY)
- 일반성(GENERALITY)
- 하드웨어와의 독립성(HARDWARE INDEPENDENCE)
- 에러의 검출성(INSTRUMENTATION)
- 모듈성(MODULARITY)
- 조작용이성(OPERABILITY)
- 보안성(SECURITY)
- 문서도(SELF-DOCUMENTATION)
- 간단성(SIMPLICITY)
- 시스템과의 독립성(SOFTWARE SYSTEM INDEPENDENCE)
- 추적용이성(TRACEABILITY)
- 훈련지원도(TRAINING)

소프트웨어 품질요소와 측도와의 관계

품질영향 요소 / 품질측도	정확도	신뢰도	효율성	무결성	유지보수 용이성	변통성	테스트 용이성	이식성	재사용성	연동성	유용성
감리용이성				*			*				
정밀도		*									
통신일반성										*	
완전성	*										
복잡도		*			*	*	*				
간결성			*		*	*					
일관성	*	*			*	*					
데이타일반성										*	
에러수용도		*									
수행효율성			*								
확장성						*					
일반성						*		*	*	*	
하드웨어와의 독립성								*	*		
에러검출성				*	*		*				
모듈성					*	*	*	*	*	*	
조작용이성			*								*
보안성				*							
문서도					*	*	*	*	*		
간단성					*	*	*				
시스템과의 독립성								*	*		
추적용이성	*										
훈련지원도											*

Halstead의 측도

▷ 프로그램의 길이 (N)

= n1 * log n1 + n2 * log n2

n1 : 프로그램 내에 존재하는 연산자의 수

n2 : 프로그램 내에 존재하는 피연산자의 수

▷ 프로그램의 양 (V)

= N * log (n1 + n2)

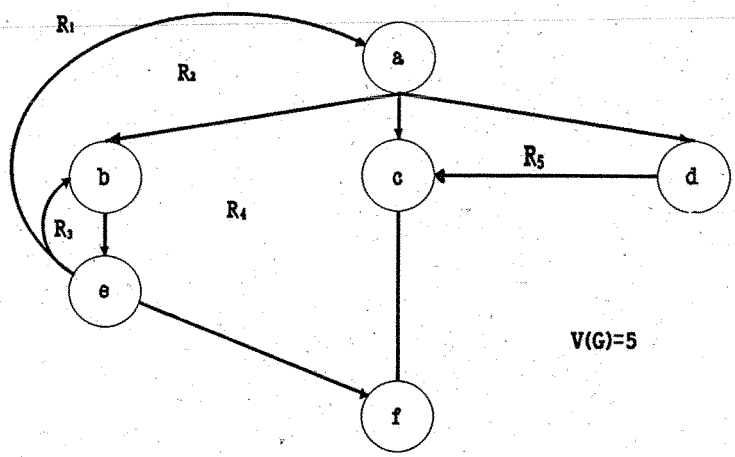
▷ 최소양과의 비율 (Volume Ratio)

= (2 / n1) * (n2 / N2)

N2 : 피연산자의 어커런스 수

McCabe의 복잡도 측도

▷ Control Graph(제어 그래프)에서 폐쇄된 지역의 갯수 (V(G))



▷ V(G)값의 증가는 결정 경로와 반복 갯수의 증가를 나타내므로 복잡도의 증가를 의미

▷ 소프트웨어 신뢰도

특정 기간동안 특정 환경 하에서 컴퓨터 소프트웨어가 Failure(고장) 나지 않고 작동할 확률

▷ Failure

소프트웨어 요구사항과의 불일치

▷ Fault

Failure를 야기시킨 소프트웨어의 결점

▷ Error

Fault를 야기시킨 사람의 행동

소프트웨어 신뢰도 측정

▷ Mean Time Between Failure (M T B F)

M T B F = M T T F + M T T R

MTTF : Mean Time To Failure

MTTR : Mean Time To Repair

▷ Availability(가용도)

$$\text{Availability} = \frac{MTTF}{MTTF + MTTR} * 100$$

▷ Musa의 Execution Model

Reliability = exp(-작동시간 / MTTF)

MTTF = 작동시간 / -ln(Reliability)

소프트웨어 테스트

Software Test

소프트웨어 테스팅

▷ 정의

에러를 찾으려는 의도를 가지고 프로그램을 수행시키는 활동

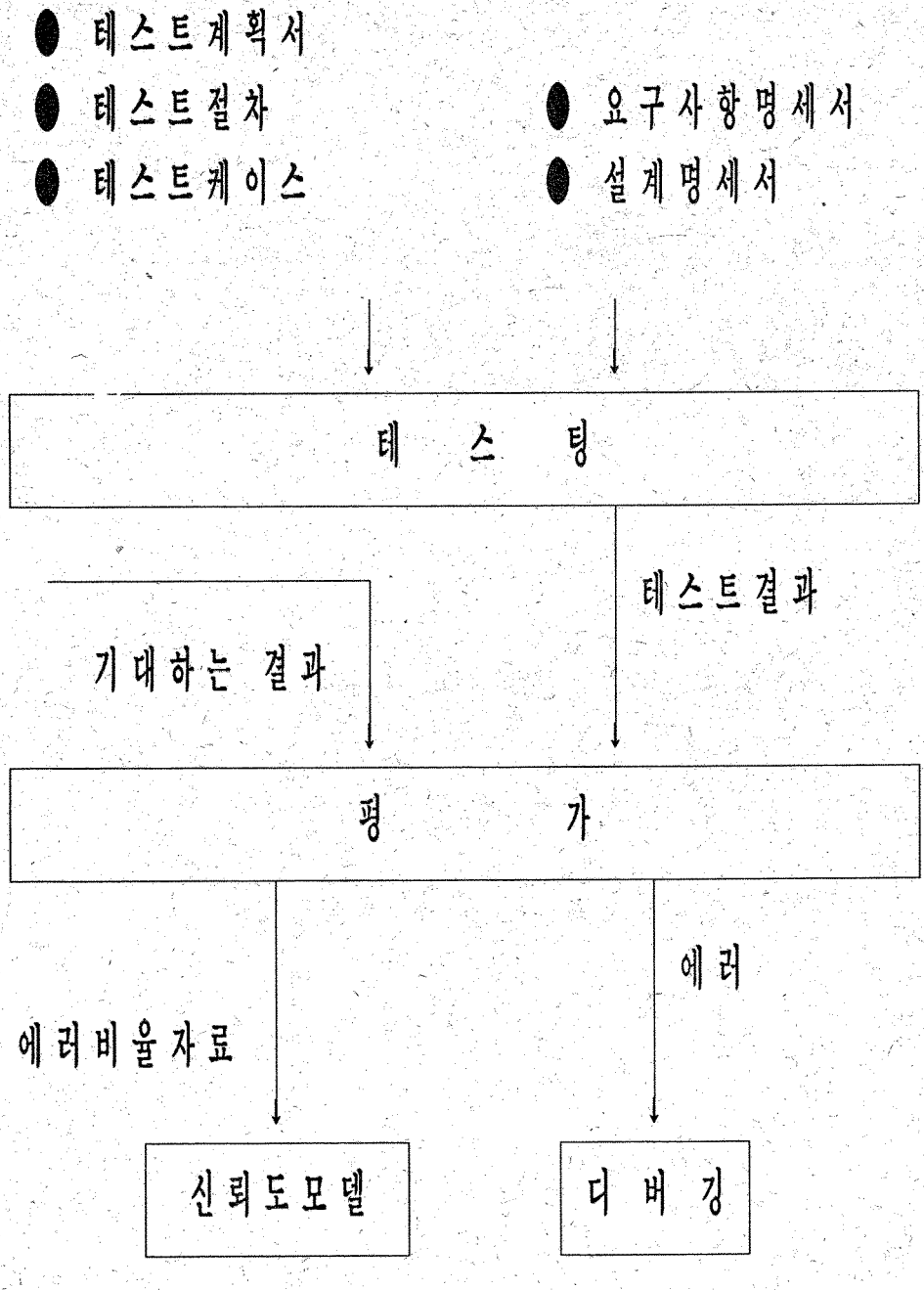
"테스팅은 에러가 없다는 것을 보여주지는 못한다.

오히려 에러가 있다는 것을 보여줄 뿐이다."

▷ 목적

- 소프트웨어에 잠재된 에러의 식별
- 소프트웨어가 명세서에 맞게 수행되도록 보장
- 소프트웨어의 신뢰도 및 품질 제고

테스팅시 정보의 흐름


● 테스트계획서
● 테스트절차
● 테스트케이스
● 요구사항명세서
● 설계명세서
테 스 팅
테스트결과
기대하는 결과
평 가
에러
에러비율자료
신뢰도모델
디 버 깅

테스트시 고려사항

▷ 테스트케이스 설계시 기대하는 출력도 정의하라.

▷ 프로그래머는 자신의 프로그램을 테스트하지 말아라.

▷ 테스트 결과를 철저히 검토하라.

▷ 부적당하거나 기대하지 않는 입력조건도 고려하라.

▷ 기대한 것을 수행하지 않는가 주시하고 또한
기대하지 않은 것을 수행하는 가도 주시하라.

▷ 테스트케이스를 버리지 말고 기록하라.

▷ 에러가 없을 것이라는 기대를 가지고 테스트하지 말아라.

▷ 에러가 발생한 곳에 또 다른 에러가 발생할 확률이 높다는
것을 기억하라.

Static/Dynamic Analysis

▷ Static Analysis (정적분석)

- 프로그램 코드를 직접 수행시키지 않고 수작업 또는 기계로 요구사항명세서, 설계명세서, 코드 등에 잠재된 에러를 발견하려는 테스팅

▷ Dynamic Analysis(동적분석)

- 프로그램 코드를 직접 수행시켜 에러를 발견하려는 테스팅

Verification / Validation

▷ Verification(검증)

개발주기상에서 현단계의 산출물과 바로 전 단계 산출물과의 일치도를 결정하는 과정

"Are we building the product right ?"

▷ Validation (확인)

개발과정 끝에서 최종 산출된 코드가 요구사항에 맞게 수행되는가를 평가하는 과정

"Are we building the right product ?"

코딩단계에서의 정적분석

▷ Code Inspection

여러사람(Group)이 프로그램 코드를 독해함으로써 코드에 잠재된 에러를 발견하려는 정적분석 방법

▷ Walkthrough

여러사람(Group)이 준비된 테스트자료를 가지고 사람이 기계가 되어 프로그램 코드를 가상적으로 수행시키는 정적분석

▷ Desk Checking

한 사람이 Inspection 또는 Walkthrough를 수행하는 정적분석 방법

Code Inspection Checklist

- Coding Style Checking
- Data Reference Checking
- Data Declaration Checking
- Computation Checking
- Comparison Checking
- Control Flow Checking
- Interface Checking
- Input/Output Checking

Data Reference Checking

- 세트 또는 초기화되지 않고 사용된 변수는 없는가 ?
- 범위를 벗어나는 배열의 첨자는 없는가 ?
- 배열의 첨자가 Discrete Type(이산형)인가 ?
- "Dangling Reference"가 존재하지 않는가 ?
- 동일한 기억장소를 여러 변수가 공유하는 경우 변수의 참조가 정확한가 ?
- 포인터 변수가 정의된 데이타형을 참조하는가 ?
- 여러 모듈에서 참조하는 데이타인 경우 각 모듈에서 동일하게 정의되었는가 ?
- 문자열의 각 문자를 지정할 때 문자열의 범위를 벗어나지 않았는가 ?

