SOFTWARE ENGINEERING
Lecture 1

By

Dr. Jayanta Mondal

KIIT Deemed to be University

Syllabus

Software Process Models	**Software product, Software crisis, Handling complexity through Abstraction and Decomposition, Overview of software development activities.** Process Models: Classical waterfall model, iterative waterfall model, prototyping model, evolutionary model, spiral model, RAD model. Agile models: Extreme programming and Scrum.
Software Requirement Engineering	Requirement Gathering and analysis, Functional and non functional requirements, Software Requirement Specification(SRS) , IEEE 830 guidelines, Decision tables and trees.
Software Project Management	Responsibilities of a Software project manager, project planning, Metrics for project size estimation, Project estimation techniques, Empirical estimation techniques, COCOMO models, Scheduling, Organization & team structure, Staffing,.
Structural Analysis & Design	Overview of design process : High level and detailed design, Cohesion & coupling, Modularity and layering, Function–Oriented software design: Structural Analysis, Structural Design (DFD and Structured Chart), Object Oriented Analysis & Design, Command language, menu and iconic interfaces.
Testing Strategies	Coding, Code Review, Documentation, Testing: - Unit testing, Black-box Testing, White-box testing, Cyclomatic complexity measure, Coverage analysis, Debugging, Integration testing, System testing, Regression testing.
Software Reliability Software Maintenance	Software reliability, reliability measures, reliability growth modelling, Quality SEI CMM, Characteristics of software maintenance, software reverse engineering, software re engineering, Software reuse
Emerging Topics	Client-Server Software engineering, Service Oriented Architecture (SOA), Software as a Service (SaaS)

Topics of Lecture 1

What is the basic difference between a program and a software?

What is software engineering?

Is software engineering a science or an art?

What is Software Crisis?

How Software Engineering can be the solution for Software Crisis?

What are the different types of Software Development Projects?

What is the problem with exploratory style of software development?

What is Human Cognition Mechanism?

What are the Principles Deployed by Software Engineering to Overcome Human Cognitive Limitations?

Topics of Lecture 1

What is Abstraction?

What is Decomposition?

How software engineering emerged?

What is Control Flow-based Design?

What is Structured programming?

What is Data Structure-oriented Design?

What is Data Flow-oriented Design?

What is Object-oriented Design?

The Next Step: COMPUTER SYSTEMS ENGINEERING

What is Software?

- More than *computer programs.*
- The collection of programs, documentation and configuration data that ensures correct execution.
- Three major types:
  - Generic Product: Stand alone, Sold on open market.
  - Customized Product: For specific customer.
  - Embedded Product: Built into hardware.

What is Engineering?

Systematically identifying, understanding, and integrating the *constraints* on a design to produce a successful result.

The innovations and past experiences towards writing good quality programs cost effectively, have contributed to the emergence of the software engineering discipline.

Definition of Software Engineering:

- “A systematic collection of good program development practices and techniques”.
- “An *engineering approach* to develop software”

Software Engineering : Art or Science ??

