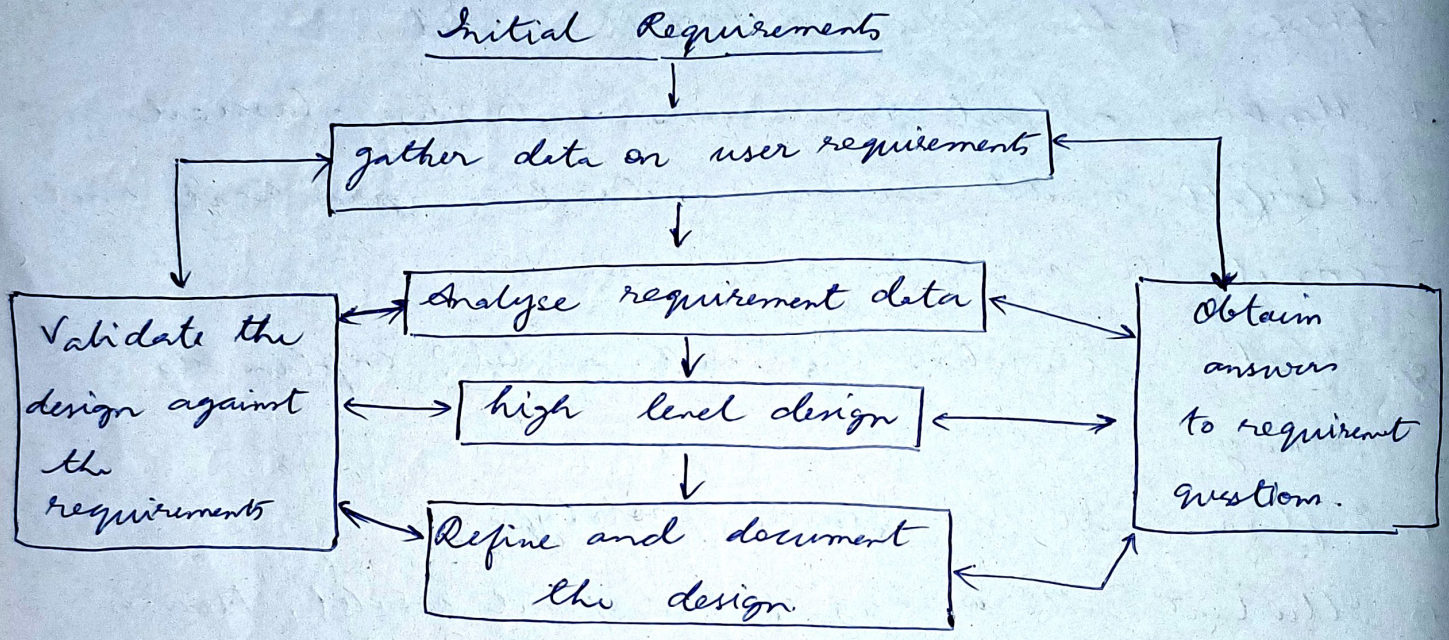
- **Software Design and Architecture**

↳ more creative than analysis

↳ problem solving activity.



Imp.

→ **Software Design:**

(i). **Conceptual Design**: (Customers)

- Written in simple language, i.e., customer understandable language.
- Detailed explanation about system characteristics
- Describes the functionality of the system.
- It is independent implementation.

- linked with requirement document.

(ii). Technical Design : (System builders / developers)

- Hardware component and design.
- Functionality and hierarchy of software components
- Software architecture.
- Network architecture
- Data structure and flow of data
- I/O components of the system
- Shows interface.

→ Advantages are required.

• Cohesion and Coupling (important w.r.t exam).

Cohesion and coupling are necessary in making any software reliable and extendable. They measure the quality of a software systems design.

→ Cohesion: It is the measure of functional strength of a module.

- In general, cohesion measures the relationship strength between the pieces of functionality within a given module in software engineering.

- The measurement leads to high cohesion and low cohesion. High cohesion implies that the modules are closely connected and changes in one module may affect the other modules.

In low cohesion, no modules are independent and changes in one module have little impact on the other modules.

- In a good module, various parts having high cohesion is preferable due to its reliability, reusability, robustness and understanding the products.