Data Declaration Checking

▷ 모든 변수는 정확하게 선언되었는가 ?

▷ 선언이 안된 경우 디폴트를 잘 이해하고 있는가 ?

▷ 변수 선언시 초기화하는 경우 정확하게 초기화되었는가 ?

▷ 각 변수에 정확한 데이타형이 선언되었는가 ?

▷ 변수의 초기화는 그 특성에 따라 초기화되었는가 ?

▷ 유사한 명칭을 갖은 변수들은 없는가 ?

Computation Checking

▷ 상이한 데이타형들 간의 연산이 존재하는가 ?

▷ Assignment연산에서 LHS의 길이가 BHS의 길이보다 작지않은가 ?

▷ 연산중 Overflow 또는 Underflow가 가능하지 않은가 ?

▷ 나눗셈에서 젯수가 "0"이 되는 경우는 없는가 ?

▷ 변수의 값이 그 변수의 의미있는 범위를 넘지 않는가 ?

▷ 산술연산자들의 수행 우선순위는 바르게 인식되었는가 ?

▷ Integer를 포함한 타당치 않은 산술식(특히 나눗셈)은 없는가 ?

Comparison Checking

▷ 상이한 테이타형 간의 비교는 없는가 ?

▷ 관계연산자의 사용은 정확한가 ?

▷ 논리연산자의 사용은 정확한가 ?

▷ 논리연산자의 피연산자는 부울형인가 ?

▷ 논리연산자들의 우선순위는 정확하게 인식되었는가 ?

▷ 실수와 분수와의 비교는 없는가 ?

Control Flow Checking

- Multiway Branch에서 가지수가 가능한 수를 넘지 않는가 ?
- Multiway Barnch에서 그가지수는 완전한가 ?
- 반복은 종료 가능한가 ?
- 프로그램은 종료 가능한가 ?
- 반복은 수행될 수 있는가 ?
- 반복 상의 "Off by One"에러는 없는가 ?

Interface Checking

- ▷ 입출력 매개변수의 수와 순서가 일치하는가 ?
- ▷ 입출력 매개변수의 속성은 일치하는가 ?
- ▷ 매개변수들의 단위는 일치하는가 ?
- ▷ Built-In Function을 호출할 때, 매개변수의 수, 속성, 순서 등이 일치하는가 ?
- ▷ 입력 매개변수에 수정을 가하지 않았는가 ?
- ▷ Global변수의 선언이 이를 사용하는 모든 모듈에서 일치하는가 ?

Input/Output Checking

- 화일은 정확하게 선언되었는가 ?
- 입출력문의 형식(Format) 명세는 정확한가 ?
- 모든 화일은 사용하기 전에 Open되었는가 ?
- End-of-file조건은 정확하게 기술되었는가 ?
- 프로그램 출력에는 철자상 또는 구문상의 에러가 존재하지 않는가 ?

Terminology

▷ Test Item

테스팅의 대상이 되는 소프트웨어 항목

▷ Test Objective

테스팅에 의해 평가하고자하는 소프트웨어의 Features

▷ Test Case

특정 Test Objective와 관련된 입력조건, 입력자료, 기대하는 결과

▷ Test Procedure

테스팅을 수행하기 위한 작업 (Setup, Operation, Evaluation 등)의 순서

테스트 케이스의 설계

▷ Black-Box 테스팅

☞ Data-Driven, I/O-Driven 테스팅

☞ 프로그램 내부구조 및 특성을 고려치 않고 프로그램 관련 명세서를 분석하여 테스트 케이스를 설계

☞ 모든 입력에 대해 테스팅 한다는 것은 불가능

White-Box 테스팅

☞ Logic-Driven 테스팅

☞ 프로그램의 내부 논리구조를 분석하여 테스트 케이스 설계

☞ 모든 경로를 테스팅 한다는 것은 불가능

Black-Box / White-Box 테스팅

▷ Black-Box 테스팅

- ☞ Equivalence Partitioning
- ☞ Boundary-Value Anaysis
- ☞ Cause-Effect Graphing
- ☞ Error Guessing

▷ White-Box 테스팅

- ☞ Statement Coverage
- ☞ Decision Coverage
- ☞ Condition Coverage
- ☞ Decision/Condition Coverage
- ☞ Path Coverage

White-Box 테스팅

▷ Statement Coverage

프로그램 내의 모든 명령문을 적어도 한번 실행하도록 테스트 케이스를 설계

▷ Decision Coverage

프로그램 내의 모든 결정문이 적어도 한번 "참"과 "거짓"이 되도록 테스트 케이스를 설계

▷ Condition Coverage

결정문 내의 각 조건들이 적어도 한번 "참"과 "거짓"이 되도록 테스트 케이스를 설계

▷ Decision/Condition Converage

Decision 및 Condition Coverage를 모두 만족하도록 설계

▷ Path Coverage

모든 경로를 적어도 한번 수행하도록 테스트 케이스를 설계

Equivalence Partitioning

▷ 프로그램 관련 명세서로 부터 테스트할 요소를 식별하고 식별된 각 요소에 따른 적합한 Equivalence Class와 부적합한 Equivalence Class를 식별함으로써 테스트 케이스를 설계

- 식별된 각 Equivalence Class에 유일한 수를 할당
- 적합한 Equivalence Class에 대해서는 가능한 적은 수의 입력자료를 선정
- 부적합한 Equivalence Class에 대해서는 각 Equivalence Class에 적어도 하나의 입력자료를 선정

Equivalence Class의 설계

▷ Equivalence Class의 설계를 위한 Table 작성

테스팅요소	적합한 E.C	부적합한요소

▷ Equivalence Class식별을 위한 지침

☞ 값의 범위를 주시하라.

☞ 값의 수를 주시하라.

☞ 값의 집합을 주시하라.

☞ 필연조건을 주시하라.

☞ Equivalence Class를

더 작은 Equivalence Class로 분할하라.

Boundary Value Analysis

▷ 입력및 출력의 경계치 조건을 보다 중점적으로 분석하여 테스트 케이스를 설계

▷ 경계치 조건의 식별을 위한 지침

- ☞ 입력 값의 범위를 주시하라.
- ☞ 입력 값의 수를 주시하라.
- ☞ 출력 값의 범위를 주시하라.
- ☞ 출력 값의 수를 주시하라.
- ☞ 최초/최종 입출력 항목에 주시하라.
- ☞ 기타 경계치 조건을 식별하라.

Cause-Effect Graphing

▷ 테스트 케이스 설계 절차

- 명세서를 구분 가능한 조각으로 분할
- 명세서에서 Cause와 Effect를 식별
- Cause와 Effect를 연관시켜주는 Cause Effect Graph를 작성
- 구문적 또는 환경적 제한사항을 Graph에 첨가
- Graph를 역으로 추적해 감으로써 Graph를 Decision Table로 변환
- Decision Table의 각 열로 부터 테스트 자료를 선정

▷ 개별적인 입력조건 뿐만 아니라 이들의 조합에서 야기될 수 있는 입력조건을 테스트함으로써 완전한 테스트 케이스를 설계

Cause-Effect Graph

▷ Cause

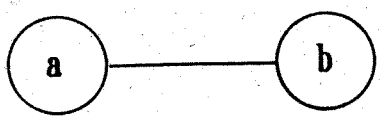
상이한 입력조건 또는 입력조건에 대한 Equivalence Calss

▷ Effect

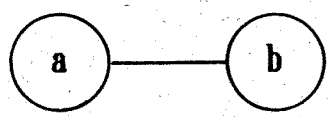
출력 또는 시스템의 상태 변환

▷ 기본적인 Cause-Effect Graph

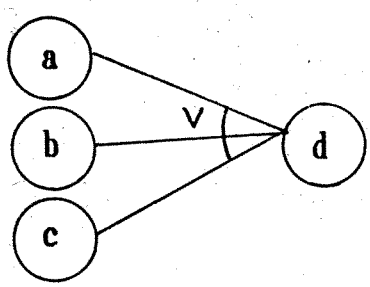
Identity



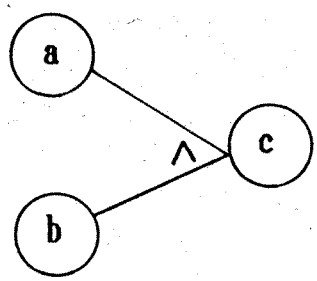
Not



or

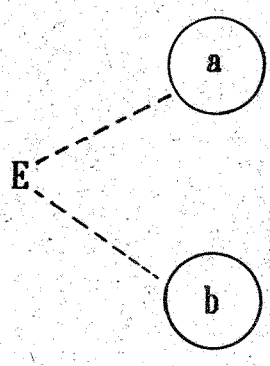


And

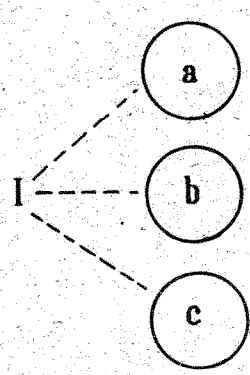


구문적/환경적 제한사항

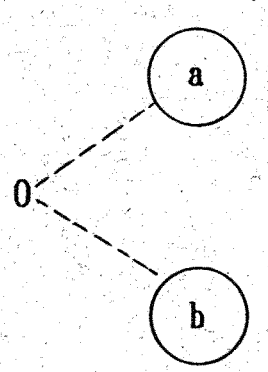
Exclusive



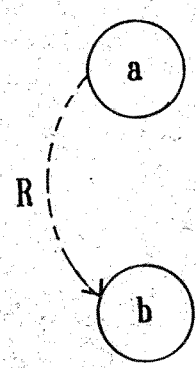
Inclusive



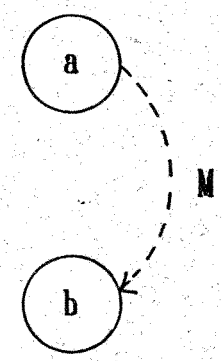
One and Only One



Requires



Masks



Graph 추적시 고려사항

▷ 출력이 "0"인 "OR"노드와 출력이 "1"인 "AND"노드 추적시에는 모든 경우를 추적하라.

▷ 출력이 "1"인 "OR"노드 추적시에는 입력중 하나만 "1"로 세트되는 한 경우 만을 추적하라.

▷ 출력이 "0"인 "AND"노드 추적시에는 모든 입력을 추적하되

- 입력을 모두 "0"으로 세트시키는 한 경우만을 추적하라
- 입력을 "1"로 세트시키는 한 경우만을 추적하라.