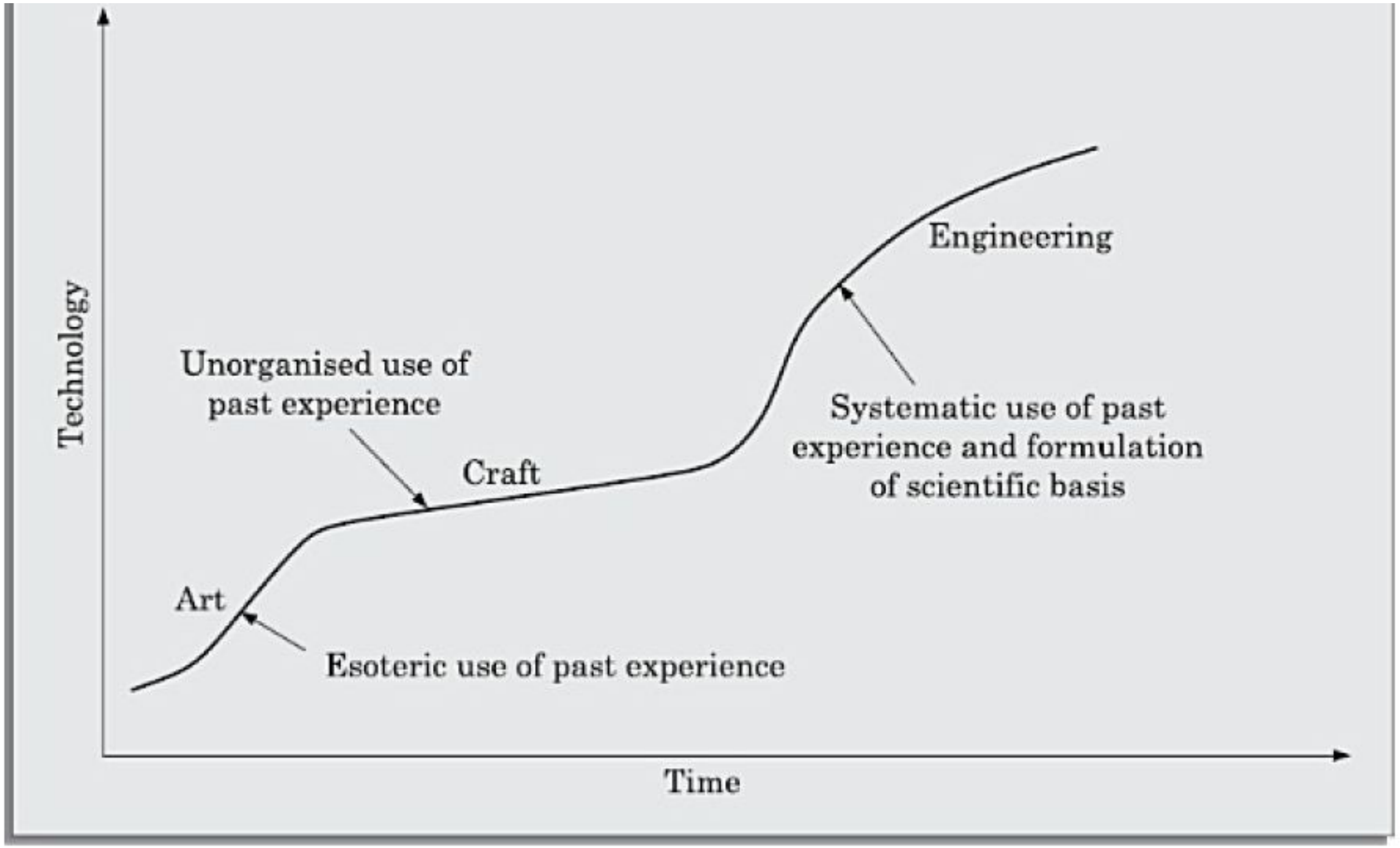


Figure 1.1: Evolution of technology with time.

Software Crisis

Software products:

- ⍰ fail to meet user requirements.
- ⍰ frequently crashed.
- ⍰ expensive.
- ⍰ difficult to alter, debug, and enhance.
- ⍰ often delivered late.
- ⍰ use resources non-optimally.