- low cohesion is associated with undesirable traits including difficulty in maintaining, reusing and understanding the product.

** (1) Explain the concept of cohesion an coupling. What are the various types of coherion an coupling?

very imp. for exams

Ans: Cohesion is the measure of the relative functional strength of the module. It is of 5 types :—

(a). coincidentally cohesive.

(b). Logically cohesion

(c). Temporal cohesion

(d). Prodedural cohesion

(e). communicational cohesion.

(a) ⟶ At the low end of the spectrum we encounter a module that performs a set of tasks that relate to each other losely. Such modules are termed as coincidentally cohesive.

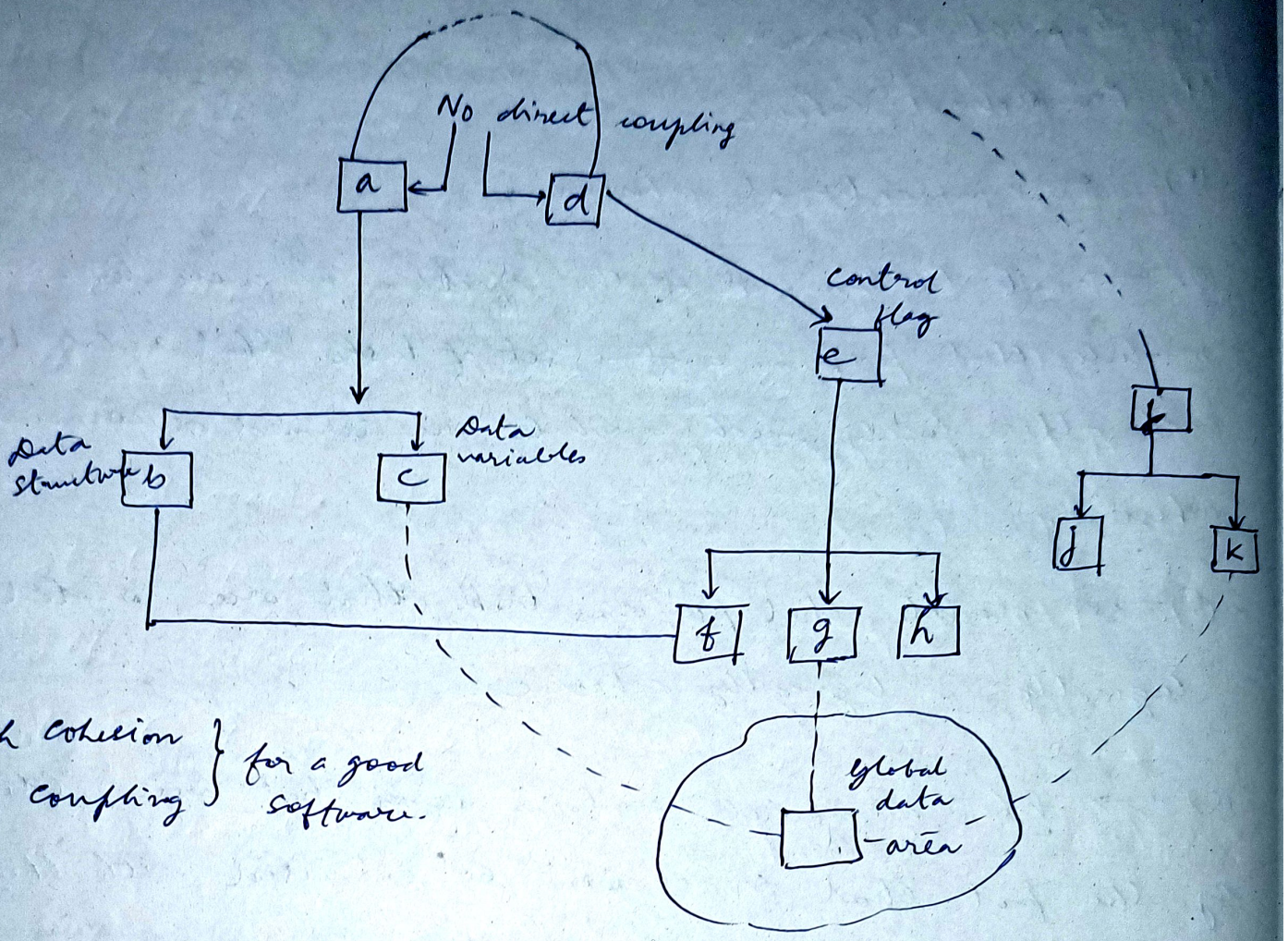
(b) ⟶ A module that performs tasks that are related logically is logically cohesive.