Error Guessing

▷ 특정 테스팅 기법에 상관없이 직관과 경험에 의해 발생확률이 많은 레러를 추측하고 이러한 에러를 검출하는 테스트 케이스를 설계

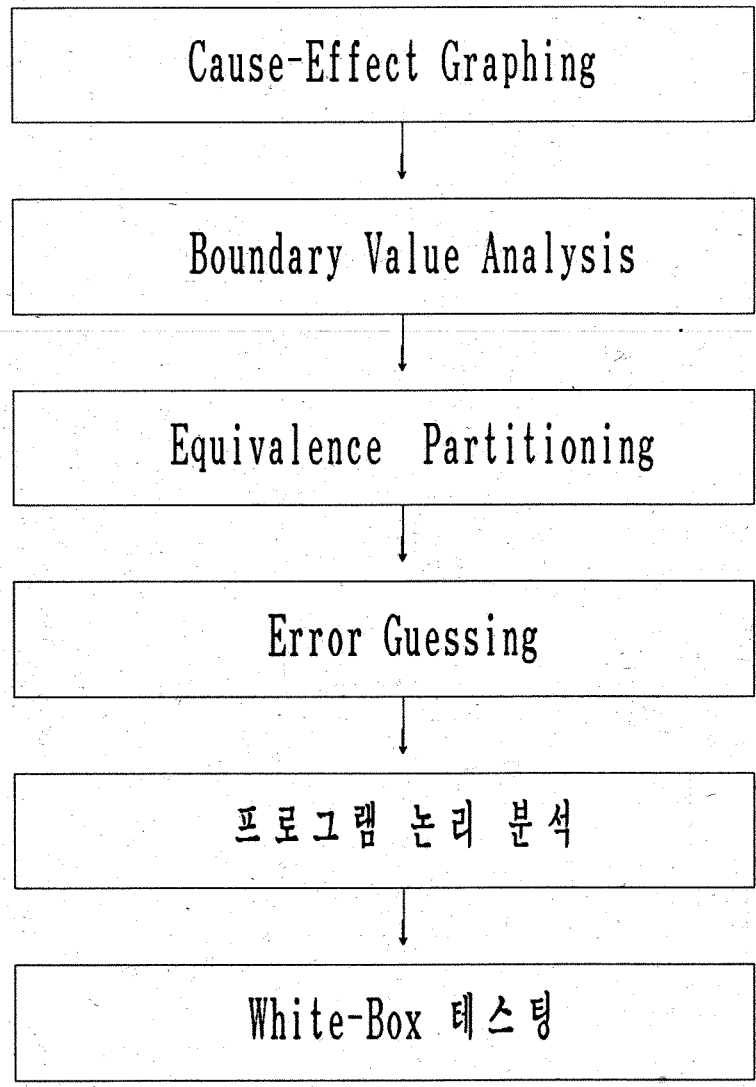
▷ Error Guessing시 지침

☞ 에러발생 경향이 높은 상황을 주시하라.

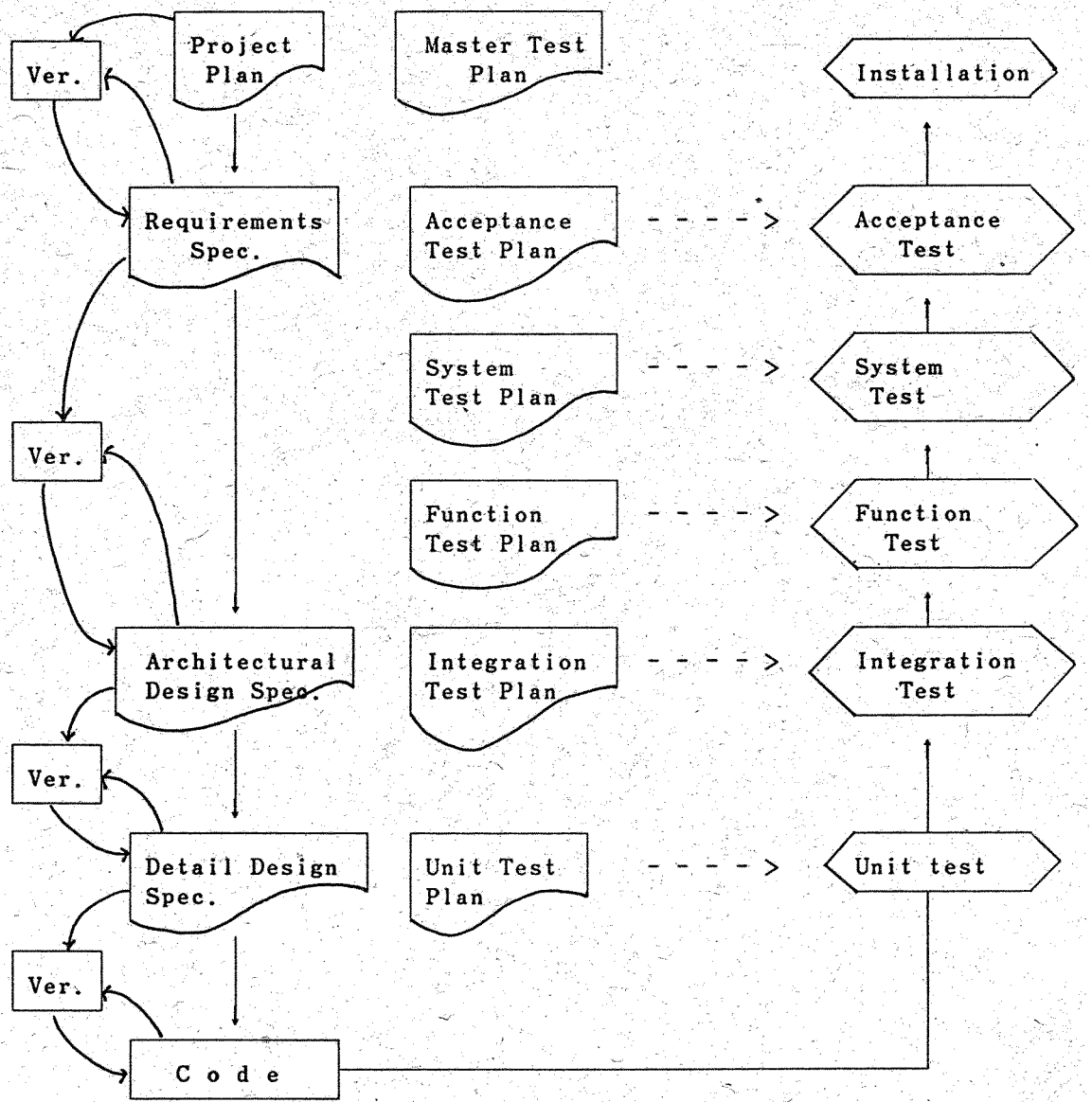
☞ 자명한 사실에 데헤 주시하라.

☞ 프로그램 설계시 누락될 수 있는 특수한 상황에 주시하라.

테스트 케이스 설계 전략


Cause-Effect Graphing
Boundary Value Analysis
Equivalence Partitioning
Error Guessing
프로그램 논리 분석
White-Box 테스팅

테스팅 전략

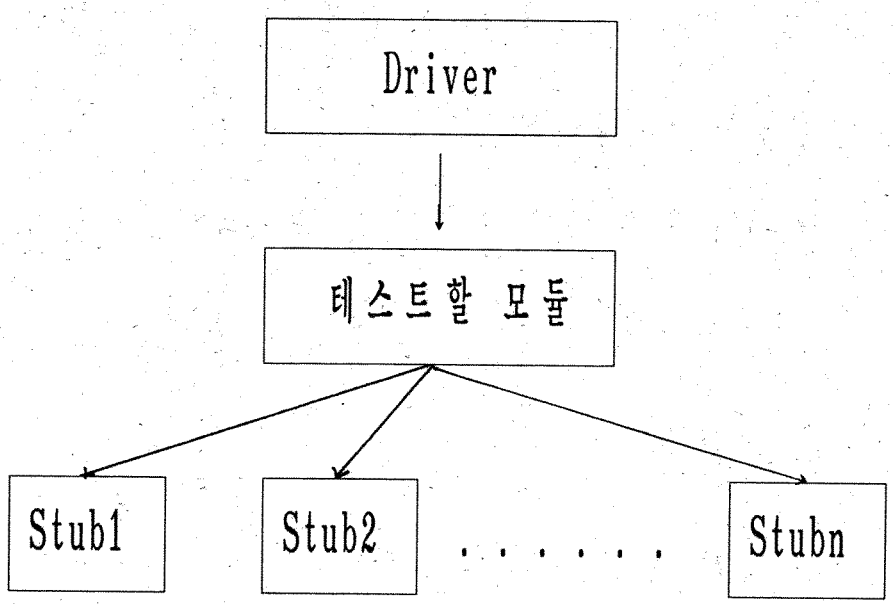

Project Plan
Master Test Plan
Installation
Ver.
Requirements Spec.
Acceptance Test Plan
Acceptance Test
System Test Plan
System Test
Ver.
Function Test Plan
Function Test
Architectural Design Spec.
Integration Test Plan
Integration Test
Ver.
Detail Design Spec.
Unit Test Plan
Unit test
Ver.
C o d e

Unit Test

▷ 개개의 단위 모듈을 테스팅

☞ Module Test

▷ White-Box 중심의 테스팅



Integration Test

▷ Non-Incremental Integration Testing(비점증적 통합테스팅)

각 모듈에 대해 개별적으로 Unit Test를 수행하고 Unit Test가 모두 종료되면 모든 모듈을 동시에 통합하여 테스팅

▷ Incremental Integration Testing(점증적 통합테스팅)

Unit Test할 모듈을 기 통합되어 테스트된 모듈과 먼저 통합한 후 테스팅

- 테스팅에 소요되는 노력 감소
- 모듈간의 접속에서 발생되는 에러를 조기에 발견
- 디버깅이 용이
- 좀더 철저한 테스팅 수행
- 기계 수행시간의 많은 낭비
- 병렬작업의 기회가 감소

Incremental Integration Test

▷ Top-Down Integration (하향식 통합)

- ☞ 상위 모듈을 하위 모듈보다 먼저 테스트
- ☞ 중요한 모듈을 가능한 먼저 테스트
- ☞ 입/출력과 관련된 모듈을 가능한 먼저 테스트

▷ Bottom-Up Integration (상향식 통합)

- ☞ 하위 모듈을 상위 모듈보다 먼저 테스트
- ☞ 입/출력과 관련된 모듈을 먼저 테스트
- ☞ 중요한 모듈을 가능한 먼저 테스트

Top-Down / Bottom-Up Integration

	Top-Down	Bottom-Up
장점	▷주요 결점이 프로그램의 상위에 존재 할 때 유효 ▷데모가 조기에 가능	▷주요결점이 프로그램의 하위에 존재 할 때 유효 ▷테스트케이스 설계가 용이
단점	▷Stub모듈이 필요 ▷하위모듈의 완성전까지는 상위 모듈의 완전한 테스트가 불가능 ▷테스트케이스 설계가 용이치 않음	▷Driver모듈이 필요 ▷최종 모듈이 테스트되기 전 까지는 데모가 불가능

Function Test

▷ 사용자의 기능적 요구사항과 구현된 시스템과의 기능적 불일치를 식별

▷ Black-Box중심의 테스팅

▷ Function Testing 시 지침

☞ 어떠한 기능이 가장 많은 에러를 포함하는지 주시하라.

☞ 부적합하고 기대하지 않는 입력조건에 충분한 관심을 가져라

☞ 기대하는 결과를 기술하라

☞ 테스팅의 목적은 에러를 검출하는 것임을 잊지 말아라

System Test

▷ 사용자의 비기능적 요구사항과 구현된 시스템과의 성능적인 측변과의 불일치를 식별

▷ System Test의 종류

☞ Facility, Volume, Stress, Usability
Security, Configuration, Compatability
Installibility, Reliability, Recovery
Maintainability, Documentation, Procedure
Storage, ...