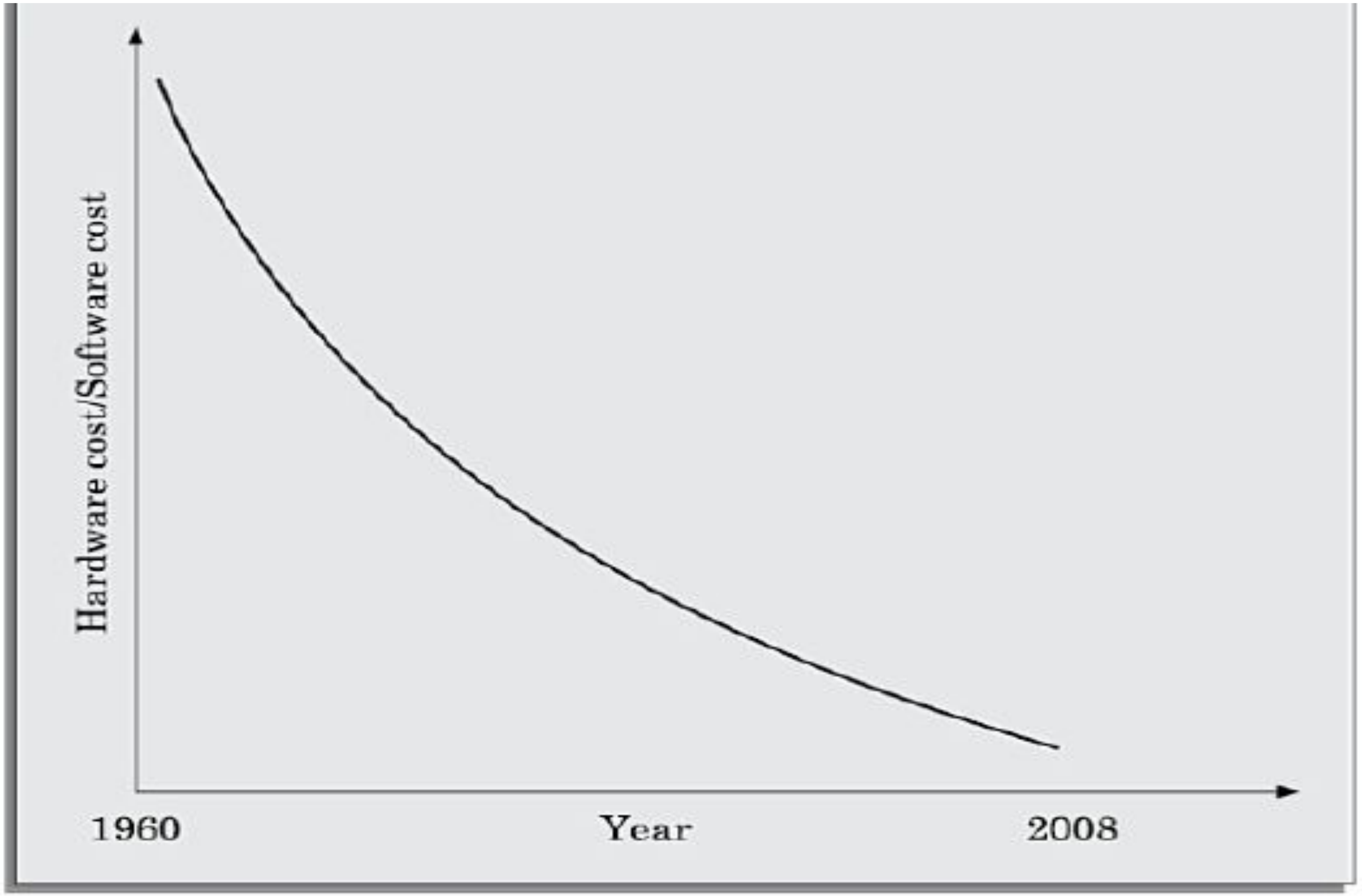


Figure 1.2: Relative changes of hardware and software costs over time.

what are the factors that have contributed to the present software crisis?

- rapidly increasing problem size
- lack of adequate training in software engineering techniques
- increasing skill shortage
- low productivity improvements.

What is the remedy?

- spread of software engineering practices among the developers
- further advancements to the software engineering discipline itself.

What is software Product?

 Computer programs and associated documentation such as requirements specification, design models and user manuals.

 Software products may be developed for a particular customer or may be developed for a general market.

 Software products may be
• Generic - developed to be sold to a range of different customers
e.g. PC software such as Excel or Word.
• Custom - developed for a single customer according
to their specification.

 New software can be created by developing new programs, configuring generic software systems or reusing existing software.

Software Services & Outsourced Software.

EXPLORATORY STYLE OF SOFTWARE DEVELOPMENT

- the programmer makes use of his own intuition to develop a program rather than making use of the systematic body of knowledge categorized under the software engineering discipline.
- The software is tested and the bugs found are fixed. This cycle of testing and bug fixing continues till the software works satisfactorily for the customer.

