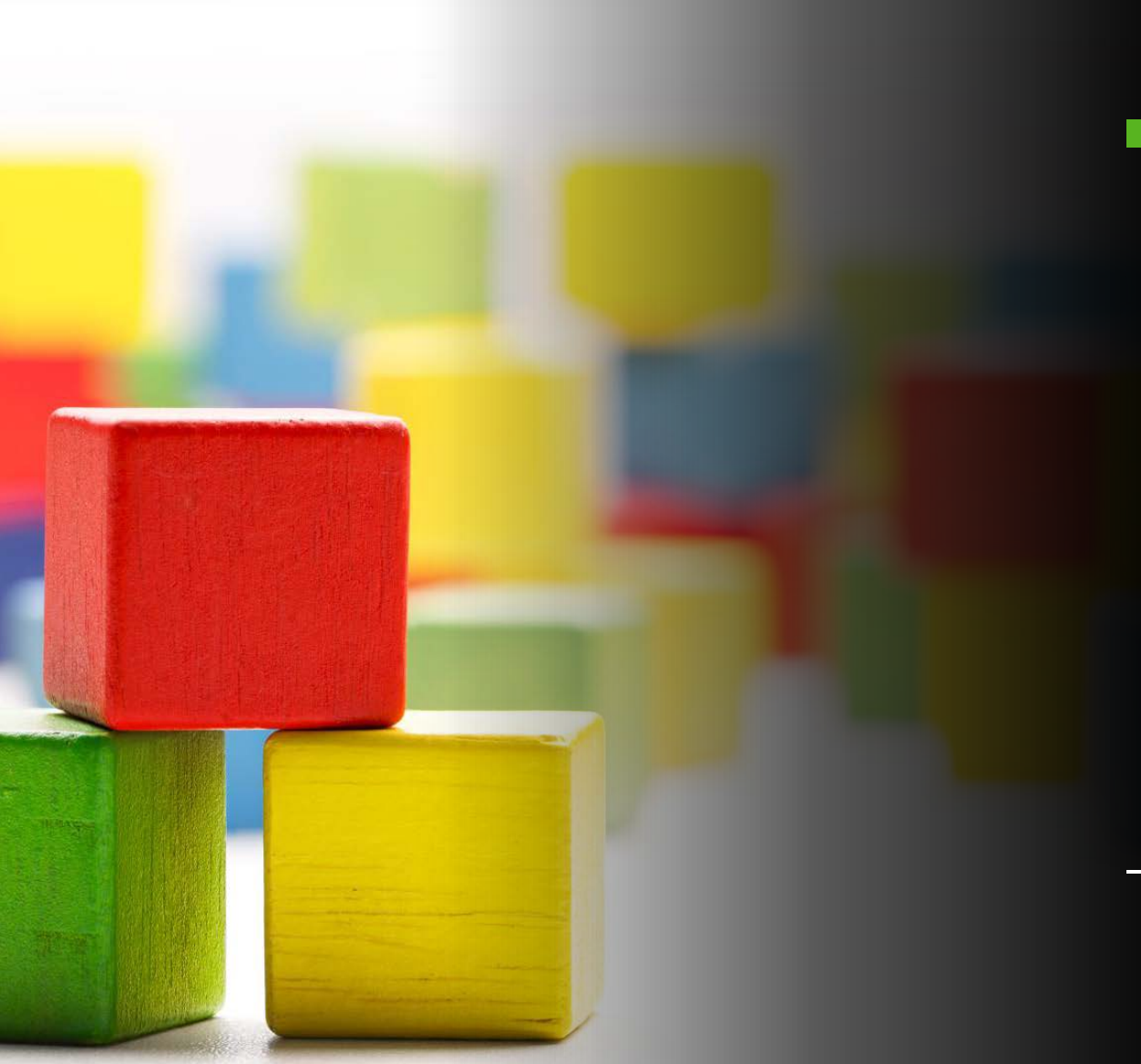


Software Engineering

Dr. Arijit Karati

Contents of the slides are prepared based on the materials from web and textbooks. It is stated that this material will be used to make the students aware of the topics and practiced for non-profit purposes.

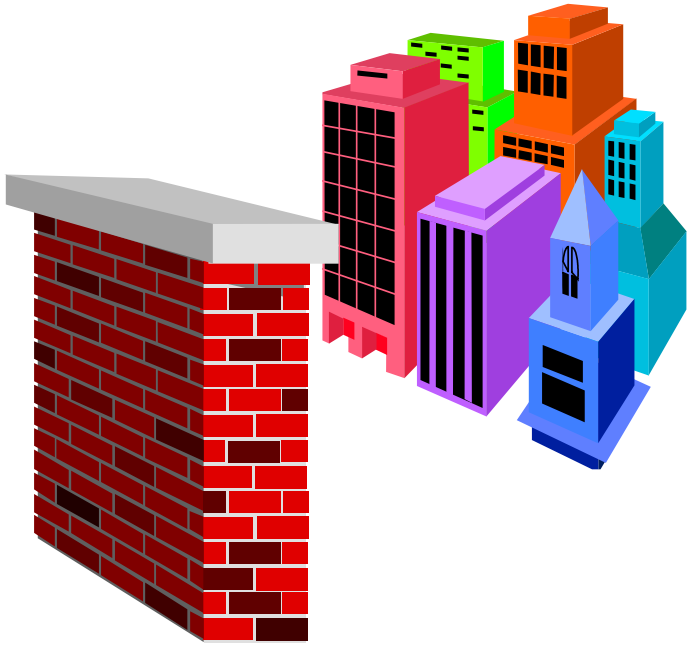
Introduction to Software Engineering

Contents...

- What is Software Engineering?
- Programs vs. Software Products
- Evolution of Software Engineering
- Notable Changes In Software Development Practices
- Introduction to Life Cycle Models
- Summary

What is Software Engineering?