Acceptance Test

▷ 사용자의 최종 승인을 얻기 위해 사용자의 요구사항과 구현된 시스템과의 불일치를 식별하기 위한 테스팅

▷ 시스템의 요구사항을 만족하지 않는 방향으로 테스트케이스 설계

▷ 시스템 사용자 또는 사용자가 고용한 제 3의 테스팅 그룹에 의해 수행

테스팅의 종료

▷ 계획된 검출없이 테스팅 수행시 자연적으로 종료

▷ 에러의 검출없이 테스팅 수행시 종료

▷ 특정 테스트 방법에 따라 설계된 케이스를
모두 수행하면 종료

▷ 에러의 수를 예측하여 예측한 수만큼 에러를 발견하면
종료

▷ 단위 시간당 검출된 에러의 분포를 관찰하여 에러
검출율이 감소할 때 타당성있는 시점에서 종료

Debugging

▷ 테스팅 후 에러의 정확한 특성과 그 위치를 식별하고 에러를 정정하는 활동

▷ 디버깅의 어려움

☞ 심리적으로 부담을 주는 활동

☞ 다른 활동에 비해 연구 및 정형화된 이론 부재

▷ 디버깅 방법

☞ Brute Force에 의한 디버깅

☞ 귀납적 방법에 의한 디버깅

☞ 연연적 방법에 의한 디버깅

☞ Backtracking 에 의한 디버깅

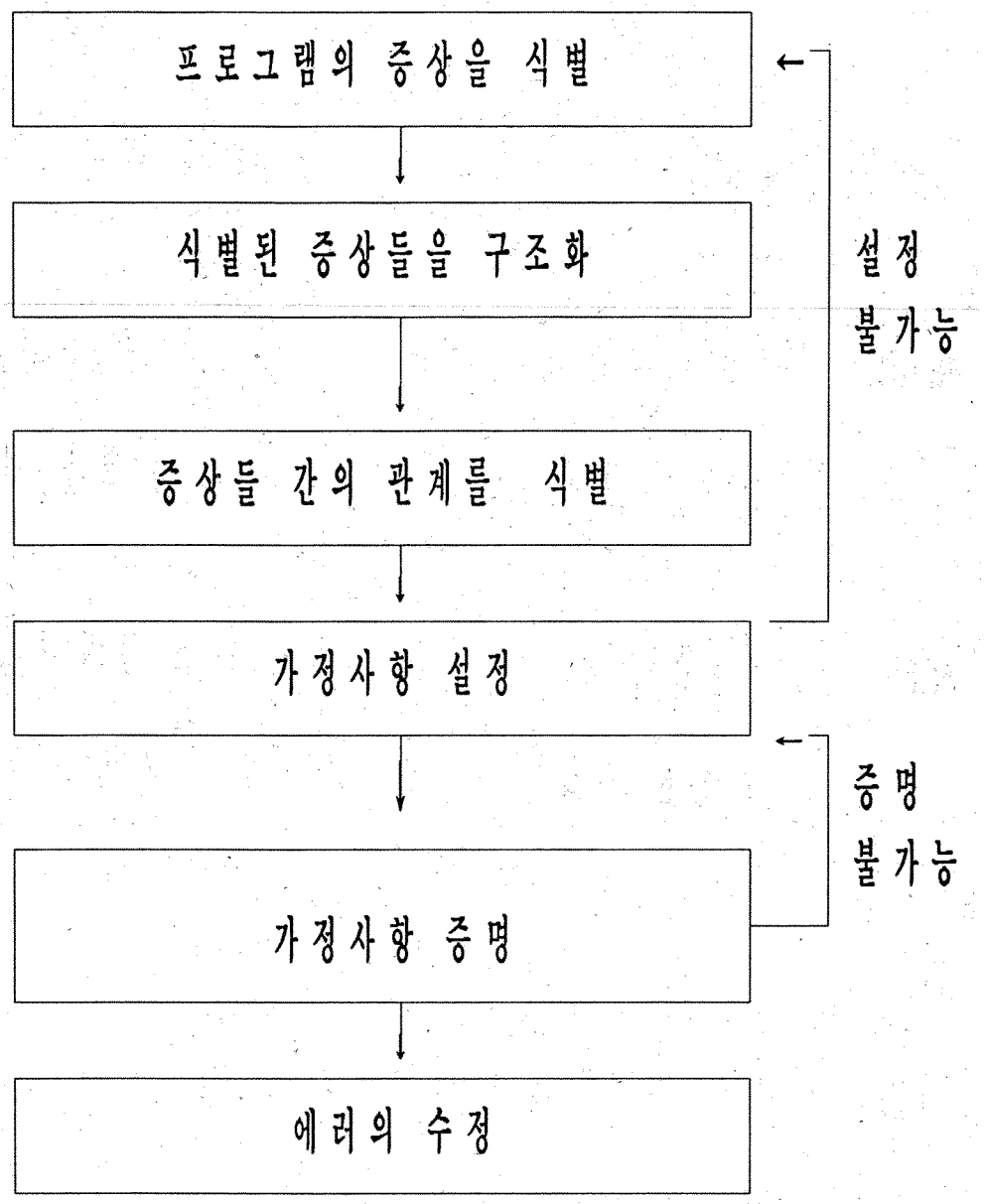
Brute Force에 의한 디버깅

▷ 가장 일반적인 방법으로 적은 사고와 정신적 부담을 주지만 비효율적인 방법

▷ Brute Force에 의한 방법

- ☞ Storage Dump에 의한 디버깅
- ☞ 출력문 삽입에 의한 디버깅
- ☞ 자동화 도구에 의한 디버깅

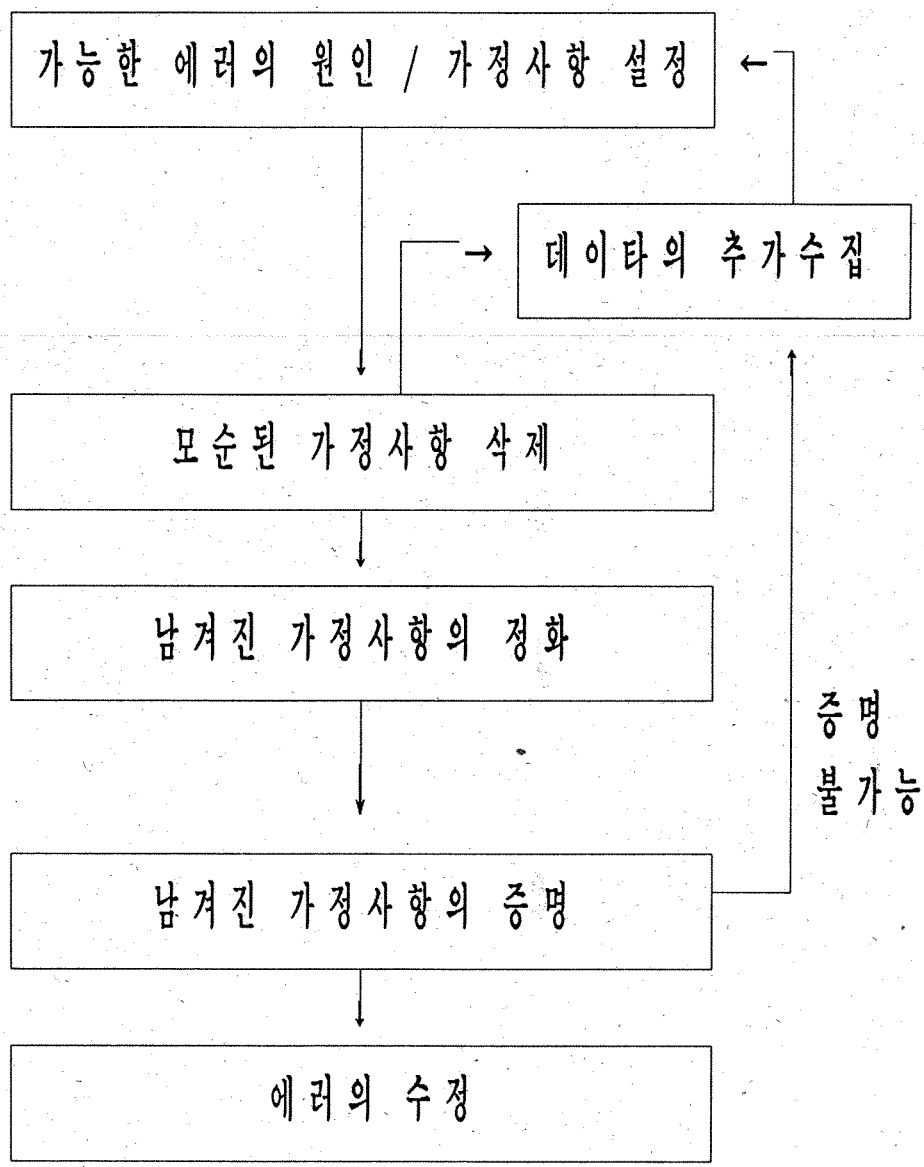
귀납적 방법에 의한 디버깅



증상의 구조화

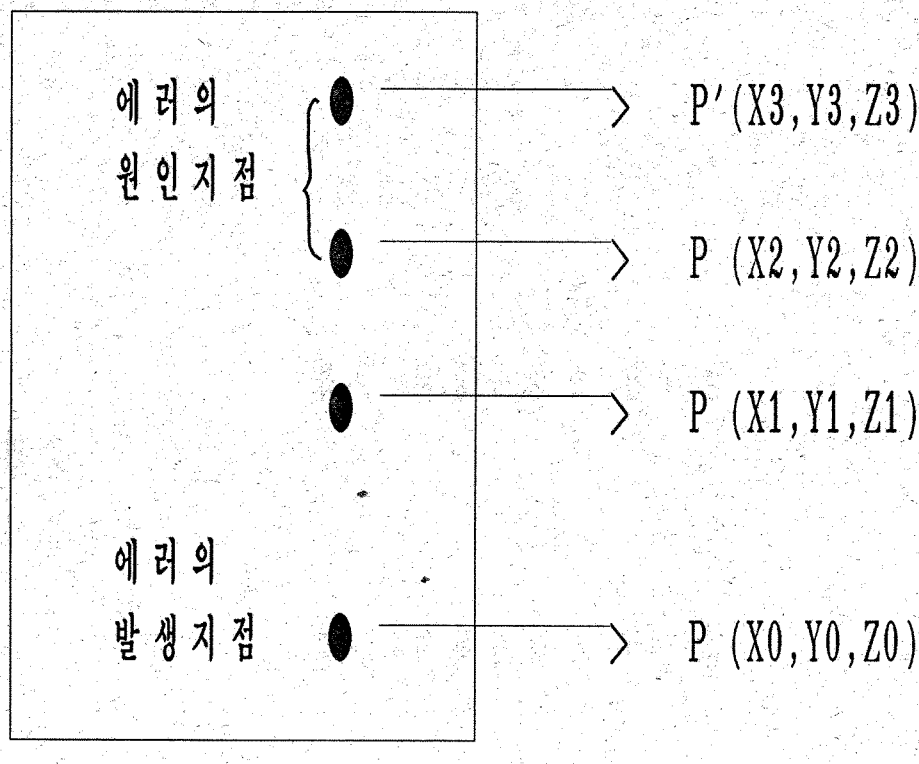
	IS	IS NOT
WHAT	에러의 증상을 기술	에러가 아닌 증상을 기술
WHERE	에러의 증상이 나타난 출력을 기술	에러의 증상이 나타나지 않은 출력을 기술
WHEN	에러의 증상이 나타난 시점을 기술	에러의 증상이 나타나지 않는 시점을 기술
TO WHAT EXTENT	다른 입력자료에 대한 증상을 기술	