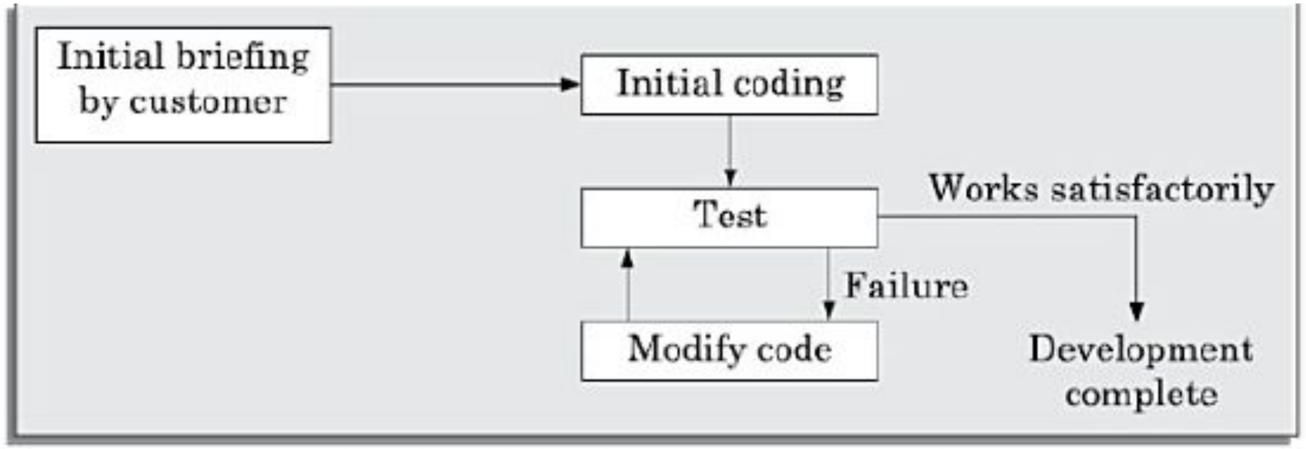


Figure 1.3: Exploratory program development.

What is wrong with the exploratory style of software development?

- The effort and time required to develop a professional software increases with the increase in program size.

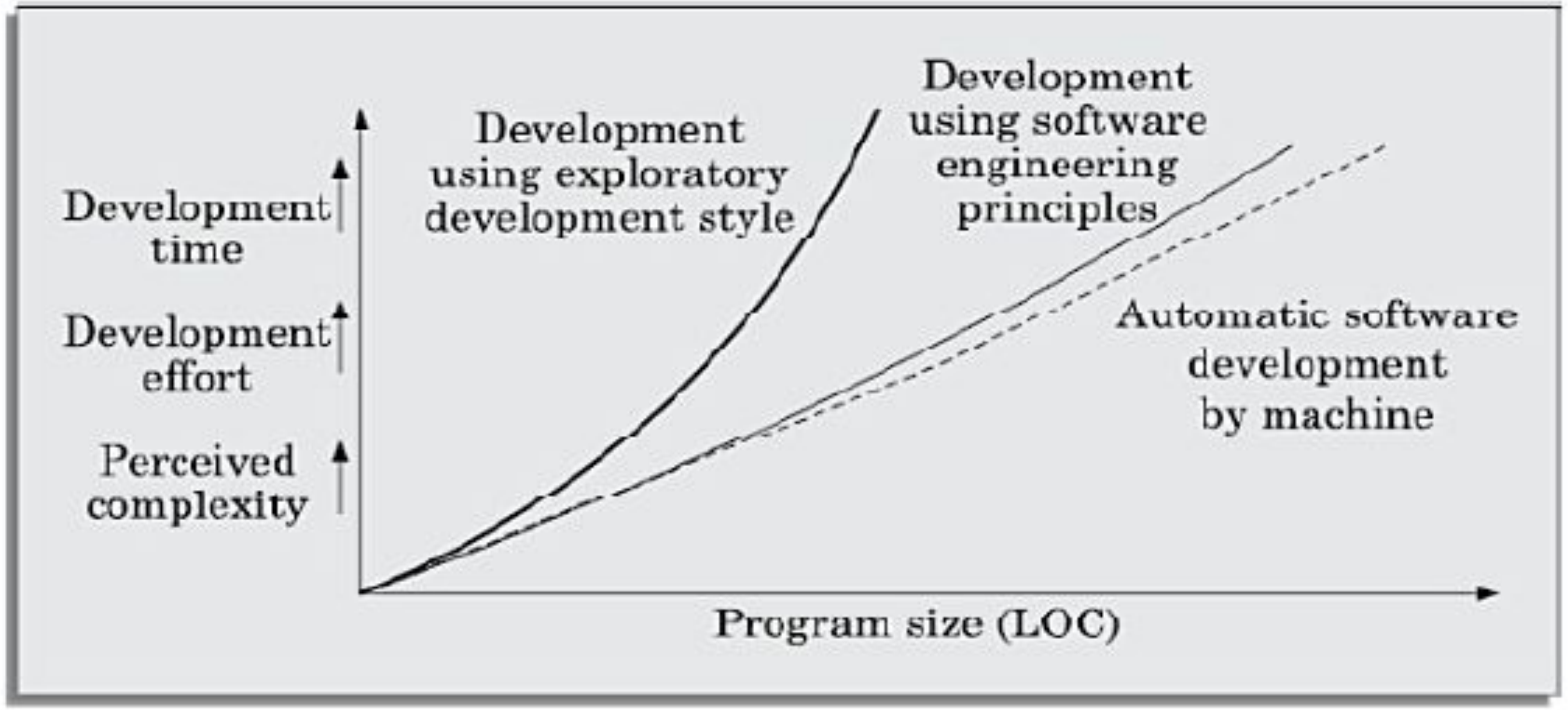


Figure 1.4: Increase in development time and effort with problem size.