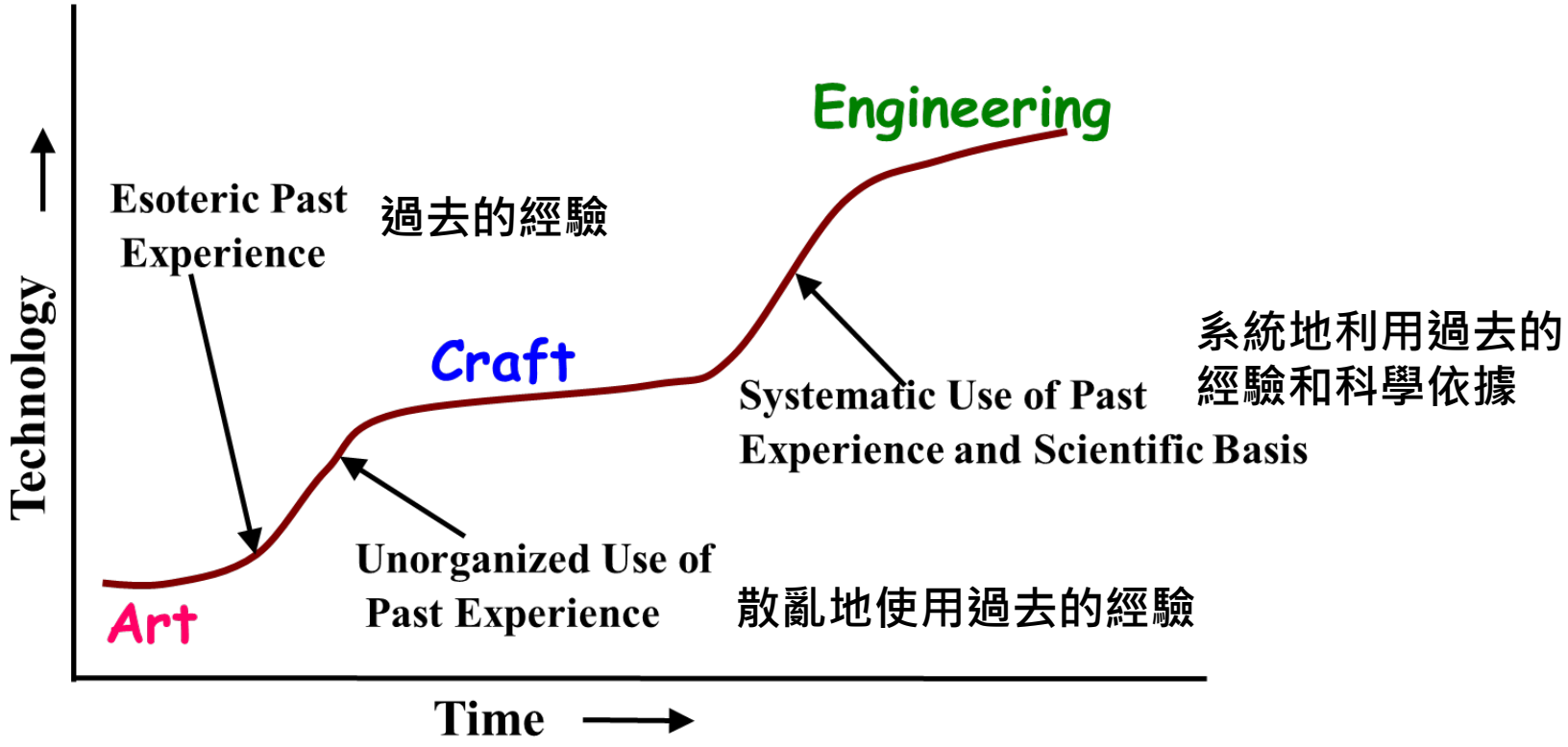
- Engineering approach to develop software.
  - ✓ Building Construction Analogy.
- Systematic collection of experience:
  - ✓ Techniques,
  - ✓ Methodologies,
  - ✓ Guidelines.



Engineering Practice

- Heavy use of past-experience:
  - ✓ Past-experience is **systematically arranged (有系統地安排)**.
- Theoretical basis and quantitative techniques provided.
- Many are just thumb rules.
- Tradeoff between alternatives.
- Pragmatic (務實) approach to cost-effectiveness (成本效益).

Technology Development Pattern

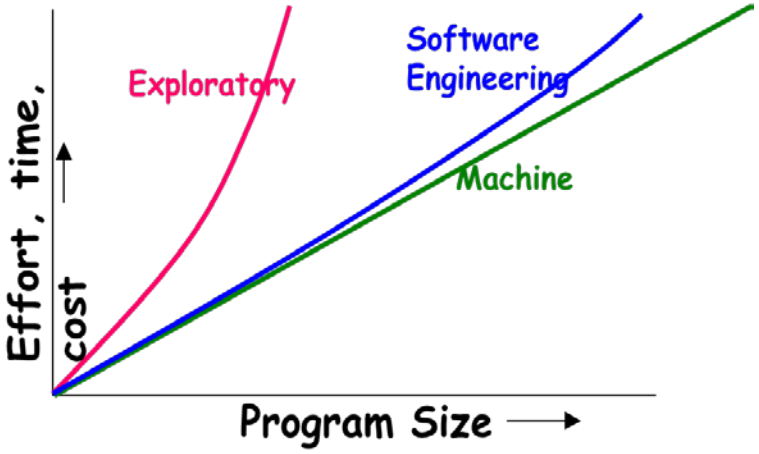


Evolution of an Art into an Engineering Discipline

- The early programmers used an exploratory (探索性的, also called build and fix) style.
- In the build and fix (exploratory) style, normally a `dirty' program is quickly developed.
- The different imperfections (瑕疵) that are subsequently noticed are fixed (固定).

What is Wrong with the Exploratory Style?

- Can successfully be used for very small programs only.
- Besides the exponential growth of effort, cost, and time with problem size:
  - ✓ Exploratory style usually results in unmaintainable (無法維持) code.
  - ✓ It becomes very difficult to use the exploratory style in a team development environment.



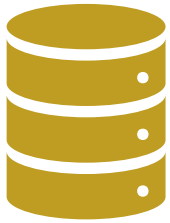
An Interpretation Based on Human Cognition Mechanism

If you are asked the question: ``If it is 10AM now, how many hours are remaining today?"

First, 10AM would be stored in the short-term memory.

Next, a day is 24 hours long would be fetched from the long-term memory into short term memory.

Finally, the mental manipulation unit would compute the difference (24-10).

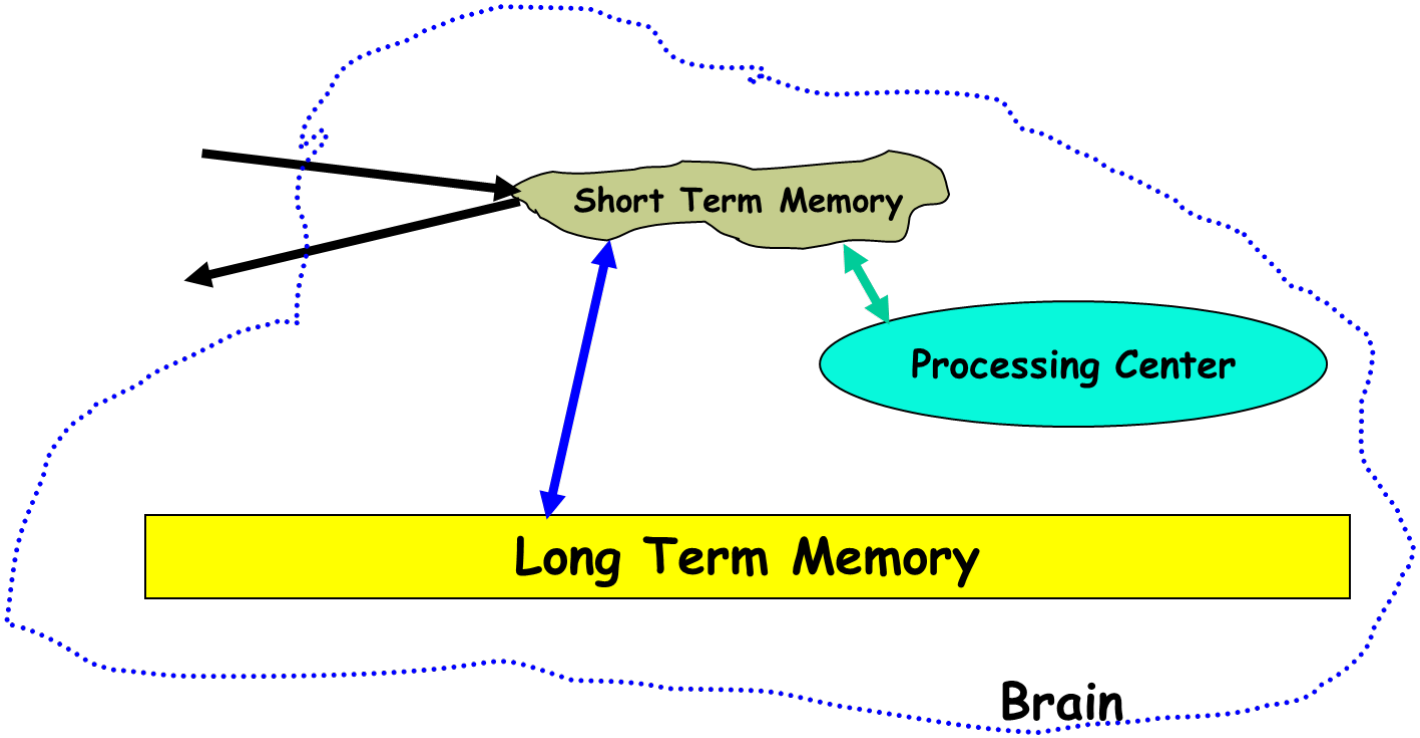


Human memory can be thought to be made up of two distinct parts [Miller 56]:

*** Short-term (短期) memory and**

*** Long-term (長期) memory.**

Human Cognition Mechanism

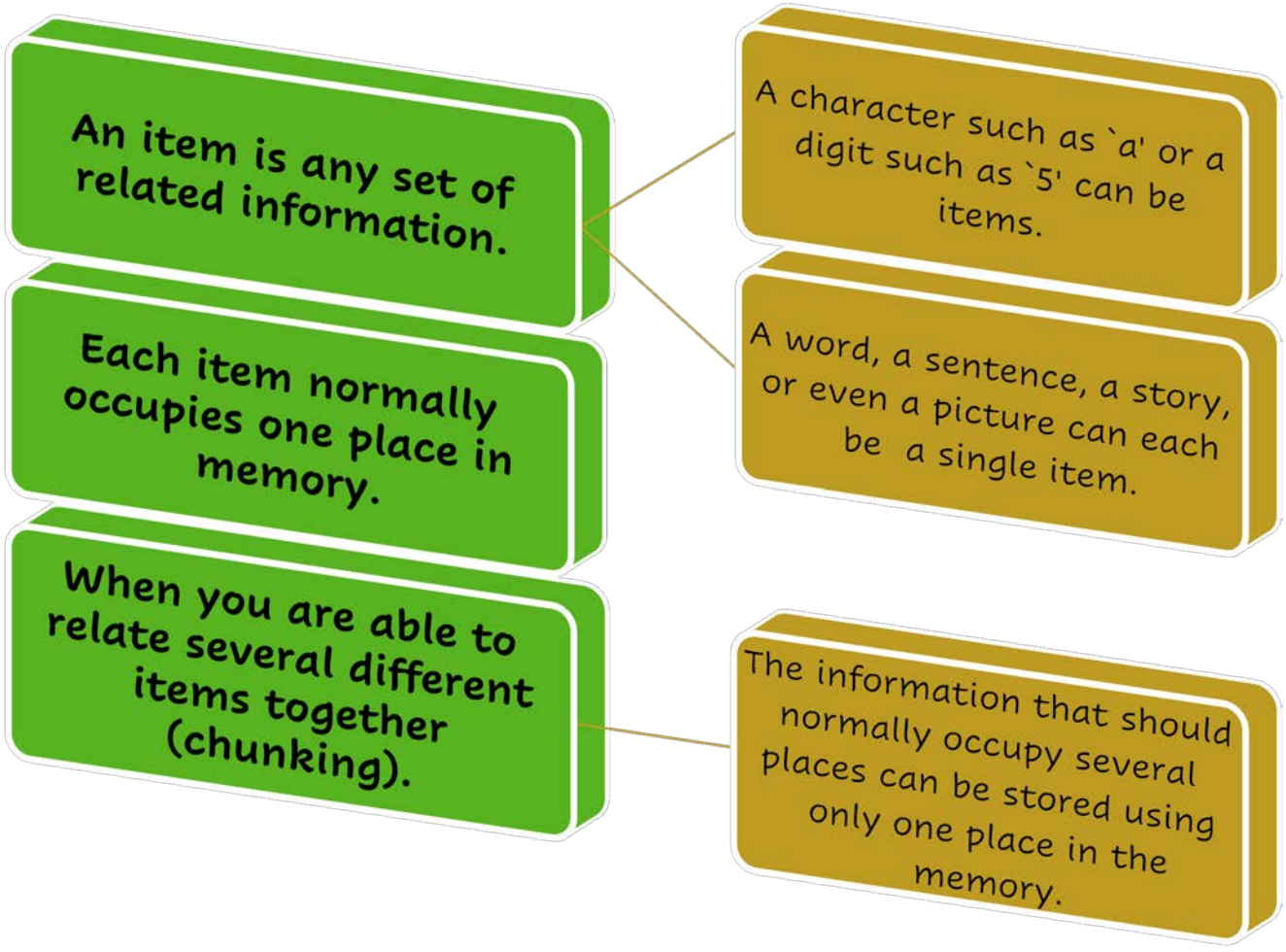


Short Term Memory

- An item stored in the short-term memory can get lost:
  - ✓ Either due to decay with time (隨著時間的流逝) or
  - ✓ Displacement (移位) by newer information.

- This restricts the time for which an item is stored in short term memory to few tens of seconds.
  - ✓ However, an item can be retained longer in the short-term memory by recycling (回收利用).

What is an Item?



Chunking

- If you are given the binary number 110010101001
  - ✓ It may prove very hard for you to understand and remember.
  - ✓ But, the octal form of 6251 (i.e. (110)(010)(101)(001)) would be easier.
  - ✓ You have managed to create chunks of three items each (您已經成功為三個項目創建了各自的塊).

Evidence of Short-Term Memory

Short term memory is evident:	•In many of our day-to-day experiences.
Suppose, you look up a number from the telephone directory and start dialing it.	• If you find the number to be busy, you can dial the number again after a few seconds almost effortlessly without having to look up the directory.
But, after several days:	• You may not remember the number at all and would need to consult the directory again.

The Magical Number 7

If a person deals with seven or less number items:

These would be easily be accommodated (容納) in the short-term (短期) memory. So, he can easily understand it.

As the number of new information increases beyond seven (超過七),

It becomes exceedingly difficult to understand (難以理解) it.

Implication in Program Development

A small program having just a few variables:

Is within the easy grasp of an individual.

As the number of independent variables in the program increases:

It quickly exceeds the grasping power of an individual:

- Requires an unduly large effort to master the problem.

Implication in Program Development

Instead of a human, if a machine could be writing (generating) a program,

The slope of the curve would be linear.

But, why does the effort-size curve become almost linear when software engineering principles are deployed?

Software engineering principles extensively use techniques specifically to overcome the human cognitive limitations.

Principles to Overcome Human Cognitive Limitations

- Two (二) important principles deployed by Software are:
  - ✓ Abstraction (抽象化)
  - ✓ Decomposition (分解)

Abstraction

Simplify a problem by omitting unnecessary details (省略不必要的細節).

Focus attention on only one aspect of the problem and ignore irrelevant details.

Suppose you are asked to develop an overall understanding of some country.

No one in his right mind would meet all the citizens of the country, visit every house, and examine every tree of the country, etc.

You would possibly refer to various types of maps for that country.

A map, in fact, is an abstract representation of a country .