연역적 방법에 의한 디버깅



Backtracking에 의한 디버깅

▷ 에러가 발생한 지점으로 부터 수작업으로 역수행하여 에러의 원인 지점을 식별



P : 기대하는 프로그램의 상태

P': 기대하지 않는 프로그램의 상태

디버깅시 고려사항

▷ 난국에 부딪쳤을 때는 일을 잠시 그만 두어라

▷ 난국에 부딪쳤을 때 다른 사람과 문제를 토의하라

▷ 하나의 에러가 있는 곳에 다른 에러가 존재할
확률이 높다는 것을 잊지 말아라

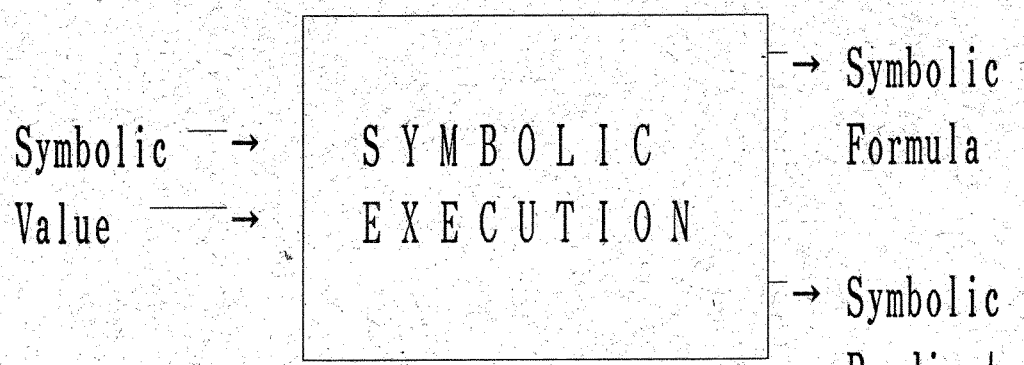
▷ 수정한 후 원프로그램보다 더 철저히 테스팅하라

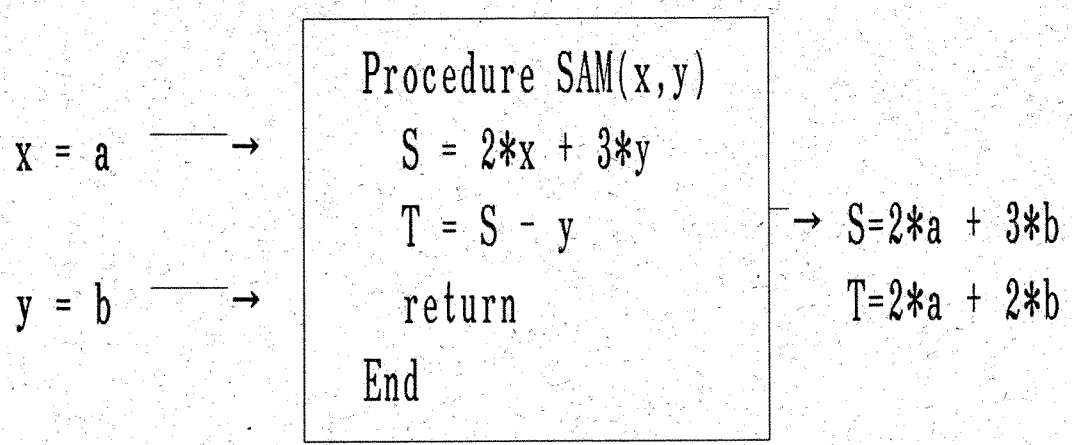
▷ 정확하게 수정할 확률은 프로그램 크기에
반비례함을 잊지 말아라

▷ 에러의 수정이 새로운 에러를 야기시킬 수 있다는
것을 기억하라

Symbolic Execution

▷ Input Value를 "Symbolic"상수로 하여 프로그램을 수행시킴으로써 에러를 발견하려는 테스팅/디버깅 방법





디버깅시 고려사항

▷ 난국에 부딪쳤을 때는 일을 잠시 그만 두어라

▷ 난국에 부딪쳤을 때 다른 사람과 문제를 토의하라

▷ 하나의 에러가 있는 곳에 다른 에러가 존재할
확률이 높다는 것을 잊지 말아라

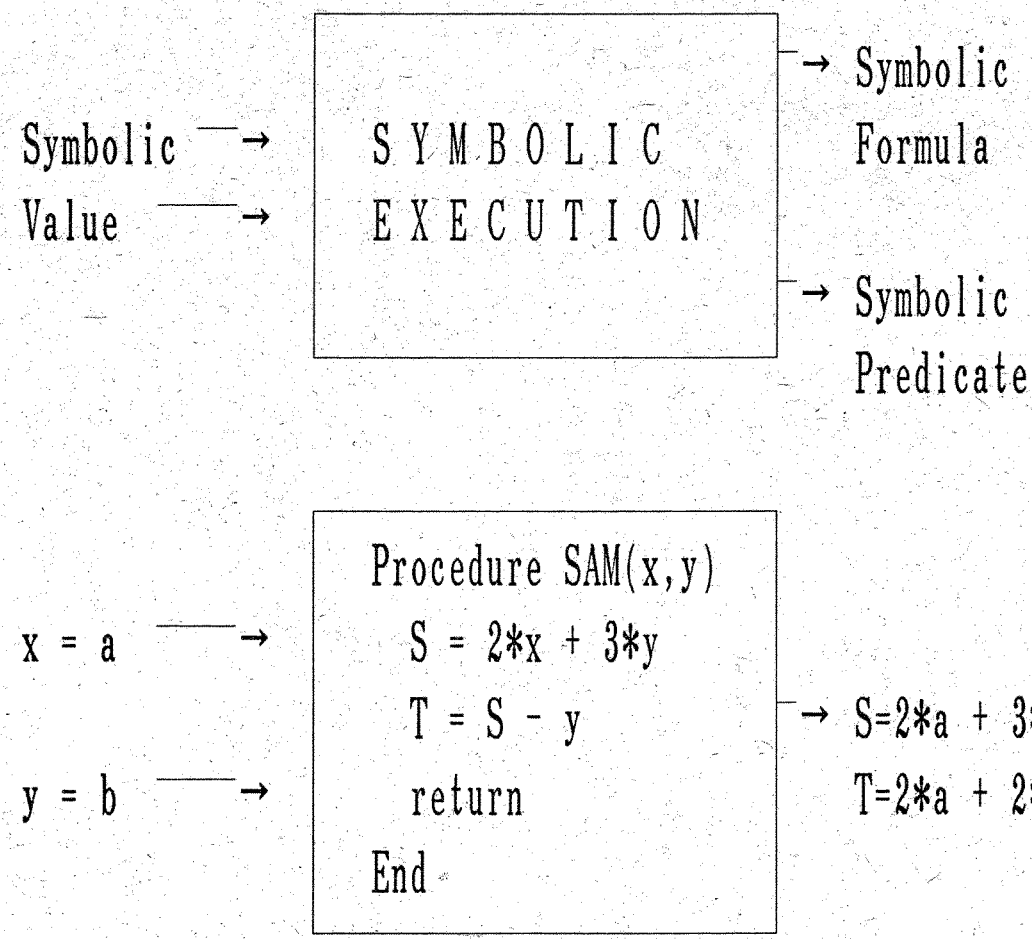
▷ 수정한 후 원프로그램보다 더 철저히 테스팅하라

▷ 정확하게 수정할 확률은 프로그램 크기에
반비례함을 잊지 말아라

▷ 에러의 수정이 새로운 에러를 야기시킬 수 있다는
것을 기억하라

Symbolic Execution

▷ Input Value를 "Symbolic"상수로 하여 프로그램을 수행시킴으로써 에러를 발견하려는 테스팅/디버깅 방법



Program Instrumentation

▷ 기능적인 측면에서는 영향을 주지 않고, 특정 변수의 범위, 문자의 수행 횟수 등의 프로그램 특성을 검사하기 위해 프로그램 내에 이러한 특성을 기록할 수 있는 Recording Statement를 삽입하여 에러를 식별하려는 테스팅/디버깅 방법

▷ Recording Statement

- ☞ Software Probe
- ☞ Software Monitor
- ☞ Software Instrument

Mutation Testing

▷ 테스트 자료의 Adequacy(적합성)를 측정하기 위해 테스트할 프로그램을 변형시킨 프로그램(Mutant)에 테스트 자료를 입력하여 수행시키는 테스팅 기법

▷ 테스트자료의 Adequacy

특정 테스트자료가 올바른 프로그램에 대해서는 바르게 수행하고, 에러가 포함된 프로그램에 대해서는 바르지 않게 수행하는 정도

▷ Adequacy의 측정

$$MS = dm / m$$

MS : Mutation Score

m : Mutant의 수

dm : 기대하지 않은 결과를 야기시킨 Mutant의 수

자동화 테스팅 / 디버깅 도구

- ▷ Static Analyzer ☞ FAVES , JAVES , RXVP
- ▷ Dynamic Analyzer ☞ DYNA
- ▷ Test Data Generator ☞ ATTEST , DATAMACS , COBOL/DV
- ▷ Test Driver ☞ AUT , RXVP , TEST MANAGER
- ▷ Instrumenter ☞ COBOL/FORTRAN INSTRUMENTER , JAVS , PET
- ▷ Symbolic Evaluator ☞ ATTEST , DISSECT , EFFIGY
- ▷ Coverage Analyzer ☞ JAVS , PET , RXVP
- ▷ Automatic Mutation System

  ☞ PROTABLE FORTRAN MUTATION SYSTEM, TEC/1

소프트웨어 유지보수와 형상관리

Software Maintenance and Configuration Management

Software Maintenance와 Configuration Management

▷ Software Maintenance(스프트웨어 유지보수)

☞ 소프트웨어가 사용자에게 인도되고 운용을 시작하면서 발생하는 변경을 관리하기 위해 수행하는 소프트웨어 공학 활동의 집합

☞ 인도이후의 변경관리

▷ Software Configuration Management(소프트웨어 형상관리)

☞ 소프트웨어가 사용자에게 인도되기 전의 개발과정 상에서 발생하는 변경을 관리하기 위해 수행하는 소프트웨어 공학의 활동의 집합

☞ 인도이전의 변경관리

유지보수의 유형

▷ Corrective Maintenance(정정을 위한 유지보수)

인도 이후 소프트웨어에 발생하는 에러의 진단과 정정을 위한 활동

▷ Adaptive Maintenance(적응을 위한 유지보수)