Main Disadvantages of exploratory style of software development:

The foremost difficulty is the exponential growth of development time and effort with problem size and large-sized software becomes almost impossible using this style of development.

The exploratory style usually results in unmaintainable code.

It becomes very difficult to use the exploratory style in a team development environment.

Human Cognition Mechanism

Human memory can be thought to consist of two distinct parts: short-term and long-term memories.

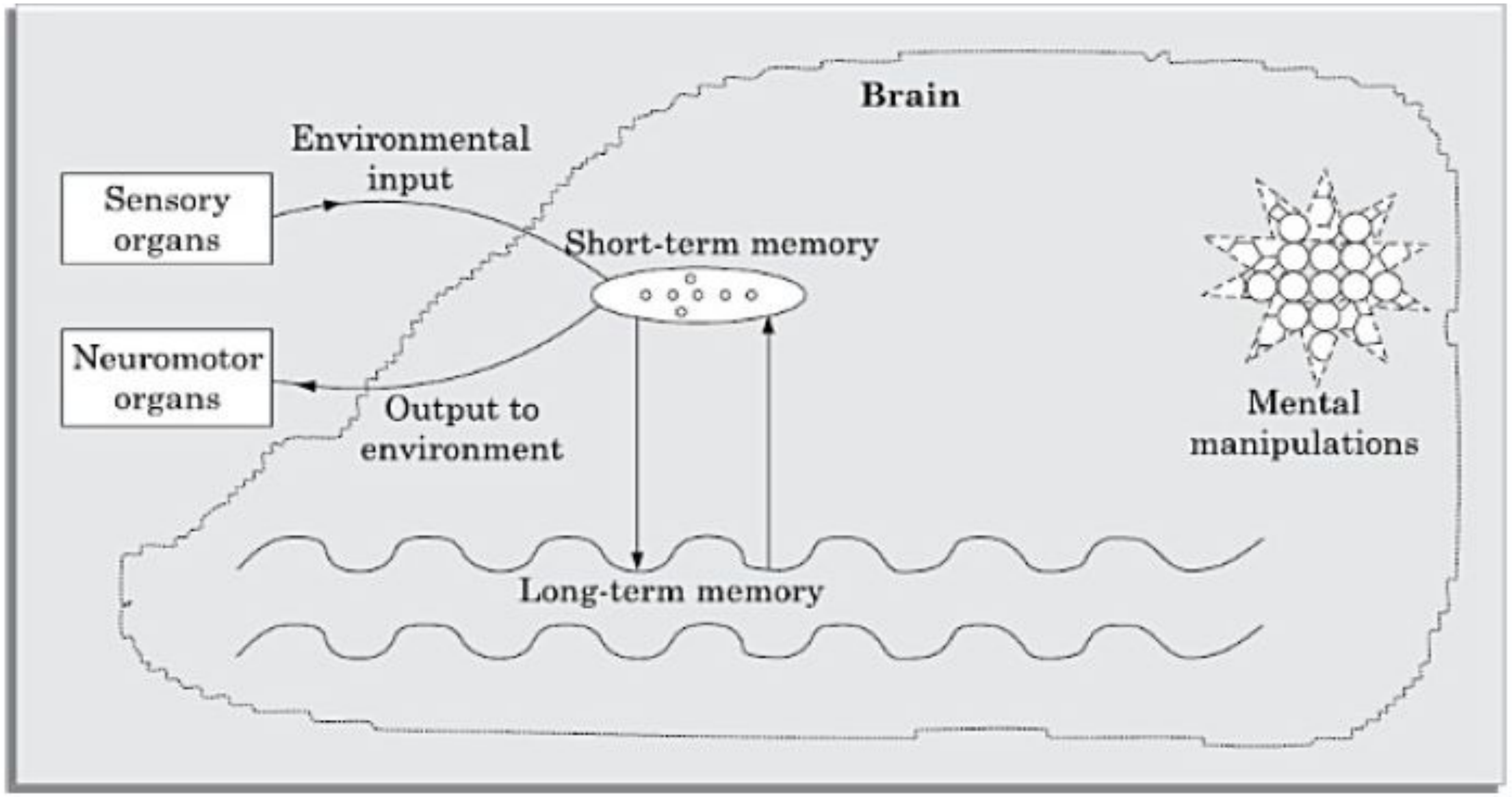


Figure 1.5: Human cognition mechanism model.