實際上，地圖是一個國家抽象的表示方式

Decomposition

- A problem into many small independent parts.
  - ✓ The small parts are then taken up one by one and solved separately (然後將小零件一個接一個地分開解決).
  - ✓ The idea is that each small part would be easy (一小部分會很容易) to grasp and can be easily solved (輕鬆解決).
  - ✓ The full problem is solved when all the parts are solved (解決了所有零件後，完整的問題就解決了).

Decomposition (Example)

A popular way to demonstrate the decomposition principle:

- Try to break a bunch of sticks tied together (一堆棍子綁在一起) versus breaking them individually (各自擊破).

Example use of decomposition principle:

- You understand a book better when the contents are organized into independent chapters
- Compared to when everything is mixed up.

Why Study Software Engineering?

To acquire skills to develop large programs.

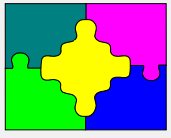
Exponential growth in complexity and difficulty level with size.

The ad hoc approach breaks down when size of software increases.

Ability to solve complex programming problems:

How to break large projects into smaller and manageable parts?

How to use abstraction?



Also learn techniques of:

Specification, design, user interface development, testing, project management, etc.

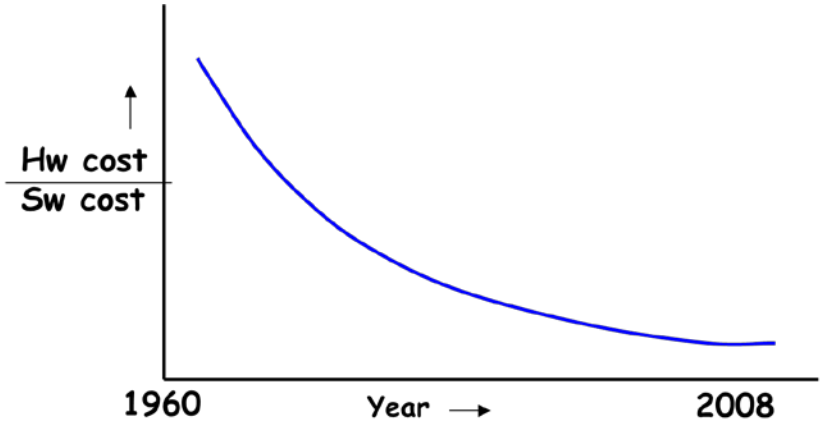
To acquire skills to be a better programmer:

Higher Productivity

Better Quality Programs

Software Crisis

- Software products:
  - ✓ Difficult to alter, debug, and enhance.
  - ✓ Fail to meet user requirements.
  - ✓ Frequently crash.
  - ✓ Expensive.
  - ✓ Often delivered late.
  - ✓ Use resources non-optimally.



Relative Cost of Hardware and Software

Factors Contributing to the Software Crisis (導致軟體危機的因素)

Larger problems (更大的問題)

Lack of adequate training (缺乏足夠的培訓) in software engineering

Increasing skill shortage (技能短缺加劇)

Low productivity improvements

Difference between Programs and Software Products (程式與軟體產品)

Program	Software
Usually small in size	Large
Author himself is sole user	Large number of users
Single developer	Team of developers
Lacks proper user interface	Well-designed interface
Lacks proper documentation	Well documented & user-manual prepared
Ad hoc development	Systematic development

Types of Software Projects

Builds for software products

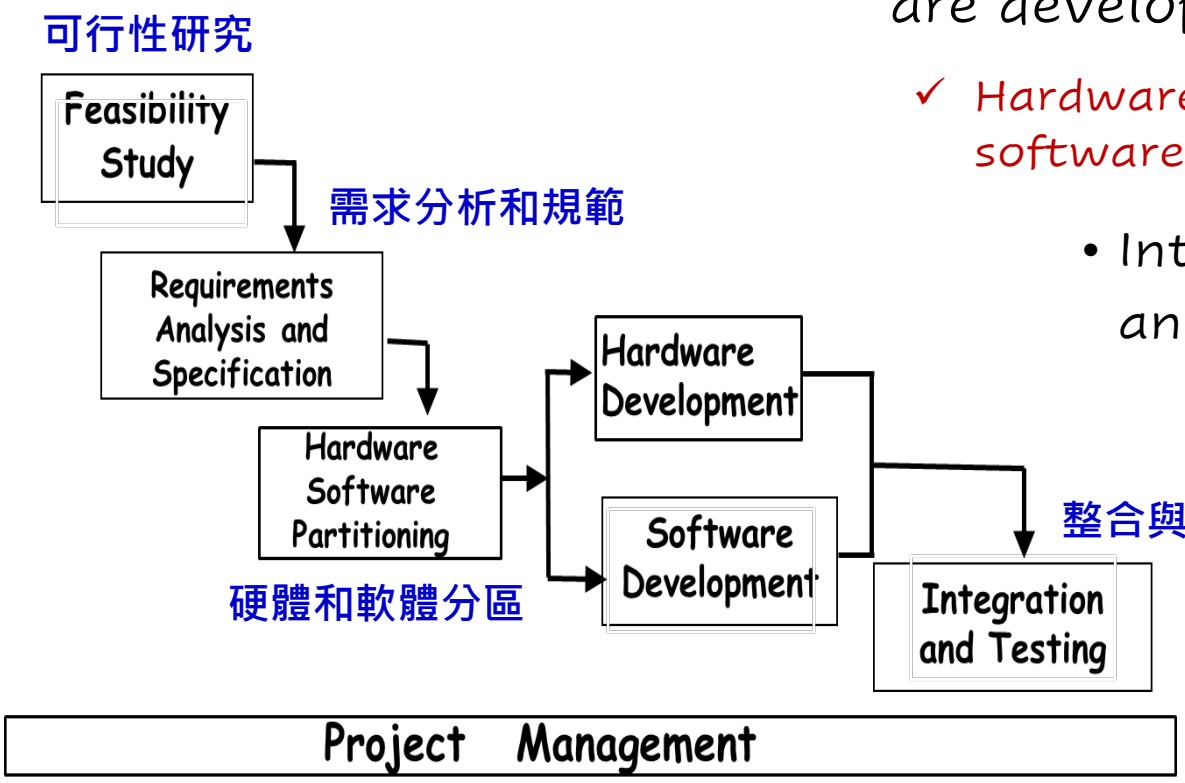
Different outsourced projects

Many MNC companies have focused on outsourced projects.