변화하는 환경에 적합한 인터페이스를 유지하기 위해 소프트웨어를 수정하는 활동

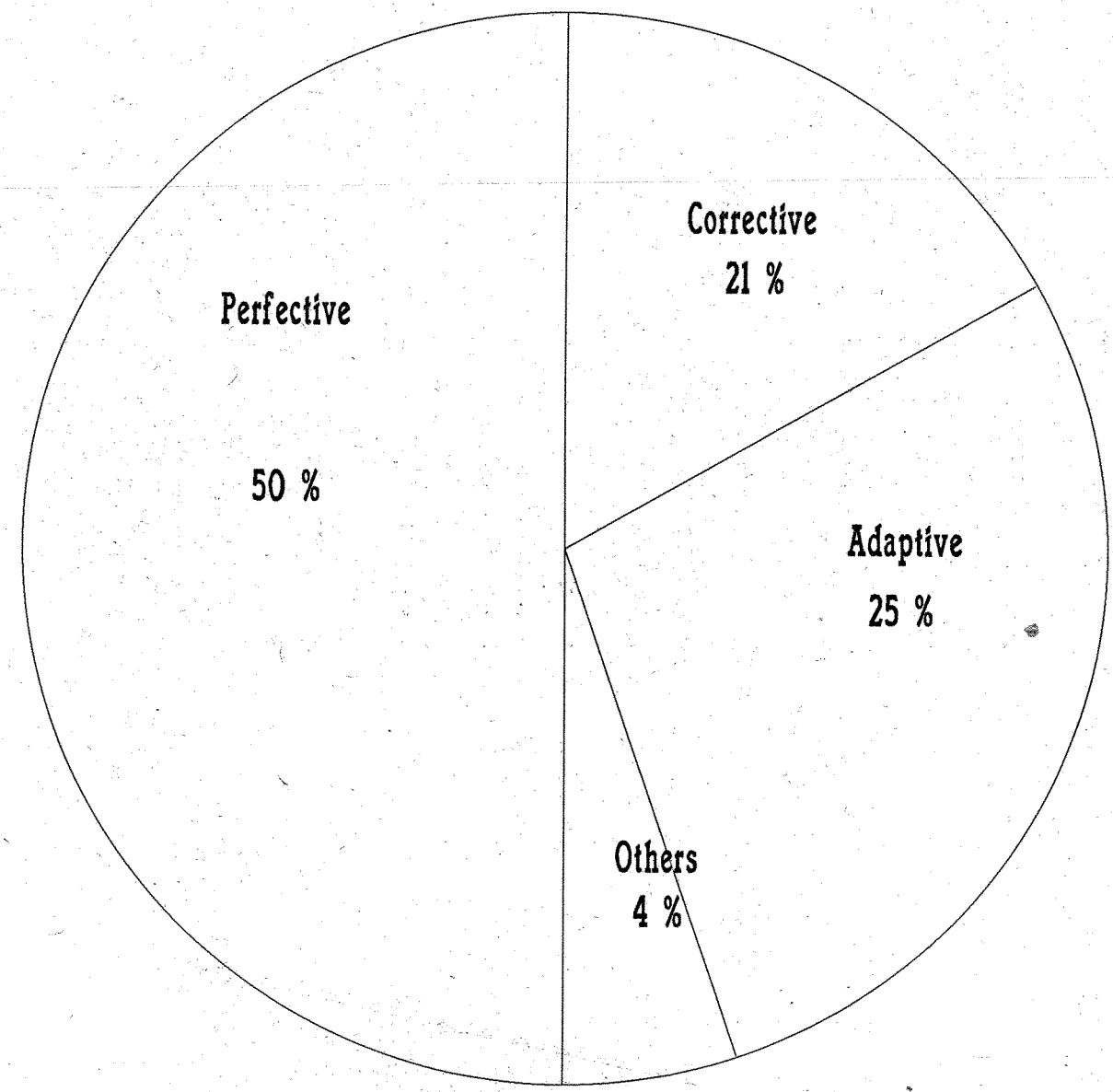
▷ Perfective Maintenance(보완을 위한 유지보수)

사용자의 새로운 성능 요구, 현존 기능의 변경 및 일반적인 요구사항을 만족하도록 소프트웨어를 수정하는 활동

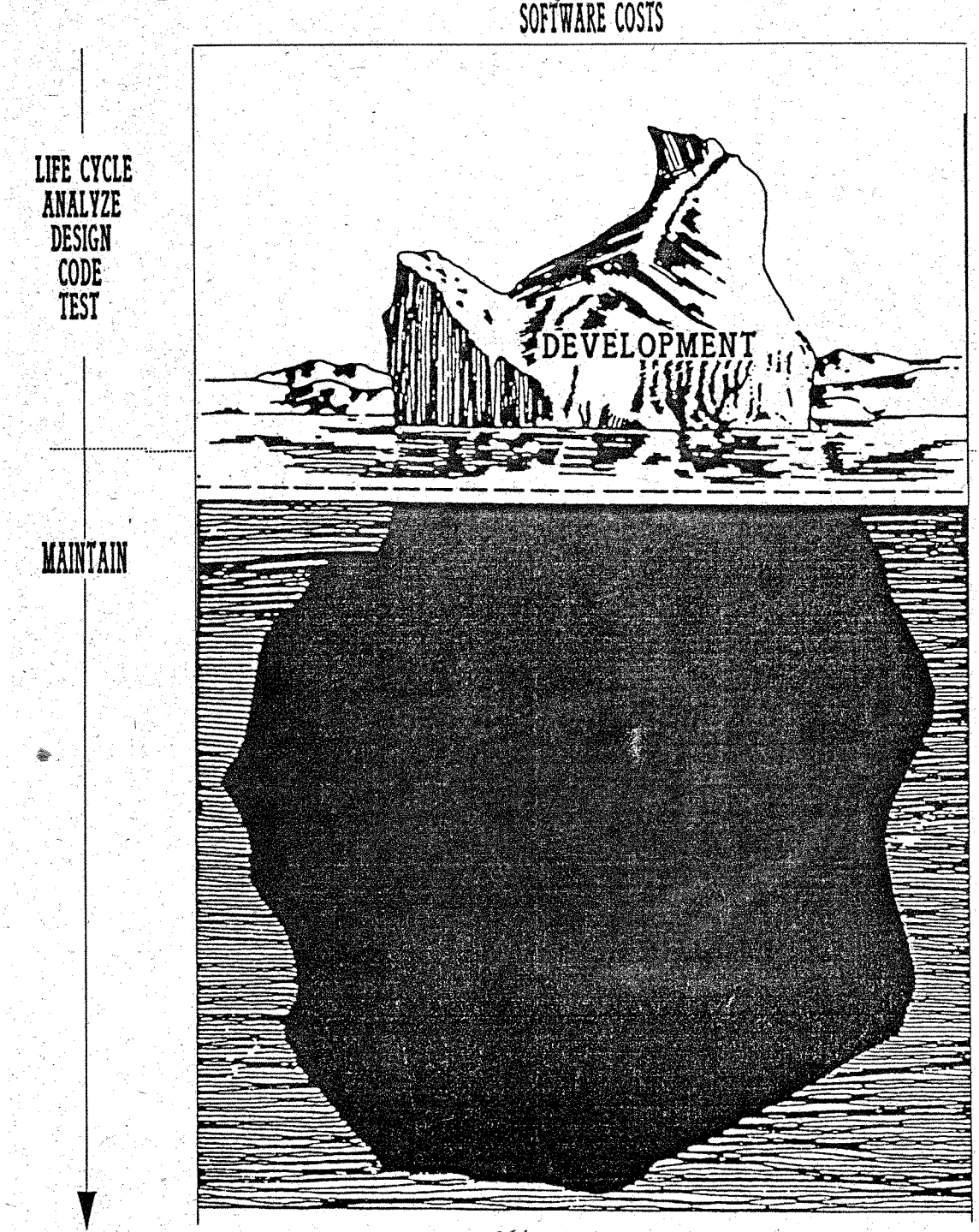
▷ Preventive Maintenance(품질제고를 위한 유지보수)

향후 용이한 유지보수를 위해 유지보수용이도, 신뢰도 등을 제고하도록 소프트웨어를 수정하는 활동 。

유지보수 유형별 노력비율



소프트웨어 유지보수의 중요성



유지보수의 문제점

▷ 다른 사람이 개발한 프로그램을 이해하기가 어렵다.

▷ 소프트웨어 관련 요원들의 빈번한 이동으로 개발자로부터 정보를 얻기가 힘들다.

▷ 프로그램 관련 문서가 없거나 문서의 질이 형편없다.

▷ 대부분의 소프트웨어가 유지보수용이성을 고려하여 설계되어 있지 않다.

▷ 유지보수 활동을 창조적이고, 흥미있는 것으로 인식하지 않고 있다.