- When the number of details (or variables) that one has to track to solve a problem increases beyond seven, it exceeds the capacity of the short-term memory and it becomes exceedingly more difficult for a human mind to grasp the problem.

- Two important principles that are deployed by software engineering to overcome the problems arising due to human cognitive limitations are—abstraction and decomposition.

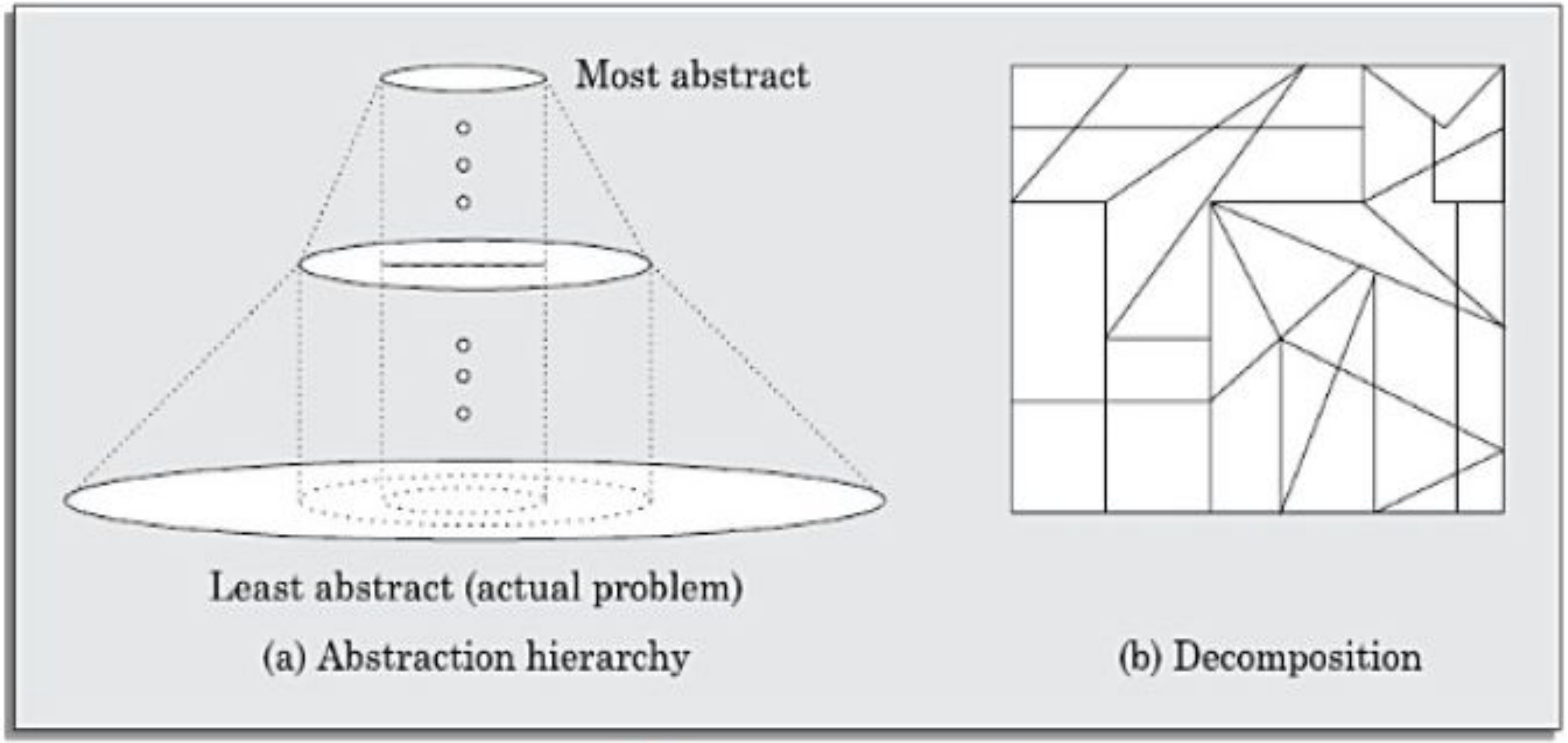


Figure 1.6: Schematic representation.

- Abstraction is the simplification of a problem by focusing on only one aspect of the problem while omitting all other aspects.