Computer Systems Engineering

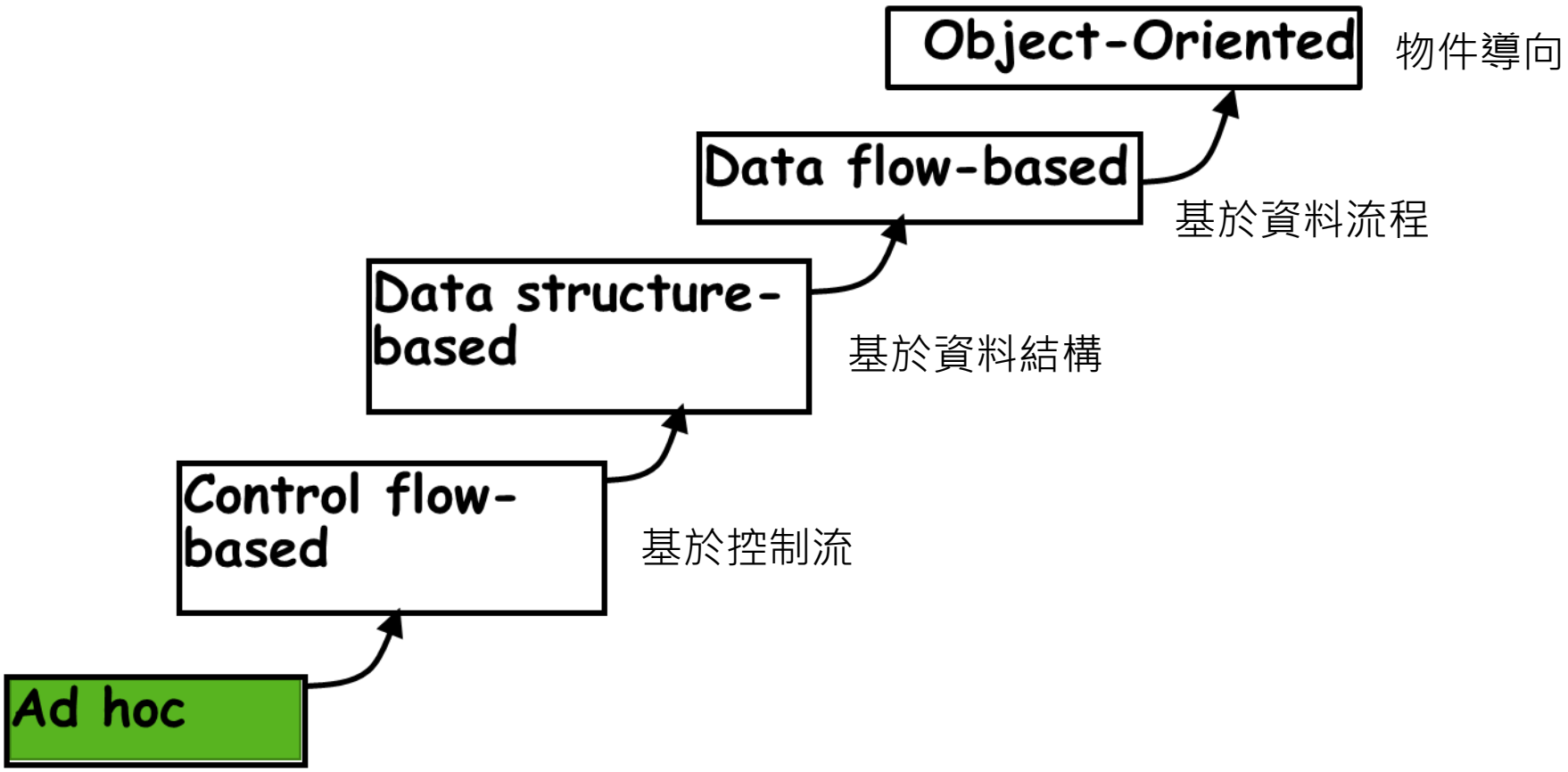
- Computer systems engineering:
  - ✓ encompasses software engineering.
- Many products require development of software as well as specific hardware to run it:
  - ✓ a coffee vending machine,
  - ✓ a mobile communication product, etc.
- The high-level problem:
  - ✓ Deciding which tasks are to be solved by software.
  - ✓ Which ones by hardware.

Computer Systems Engineering (Cont.)

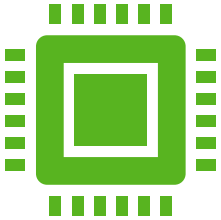


- Often, hardware and software are developed together:
  - ✓ Hardware simulator is used during software development.
    - Integration of hardware and software.
      - Final system testing

Evolution of Design Techniques



Emergence of Software Engineering



Early Computer Programming (1950s):

Programs were being written in assembly language.

Programs were limited to about a few hundreds of lines of assembly code.

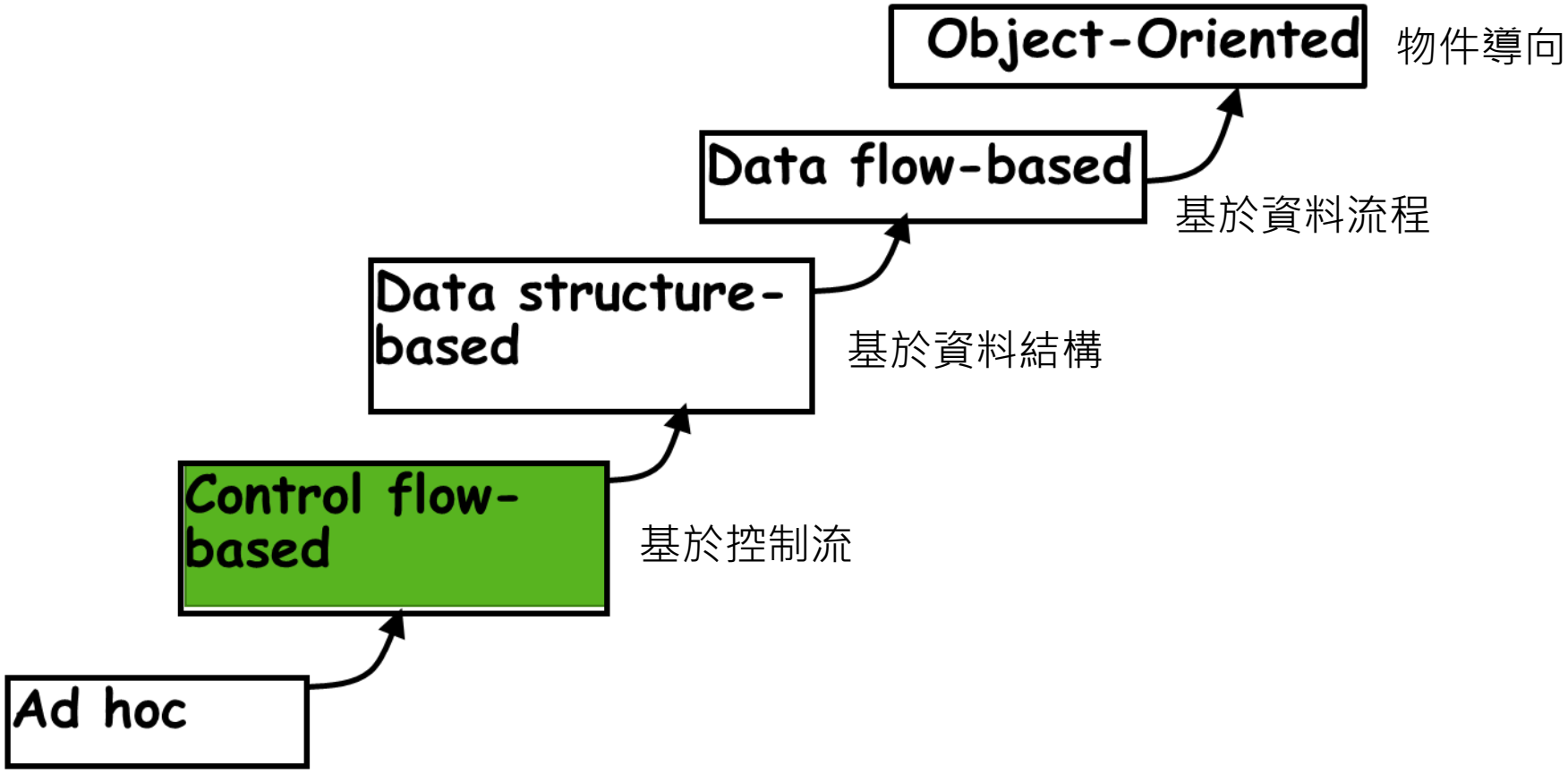
Every programmer developed his own style of writing programs:

According to his intuition (exploratory programming-探索性程式設計).