(c) ⟶ When a module contains tasks that are relelted by the fact that all most be executed with the same span of time, the module exhibit temporal cohesion. eg: Initialising modules at the beginning of a progeram.

(d) ⟶ When processing element of a module are related and are executed in a specific order, procedural cohesion exists. eg: do-while loops.

(e) ⟶ When all processing elements are concentrated on one area of a data structure then, communicational cohesion

exists. eg: error handling.



High cohesion
low coupling
} for a good software.

Coupling is the measure of the a relative interdependencies between the modules.

It is the measure of interconnection among modules in a software structure. In software design we strive for lowest possible coupling.

The figure shows different type of module coupling. Modules a and d are subordinate to different modules. Each is unrelated and so no direct coupling occurs.

Module c is subordinate to module a and is accessed via a conventional argument list through which data is passed.

couplings are of 6 types:

(a) Data coupling.
(b). Stand coupling
(c). control coupling.
(d). External coupling / common coupling
(e). Content coupling.
(f). compiler coupling.

type of external coupling

(a) ⟶ As long as simple argument list is present, i.e., simple data are passed, a one-to-one correspondence of items exists. In the figure, this occurs between modules 'a' and 'c'.

(b) ⟶ It is found when a portion of data structure rather than simple arguments are passed via a module interface. In fig: a and b

(c) ⟶ A control flag (ie, a variable that controls decisions in a subordinate) is passed in control coupling.
In fig: 'd' and 'c' modules.

(d) ⟶ High levels of coupling occur when modules are tied to an environment external to the software. It is essential but should be limited to a small number of modules within a structure. High coupling also occurs when a number of modules reference a global data area. In the fig; 'e', 'g' and 'k' modules.

(e) ⟶ It occurs when one module makes use of data or control information maintained within the boundary of another module. It also occurs when branches are made into the middle of a module. This mode of ~~coupl~~ coupling should be avoided. In the fig; modules 'b' and 'f'

(f) ⟶ It is another variant of external coupling. It ties source code to specific attributes of a compiler.

- **Data Flow Diagram (DFD)**

- Graphical representation of the flow of data through an information
- designed to show how a system is divided into smaller portion and to highlight the flow of data between those parts.
- DFD Symbols :

① Square : ☐

Defines a source or destination of a data.

(ii). Arrow: →

Identifies the data flow

(iii). Circle or bubble:

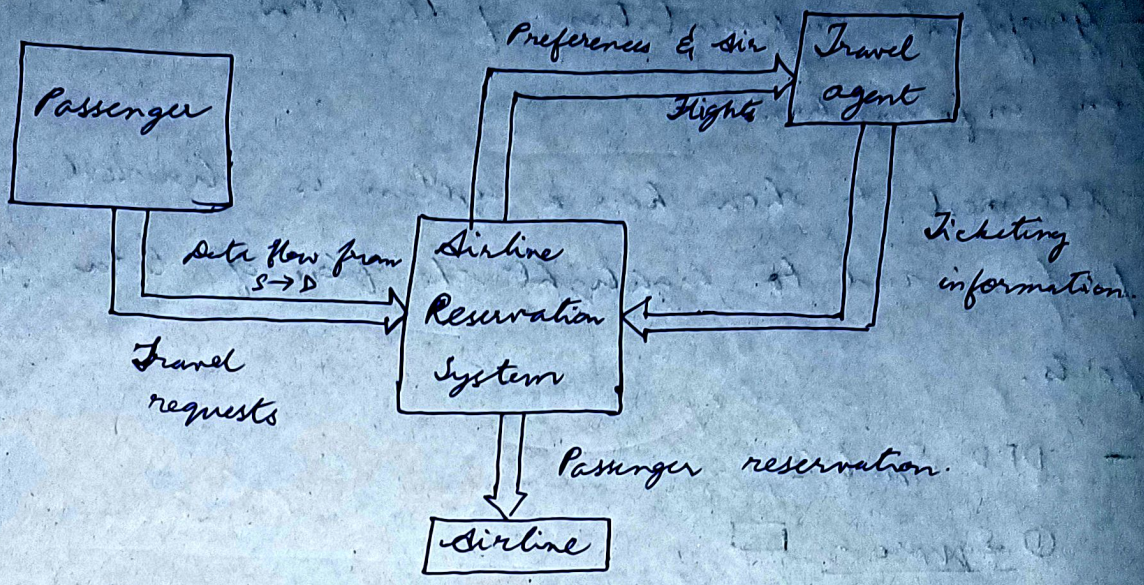
Represents process that transforms incoming data flow into outgoing data.