생산성의 감소

유지보수용이성에 영향을 주는 요소들

유지보수를 고려한 개발

▷ 요구사항 검토시

- ☞ 향후 보완 및 수정가능 분야 식별
- ☞ 소프트웨어 이식성 고려
- ☞ 유지보수에 영향을 줄 시스템 인터페이스 식별

▷ 설계서 검토시

- ☞ 설계서의 수정용이성 검토
- ☞ 모듈성, 기능적 독립성 검토

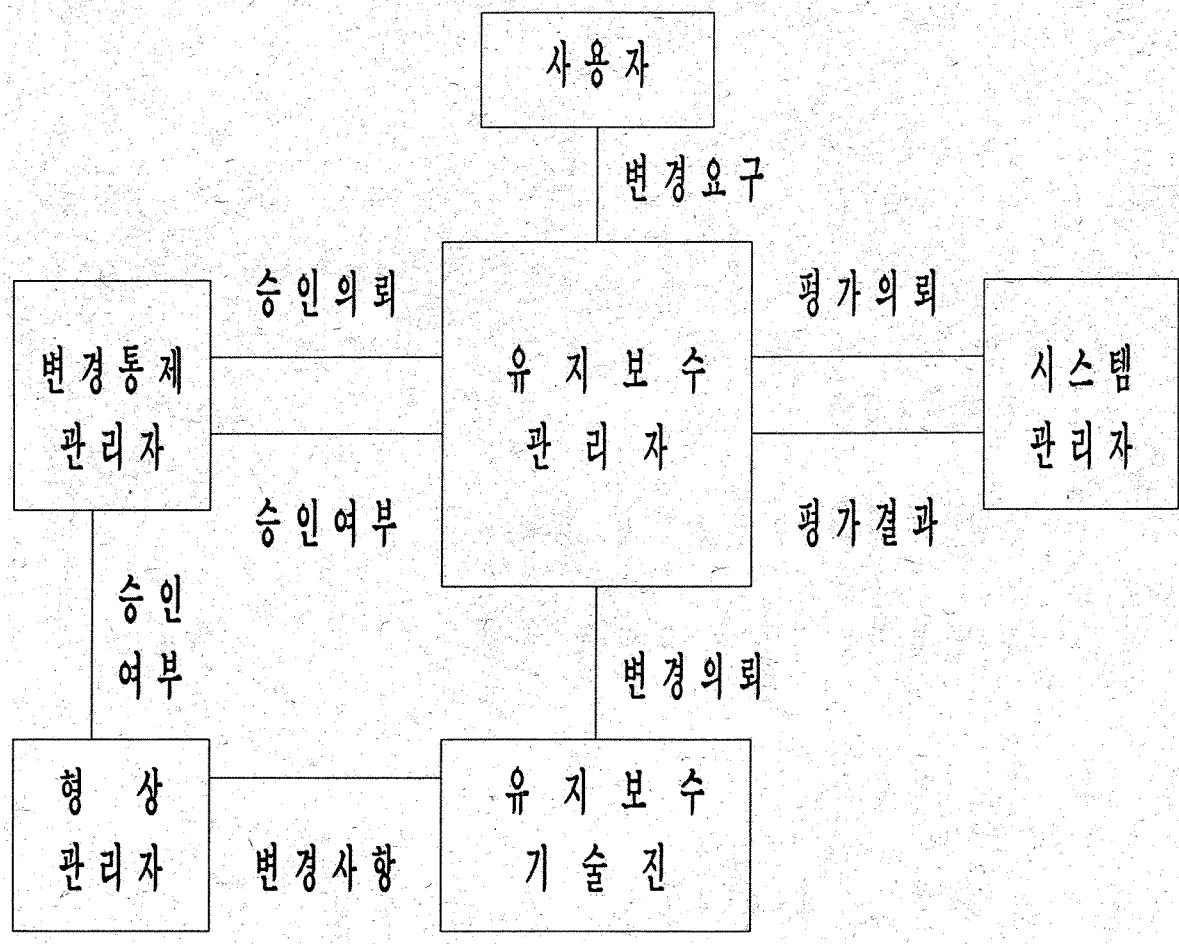
▷ 코드 검토시

- ☞ 유지보수가 용이하도록 코딩하였는가 검토
- ☞ 프로그램의 내부 문서화 검토

▷ 테스팅 종료후

- ☞ Configuration Review(형상검토) 수행

조직적인 유지보수팀 구성



조직적인 팀 구성의 효과

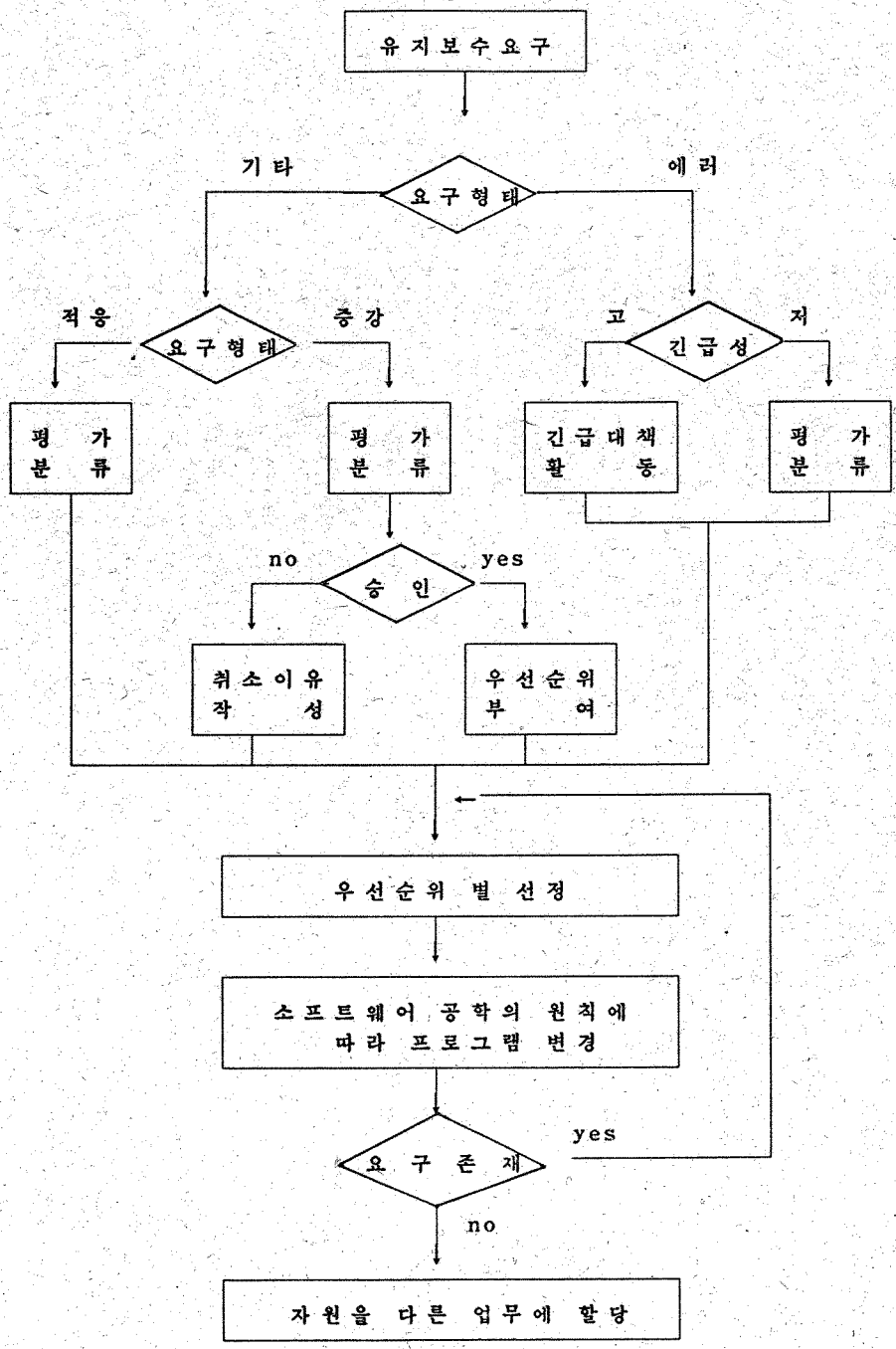
▷ 유지보수 활동의 혼란을 감소할 수 있다.

▷ 유지보수 활동의 흐름을 향상 시킨다.

▷ 적어도 한 사람은 항상 생산된 프로그램과 친숙하게 되기 때문에 변경에 대한 요구를 신속히 평가할 수 있다.

▷ 변경통제 권한자에 의해 변경통제를 수행하기 때문에 다른 여러 사용자들에게 악 영향을 미칠수 있는 변경요구를 회피할 수 있다.

유지보수 활동의 흐름


유지보수요구
요구형태
기타
에러
요구형태
적응
증강
긴급성
고
저
평가 분류
평가 분류
긴급대책 활동
평가 분류
승인
no
yes
취소이유 작성
우선순위 부여
우선순위 별 선정
소프트웨어 공학의 원칙에 따라 프로그램 변경
요구존재
yes
no
자원을 다른 업무에 할당

유지보수시 기록사항

▷ 유지보소 기법의 효율성, 프로그램의 질, 유지보수 비용의 평가를 위해 다음 사항들을 기록

- 시스템 설치 일자
- 설치후 프로그램 수행 횟수
- 프로그램 수행시 Failure의 횟수
- 시스템 변경의 내용, 정도, 형태, 일자, 담당자
- 시스템 변경으로 추가되거나 삭제된 라인수
- 각 변경시 소비된 Man-Hour(인시)
- 유지보수 종료일자
- 유지보수에 소요된 총 Man-Hour

유지보수 활동의 평가

▷ 유지보수시 기록사항을 토대로 아래 항목들을 산출함으로써 유지보수 활동을 평가

- 시스템 수행시 평균 Failure 횟수
- 유지보수 형태별로 소비된 Man-Hour
- 프로그램, 언어, 유지보수 형태별 평균 변경 횟수
- 유지보수로 한 라인을 추가 또는 삭제하기 위해 소요된 Man-Hour
- 유지보수 요구에 대한 평균 응답시간
- 유지보수 형태별 유지보수 횟수의 비율

변경으로 인한 부작용의 주요 원인

▷ 프로그램 명령문의 변경시

- 서브 프로그램, 레이블, 식별자의 삭제 또는 수정
- 성능을 증가시키기 위한 변경
- 화일의 Open 또는 Close의 수정
- 논리연산자의 변경
- 경계치 조건의 변경

▷ 데이타 변경시

- 레코드, 화일 형식, 상수의 재정의
- Global Data의 수정
- Control Flag 또는 포인터의 재초기화
- 입출력 또는 서브 프로그램 인수의 재배열

▷ 관련 기술문서를 변경하지 않으므로

Alien Code의 유지보수를 위한 지침

▷ 가능한 관련자료를 수집하여 프로그램을 연구하라

▷ 먼저 프로그램의 전반적인 제어흐름을 파악하라

▷ 확실히 이해한 부분에 대해서는 설명문을 삽입하라

▷ 컴파일러에서 제공하는 정보를 잘 이용하라

▷ 사용되지 않는 코드라고 확신이 들기 전에는 삭제하지 말아라

▷ 기존 프로그램에 존재하는 변수와 기억장소를 공유하지 말아라

▷ 유지보수 활동 및 결과를 상세히 기록하라

새로운 유지보수 전략

▷ 품질제고를 위한 유지보수

☞ 향후 요구사항을 효율적으로 수용하기 위해 과거의 시스템을 현재의 기술을 적용하여 변경

☞ 한 라인을 유지보수하는 비용이 개발비용보다 40배 정도 더 소요됨

☞ 기존 시스템의 개발 경험으로 생산성이 높음

☞ Alien Code같은 경우는 관련 문서가 창출됨

▷ Spare Parts 전략

☞ 모형화 개발방법 수행시는 시스템의 모형이 여러 수준에 걸쳐 산출되므로 현존 시스템을 이 모형중 하나로 교체하여 개발

☞ 기존 프로그램을 유지보수하는 것보다 비용절감

Software Configuration Management

▷ Software Configuration Item(소프트웨어 형상항목)

소프트웨어 개발주기의 일부 단계에서 산출되는

모든 정보를 포함하도록 구분된 항목

▷ Software Configuration(소프트웨어 형상)

소프트웨어 전 개발주기에서 산출되는

Software Configuration Item의 집합

▷ Software Configuration Management(소프트웨어 형상관리)