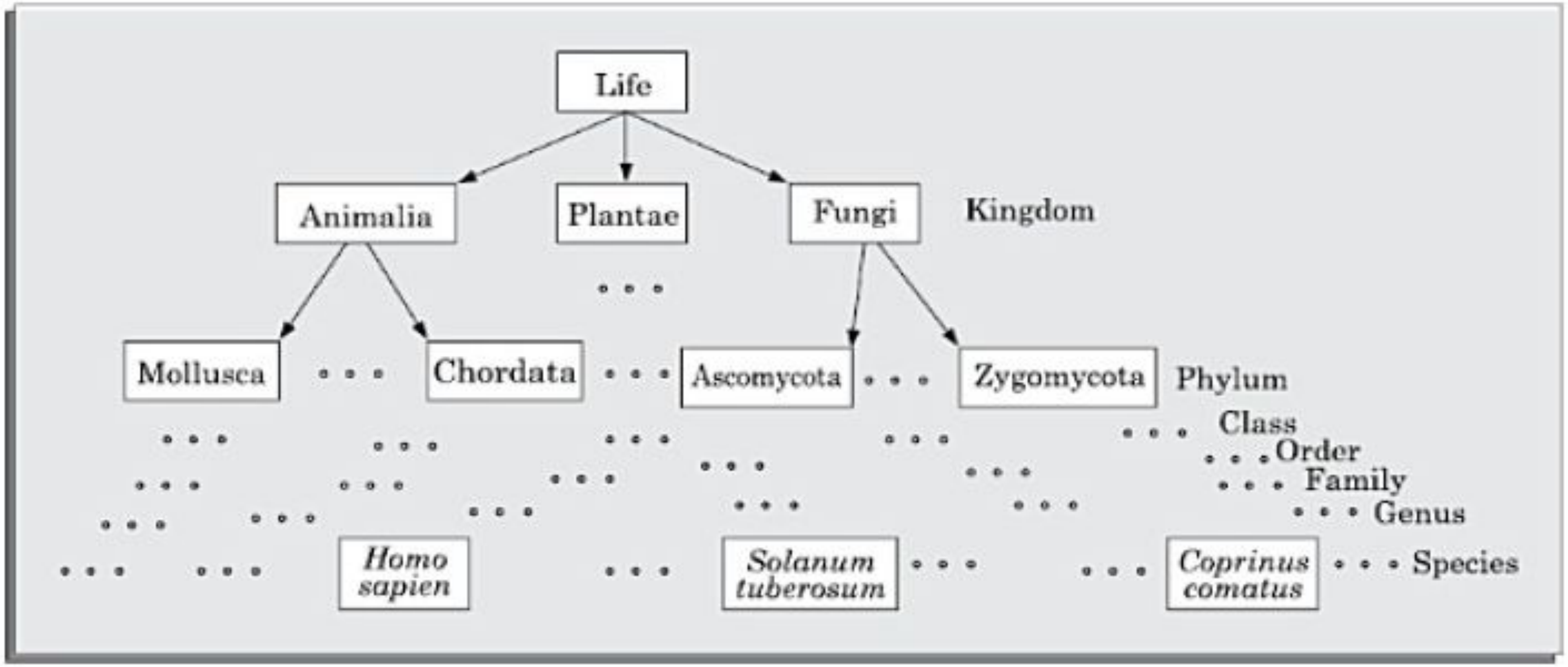


Figure 1.7: An abstraction hierarchy classifying living organisms.

- The decomposition principle advocates decomposing the problem into many small independent parts. The small parts are then taken up one by one and solved separately. The idea is that each small part would be easy to grasp and understand and can be easily solved. The full problem is solved when all the parts are solved.

- Figure 1.6(b) shows the decomposition o f a large problem into many small parts.

- However, it is very important to understand that any arbitrary decomposition of a problem into small parts would not help. The different parts after decomposition should be more or less independent of each other.

Why study software engineering?

- The skill to participate in development of large software. You can meaningfully participate in a team effort to develop a large software only after learning the systematic techniques that are being used in the industry.
- You would learn how to effectively handle complexity in a software development problem. In particular, you would learn how to apply the principles of abstraction and decomposition to handle complexity during various stages in software development such as specification, design, construction, and testing.