(iv). Open rectangle : ☐

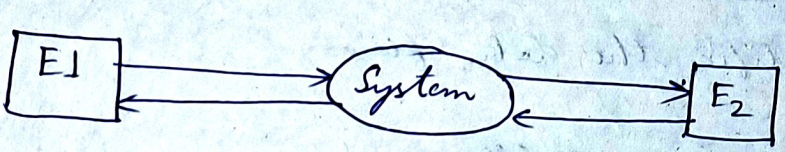
Indicates a data stored or addressed or temporary repository of data (constant data, data of rest).

- **constructing a Data Flow Diagram -**

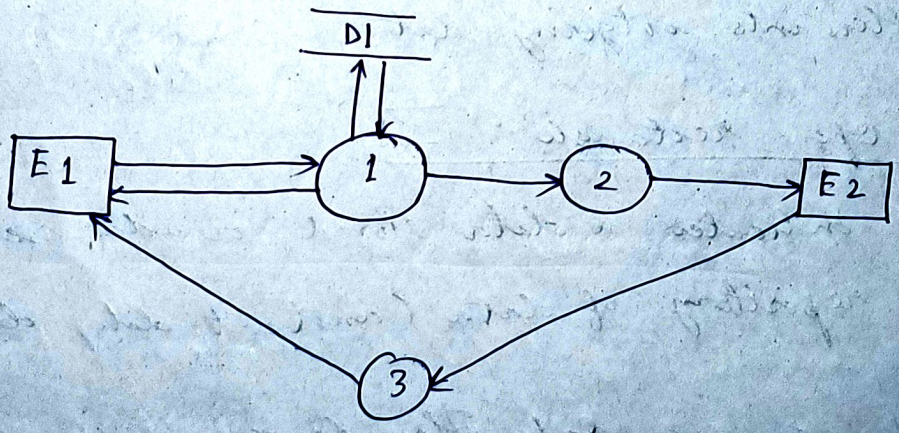
The content level flow diagram for an ~~a~~ airline reservation system



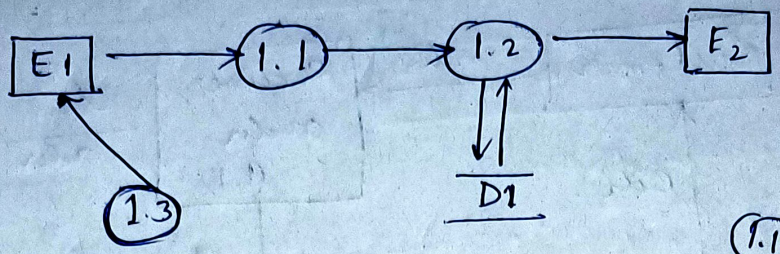
- DFD level 0:



- DFD level 1:



• DFD level 2:



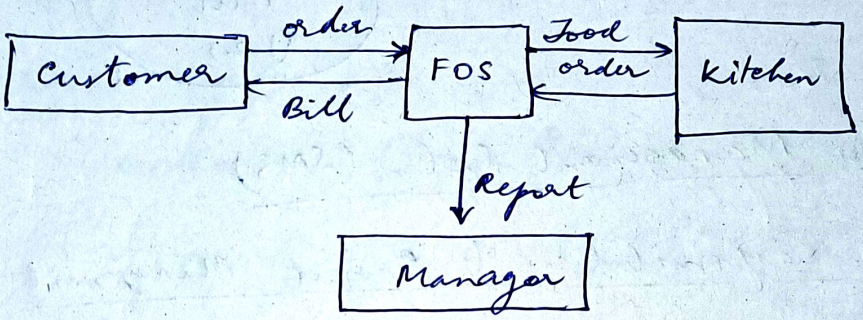
E_1, E_2 : source destination

①, ② are processes

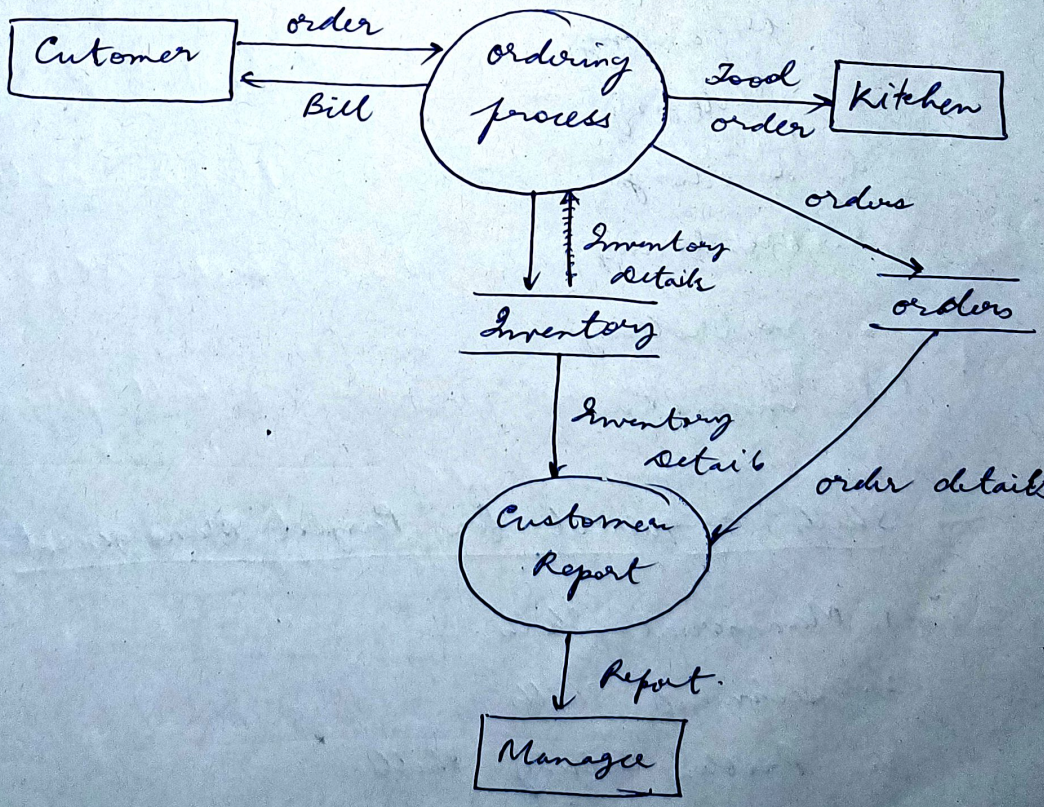
1.1, 1.2 are sub processes.

Eg: Food ~~org~~ ordering system.