High-Level Language Programming
[Early 60s]

- High-level languages such as FORTRAN, ALGOL, and COBOL were introduced:
  - ✓ This reduced software development efforts greatly.
- Software development style was still exploratory.
  - ✓ Typical program sizes were limited to a few thousands of lines of source code.

Evolution of Design Techniques



Control Flow-Based Design
[Late 60s]

- Size and complexity of programs increased further:
  - ✓ Exploratory programming style proved to be insufficient.
- Programmers found:
  - ✓ Very difficult to write cost-effective and correct programs (難以編寫具有成本效益的正確程式).
  - ✓ programs written by others (別人寫的) very difficult to understand and maintain.
- To cope up (應對) with this problem, experienced programmers advised: ``Pay attention to the design of the program's control structure (特別注意程式控制結構的設計).''

Control Flow-Based Design (Cont.)
[Late 60s]

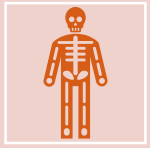
- A program's control structure indicates:
  - ✓ The sequence in which the program's instructions are executed.
- To help design programs having good control structure:
  - ✓ Flow charting technique (流程圖技術) was developed.
- Using flow charting technique:
  - ✓ One can represent & design a program's control structure.
  - ✓ Usually one understands a program:
    - ❑ By mentally simulating program's execution sequence (執行順序).

Control Flow-Based Design (Cont.)
[Late 60s]

- Program having a messy flow chart (混亂的流程圖) representation:
  - ✓ Difficult to understand and debug (調試).
- It was found:
  - ✓ GO TO statements makes control structure of a program messy.
  - ✓ GO TO statements alter (改變) the flow of control arbitrarily.
  - ✓ The need to restrict use of GO TO statements was recognized.
- Many programmers had extensively (廣泛地) used assembly languages.
  - ✓ JUMP instructions are frequently used for program branching in assembly languages.
  - ✓ Programmers considered use of GO TO statements inevitable.

Control Flow-Based Design (Cont.)

[Late 60s]



At that time, Dijkstra published his article:

"**Goto Statement Considered Harmful**" Comm. of ACM, 1969.

Many programmers were unhappy to read his article (許多工程師很不喜歡讀他的文章).

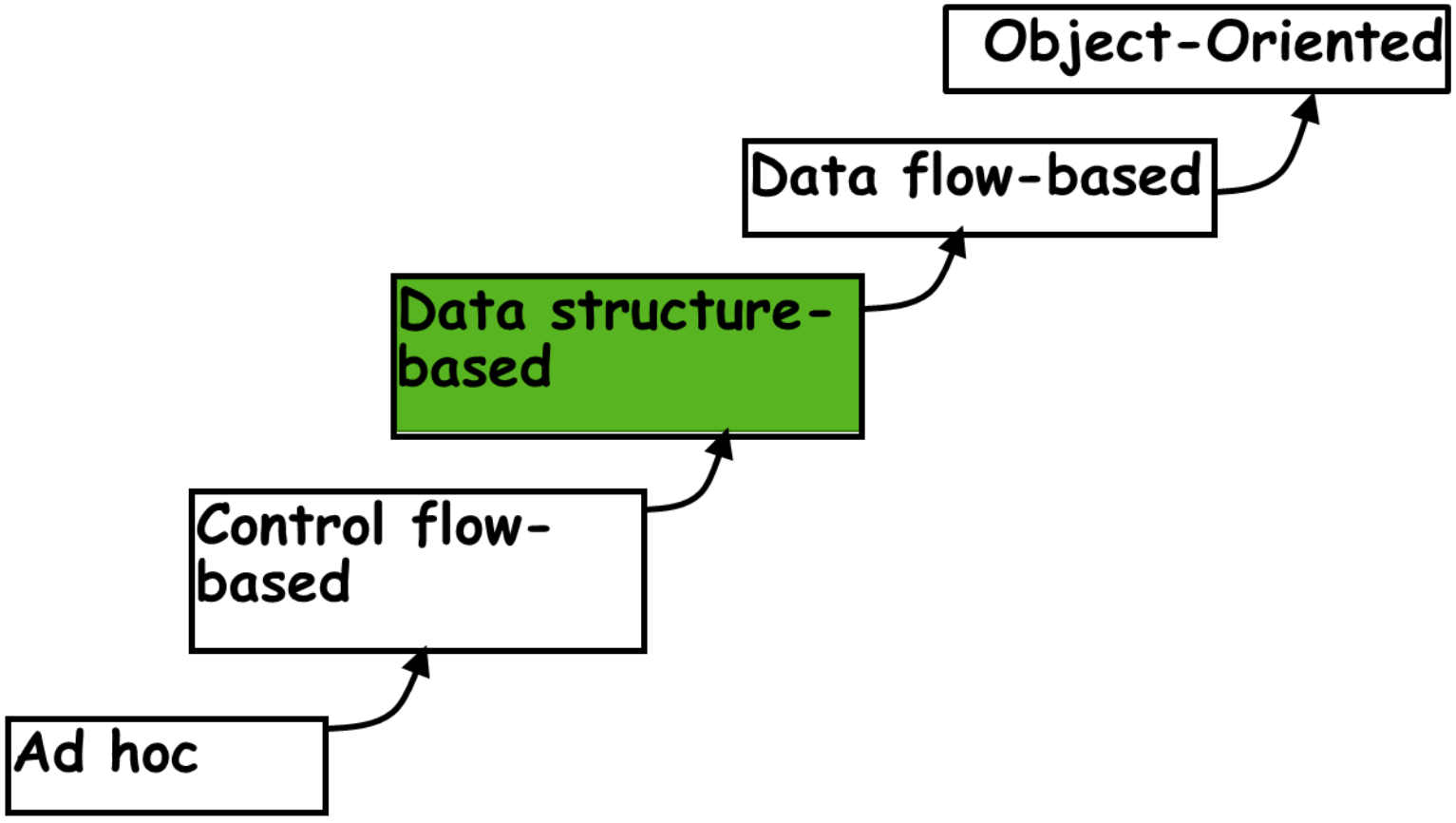
They published several counter articles:

Highlighting the advantages and inevitability of GO TO statements.

Control Flow-Based Design (Cont.)
[Late 60s]

- But, soon it was conclusively proved:
  - ✓ Only three programming constructs are enough to express any programming logic (三種程式設計風格足以表達任何邏輯):
    - ❑sequence (e.g. a=0;b=5;)
    - ❑selection (e.g. If(c=true) k=5 else m=5;)
    - ❑iteration (e.g. while(k>0) k=j-k;)
- Everyone accepted:
  - ✓ It is possible to solve any programming problem without using GO TO statements.
  - ✓ This formed the basis (奠定了基礎) of Structured Programming methodology.

Evolution of Design Techniques



Structured Programming

A program is called **structured**

When it uses only the following types of constructs:

- sequence,
- selection,
- iteration

Unstructured control flows are avoided.

Consist of a neat set of modules (模組).

Use single-entry, single-exit program constructs.

Structured Programming (Cont.)

However, violations to this feature are permitted:

* Due to practical considerations such as:
  - Premature loop exit to support exception handling.

Structured programs are:

* Easier to read and understand,
* Easier to maintain (易於維護),
* Require less effort and time (更少的精力和時間) for development.