Software Configuration의 변경을 관리하기 위한

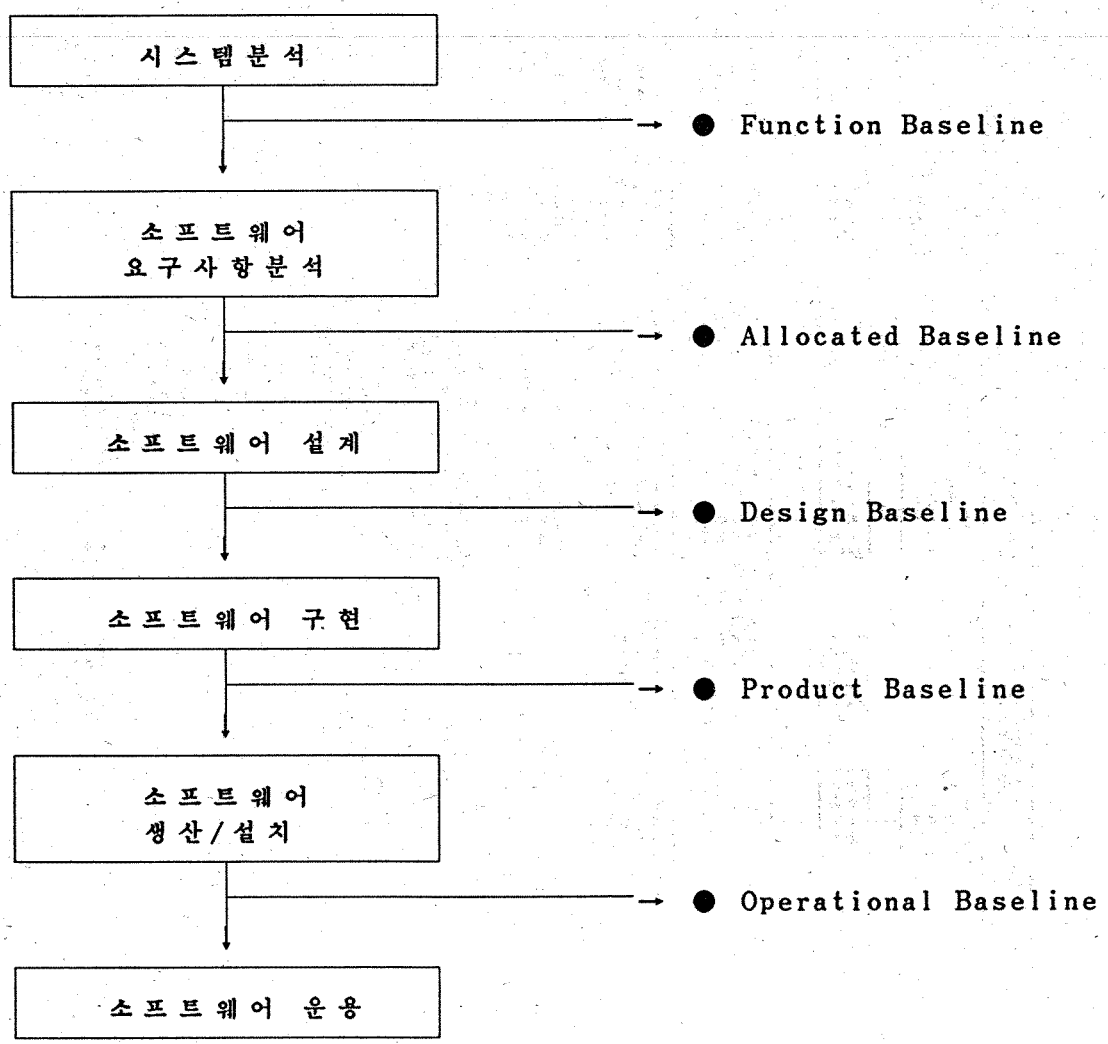
소프트웨어 공학 활동의 집합

Baseline

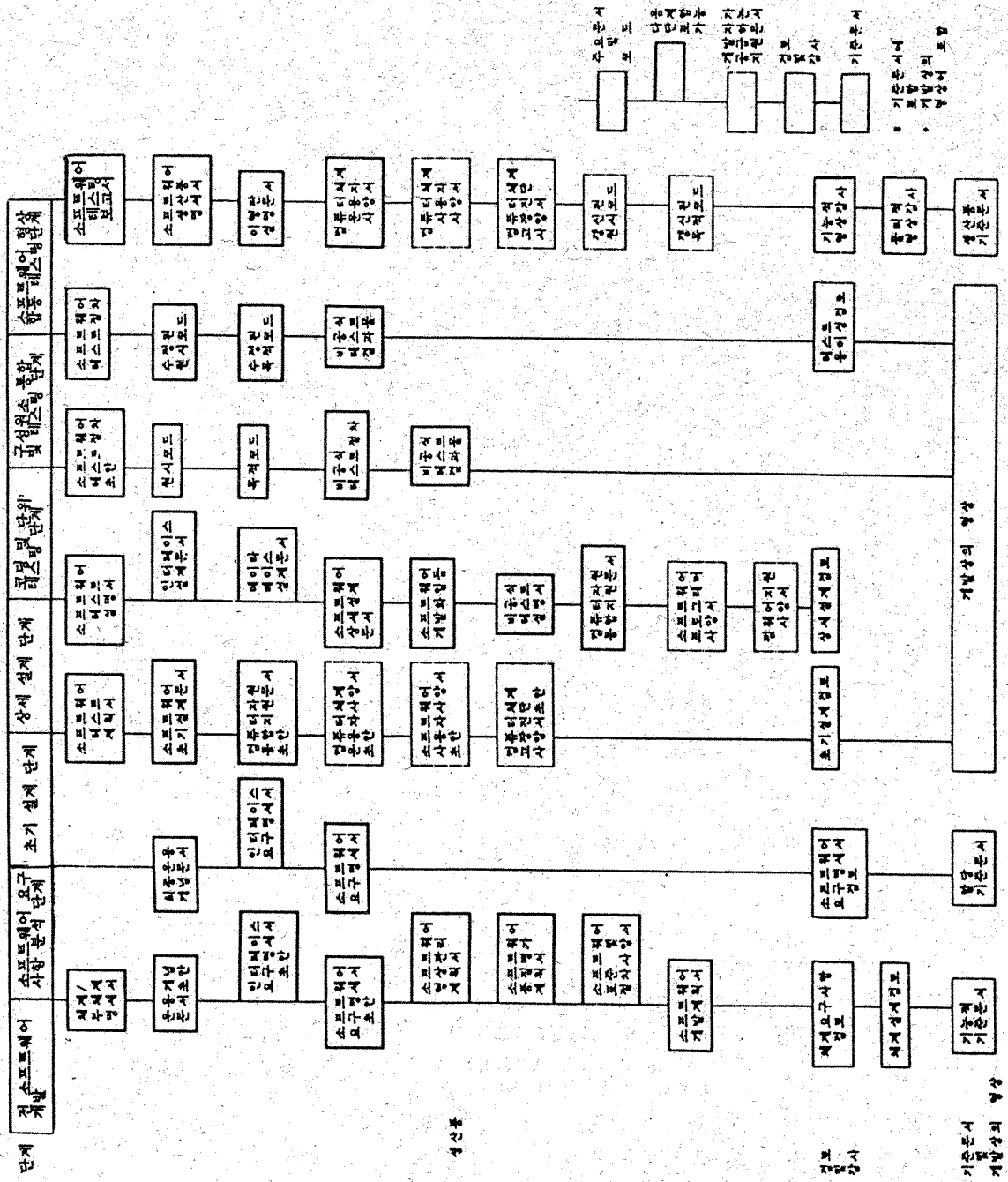
▷ Baseline(기준형상)

소프트웨어 개발주기 상에 정의된 Milestone(검증점)과

검증점에서 승인된 소프트웨어 형상항목



미 국방성 표준 소프트웨어 형상항목



형상관리 업무

▷ 형상관리는 소프트웨어 품질보증의 중요한 한 활동

▷ 형상관리의 4가지 업무

- ☞ Configuration Identification(형상식별)
- ☞ Configuration Change Control(형상변경관리)
- ☞ Configuration Audit(형상감사)
- ☞ Configuration Status Reporting(형상상태보고)

Configuration Identification

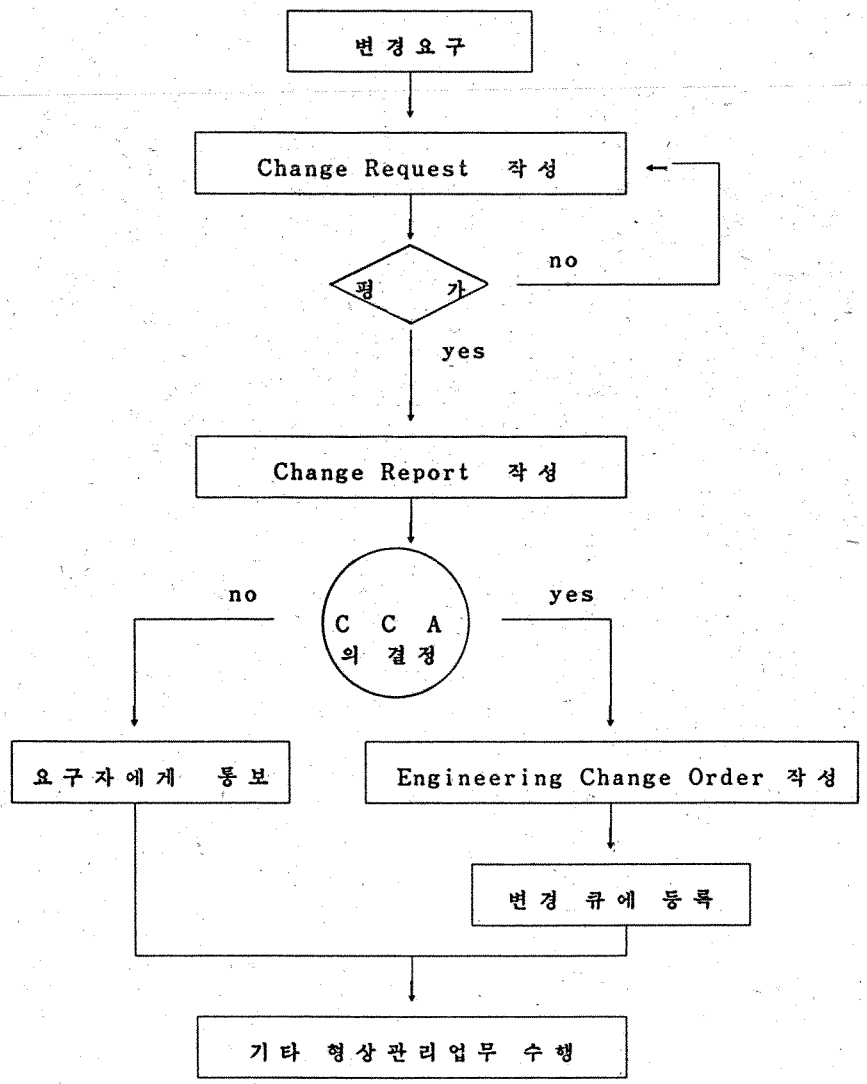
▷ 모든 소프트웨어 형상항목에 대해 의미있고 일관성 있는 명명을 보장하기 위한 형상관리 업무

▷ 기술되어야 할 명명 항목

- 소프트웨어 형상항목의 Type
- 소프트웨어 형상항목의 Name
- 프로젝트 형상항목의 Name
- 프로젝트 Identification
- Version Number
- 최종 배포일

Configuration Change Control

▷ 변경관리를 위한 메카니즘을 제공하는 가장 중요한 형상관리 업무



Configuration Audit

▷ 검토시 고려되지 않은 특성에 대해 소프트웨어 형상항목을 평가함으로써 FTR을 보완

▷ 감사 항목

☞ Engineering Change Order에 정의된 변경사항이 구현되었는가 ?

☞ 기술적정확성을 평가하기 위한 공식적 기술검토가 검토되었는가 ?

☞ 소프트웨어 공학 표준안 들을 잘 준수했는가 ?

☞ 변경사항에 대해 Configuration Identification이 수행되었는가 ?

☞ 변경사항 처리를 위해 형상관리 절차를 준수했는가 ?

☞ 다른 관련 형상항목들도 적합하게 변경되었는가 ?

Configuration Status Reporting

▷ 형상상태 보고의 중요성

- ☞ "왼손이 하는 일을 오른손이 모르는 현상"을 방지
- ☞ 둘이상의 개발자가 다른 상반된 의도를 가지고 형상항목을 수정하는 것을 방지
- ☞ 변경되기전의 형상항목을 사용하는 시간 낭비를 방지

▷ Configuration Status Report의 내용

- ☞ 변경된 사항
- ☞ 변경 수행자
- ☞ 변경 일시
- ☞ 변경으로 영향을 받을 사항