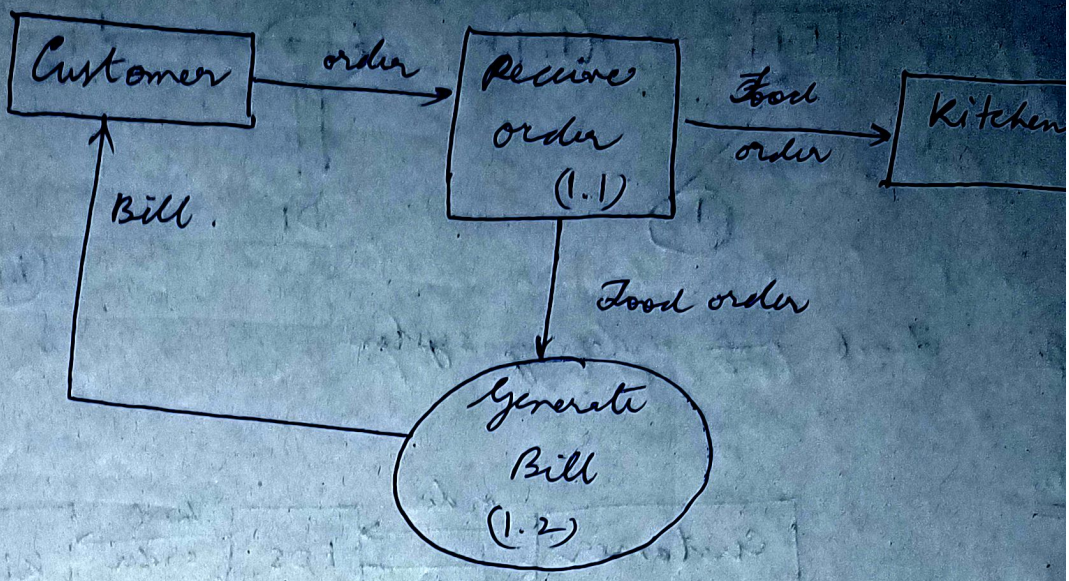
L0



L1



L2



- **Software Management System (SMS)**

— **Job responsibility of Project Management:**

1. Planning
2. Organising
3. Stuffing
4. Directing
5. Monitoring
6. controlling
7. Innovative

— **Skills required for Project Management:**

1. Managerial Skills
2. Technical Skills
3. Problem Solving Skills
4. Conception.

5. Reursive Skills.
6. communication Skills.

— **Project Planning:**

1. Estimation (cost, derivation, effort)
2. Staffing
3. Scheduling manpower and other resources.
4. Risk Management
5. Miscellaneous Plan (Quality Assurance plan, configuration and installation plan).

- **cost Estimation:**

Models

Static single variable model
(single unique variable is taken as unique variable to calculate all the other factors)

Static multivariate models → (cocomo)

— **Static single variable model**

$$C = aL^b$$

, C = cost; a, b ⇒ constant
L = size of code.

$$\text{Effort, } E = 1.4 L^{0.93}$$

$$DOC = 30.4 L^{0.90}$$

(Documentation)

$$D = 4.6 L^{0.26}$$ ⟶ duration

- **Cost estimation:**

There are no. of estimation techniques that have been developed which have some common attributes.

Common attributes:

The common attributes are → project scope*, that must be established in advance, software matrices* which are used as basics from which estimates are made.

* project is divided into small pieces which are estimated individually.

- **COCOMO:**

↳ Construction cost model

- It is a cost estimation model.
- Applies on 3 classes of software project.

(a) Organic projects (small team, good experience, less rigid requirements)

(b). Semi-detached projects (medium team, mixed experience)

(c). Embedded-projects (mixer of (a) and (b))

- **Stages of COCOMO:**

(i). Basic -

(ii). Intermediate

(iii). Detailed

(i) — **Basic COCOMO:**

- Approximate estimate of project parameters
- Software development effort is estimated using lines of code.

(E) Effort applied = $a_{b\,(basic)} (K\,LOC)^{b_b}$ $(K \rightarrow kilo)$

(D) Development time = c_b (Effort applied)^{d_b}

People required = E/D

class	a_b	b_b	c_b	d_b
Organic	2.4	1.05	2.5	0.38
Semi-detached	3	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

(ii) —— Intermediate COCOMO :

- It is an extension of basic COCOMO.
- It refines the estimate obtained by basic COCOMO

Effort applied, $E = a_i (KLOC)^{b_i}$

development time, $D = c_i (\text{Effort applied})^{d_i}$

People required, $P = E/D$

Same table, with same values except

$a_b \to a_i$, $b_b \to b_i$ —— and so on.

- It refines the estimate by using a set of 15 cost drivers.

(iii) — Detailed COCOMO:

- It refines intermediate COCOMO.
- It has 6 phases

(a). Planning ① and requirement ②.
(b). System Design ③
(c). Module code and test ④
(d). Integration and test ⑤
(e). cost construction ⑥

} Study 1 line each.

: Same formula as above and table.