Research experience shows:

* Programmers commit a smaller number of errors:
  - While using structured **if-then-else** and **do-while** statements.
  - Compared to **test-and-branch** constructs.

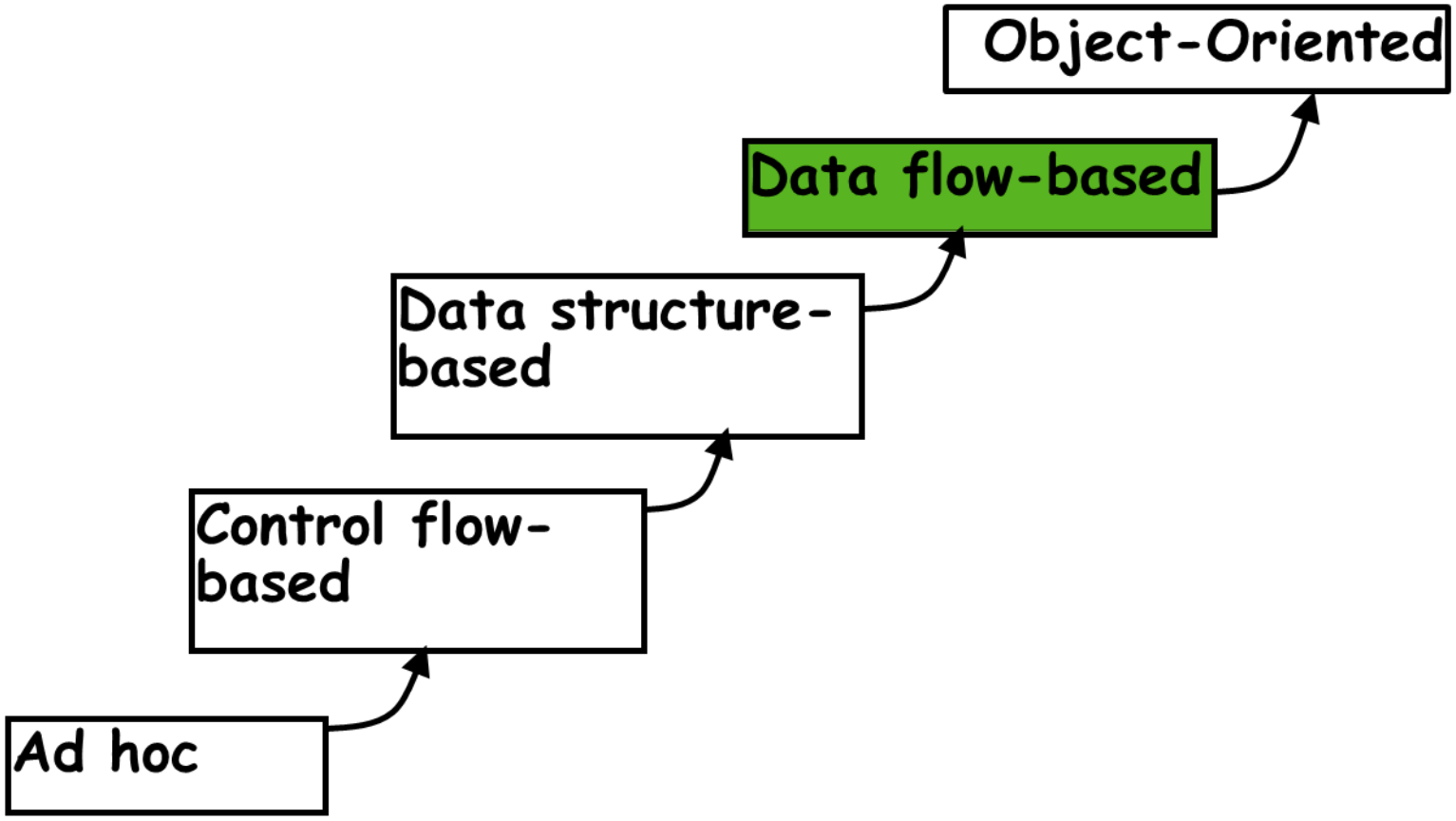
Data Structure-Oriented Design
[Early 70s]

- Soon it was discovered:
  - ✓ It is important to pay more attention to the design of data structures of a program
    - ❑ Than to the design of its control structure.
- Techniques which emphasize designing the data structure:
  - ✓ Derive program structure from it:
    - ❑ Are called data structure-oriented design techniques.

Data Structure-Oriented Design (Cont.)
[Early 70s]

- Example of data structure-oriented design technique:
  - ✓ Jackson's Structured Programming(JSP) methodology
    - ❑ Developed by Michael Jackson in 1970s.
- JSP technique:
  - ✓ Program code structure should correspond to the data structure.
- In JSP methodology:
  - ✓ A program's data structures are first designed using notations for
    - ❑sequence, selection, and iteration.
  - ✓ Then data structure design is used :
    - ❑To derive the program structure.

Evolution of Design Techniques



Data Flow-Oriented Design

[Late 70s]

- Data flow-oriented techniques advocate:
  - ✓ The data items input to a system must first be identified (必須先確定).
  - ✓ Processing required on the data items to produce the required outputs should be determined.

- Data flow technique identifies:
  - ✓ Different processing stations (functions) in a system.
  - ✓ The items (data) that flow between processing stations.

Data Flow-Oriented Design (Cont.)

[Late 70s]

- Data flow technique is a generic technique (通用技術):
  - ✓ Can be used to model the working of any system.
    - ❑ not just software systems (不只是軟體系統).
- A major advantage of the data flow technique is its **simplicity (簡單)**.

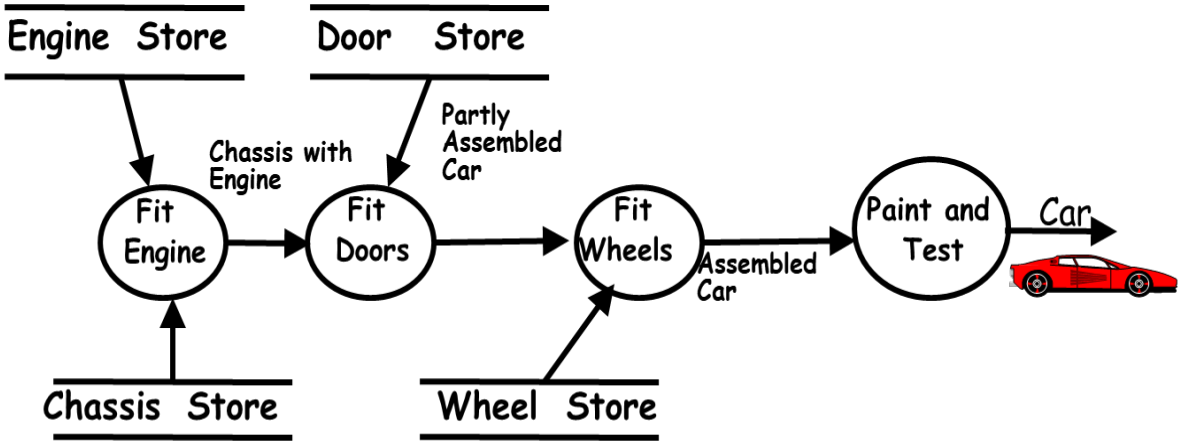
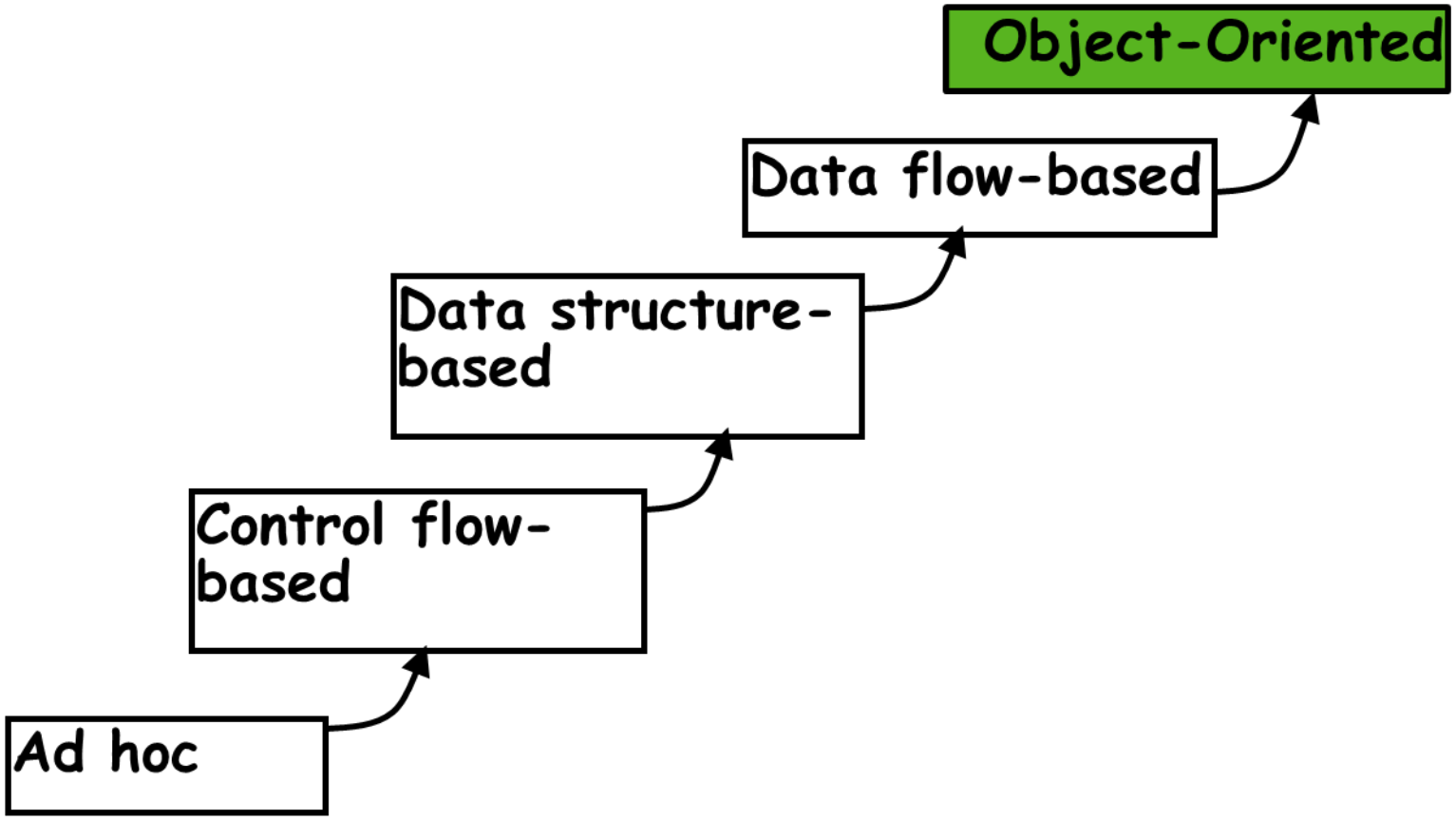


Fig: Data Flow Model of a Car Assembly Unit

Evolution of Design Techniques



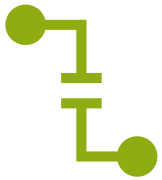
Object-Oriented Design

[80s]

- Object-oriented technique:
  - ✓ An intuitively appealing (吸引人的) design approach:
  - ✓ Natural objects (such as employees, pay-roll-register, etc.) occurring in a problem are first identified.
- Relationships among objects:
  - ✓ Such as composition, reference, and inheritance are determined.

Object-Oriented Design (cont.)

[80s]



Each object (每個物件) essentially acts as

A data hiding (or data abstraction, 資料抽象) entity.

Object-Oriented Techniques have gained wide acceptance:

* Simplicity
* Reuse possibilities
* Lower development time & cost
* More robust code
* Easy maintenance

Evolution of Other Software Engineering Techniques

- The improvements to the software design methodologies
  - ✓ are indeed very conspicuous (顯眼的).
- In additions to the software design techniques:

  Several other techniques evolved:
  - ✓ life cycle models,
  - ✓ specification techniques,
  - ✓ project management techniques,
  - ✓ testing techniques,
  - ✓ debugging techniques,
  - ✓ quality assurance techniques,
  - ✓ software measurement techniques,
  - ✓ CASE tools, etc.

Exploratory style vs Modern software development practices

- Use of Life Cycle Models
- Software is developed through several well-defined stages:
  - ✓ requirements analysis and specification,
  - ✓ design,
  - ✓ coding,
  - ✓ testing, etc.
- Emphasis has shifted
  - ✓ from error correction (糾錯) to error prevention (錯誤預防).
- Modern practices emphasize:
  - ✓ detection of errors as close to their point of introduction as possible.

Exploratory style vs Modern software development practices (cont.)

- In exploratory style,
  - ✓ errors are detected only during testing (在測試期間),
  - ✓ coding is synonymous with program development.

- Now,
  - ✓ focus is on detecting as many errors as possible in each phase of development (在開發的每個階段檢測錯誤).
  - ✓ coding is considered only a small part of program development effort.

Exploratory style vs Modern software development practices (cont.)

A lot of effort and attention is now being paid to:	Requirements specification.
Also, now there is a distinct design phase:	Standard design techniques are being used.
During all stages of development process:	Periodic reviews are being carried out
Software testing has become systematic:	Standard testing techniques are available.

Exploratory style vs Modern software development practices (cont.)

- During all stages of development process:
  - ✓ Periodic reviews (定期審查) are being carried out
- Software testing has become systematic:
  - ✓ Standard testing techniques are available.
- There is better visibility of design and code:
  - ✓ Visibility means production of good quality, consistent (一致的) and standard documents.
  - ✓ In the past, very little attention was being given to producing good quality and consistent documents.
  - ✓ We will see later that increased visibility makes software project management easier.

Exploratory style vs Modern software development practices (cont.)

- Because of good documentation:
  - ✓ fault diagnosis (故障診斷) and maintenance are smoother now.
- Several metrics are being used:
  - ✓ help in software project management, quality assurance, etc.
- Projects are being thoroughly planned:
  - ✓ estimation,
  - ✓ scheduling,
  - ✓ monitoring mechanisms.
- Use of CASE tools.

Summary

- Software engineering is:
  - ✓ Systematic collection of decades of programming experience
  - ✓ Together with the innovations made by researchers.
- Principles deployed by Software Engineering to overcome human cognitive limitations
  - ✓ Abstraction (抽象化)
  - ✓ Decomposition (分解)
- Programs versus Software Products (程式與軟體產品)
- Emergence of Software Engineering (軟體工程的出現)