EMERGENCE OF SOFTWARE ENGINEERING

Early Computer Programming

High-level Language Programming

Control Flow-based Design

Data Structure-oriented Design

Data Flow-oriented Design

Object-oriented Design

Early Computer Programming

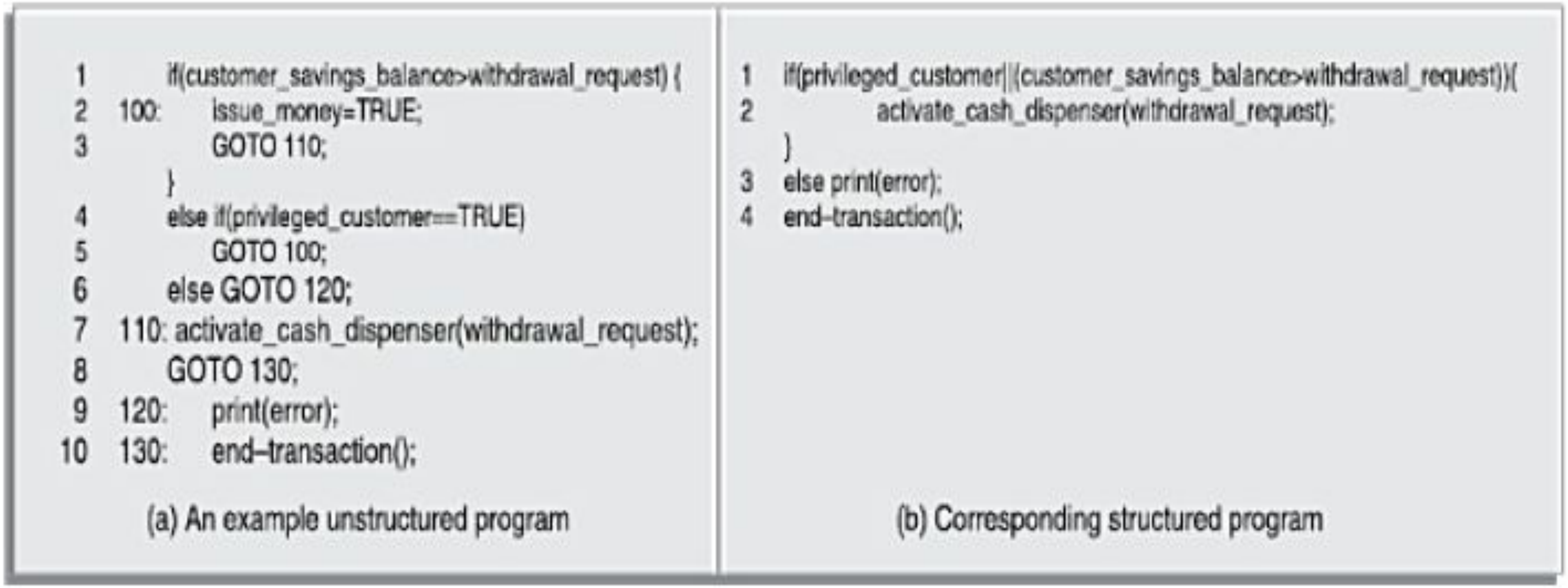
- Early commercial computers were very slow and too elementary as compared to today's standards.
- Even simple processing tasks took considerable computation time on those computers.
- Program lengths were typically limited to about a few hundreds of lines of monolithic assembly code.
- Every programmer developed his own individualistic style of writing programs according to his intuition and used this style *ad hoc* while writing different programs.

High-level Language Programming

- Computers became faster with the introduction of the semiconductor technology in the early 1960s.
- At this time, high-level languages such as FORTRAN, ALGOL, and COBOL were introduced.
- Typical programs were limited t o sizes of around a few thousands of lines of source code.
- However, programmers were still using the exploratory style of software development.

Control Flow-based Design

- As the size and complexity of programs kept on increasing, the exploratory programming style proved to be insufficient.
- A program’s control flow structure indicates the sequence in which the program’s instructions are executed.
- In order to help develop programs having good control flow structures, the *flow charting technique* was developed.



```
     if(customer_savings_balance>withdrawal_request) {
100:     issue_money=TRUE;
         GOTO 110;
     }
     else if(privileged_customer==TRUE)
         GOTO 100;
     else GOTO 120;
110: activate_cash_dispenser(withdrawal_request);
     GOTO 130;
120:     print(error);
130:     end-transaction();
```

(a) An example unstructured program

```
if(privileged_customer||(customer_savings_balance>withdrawal_request)){
        activate_cash_dispenser(withdrawal_request);
}
else print(error);
end-transaction();
```

(b) Corresponding structured program

Figure 1.8: An example of (a) Unstructured program (b) Corresponding structured program.

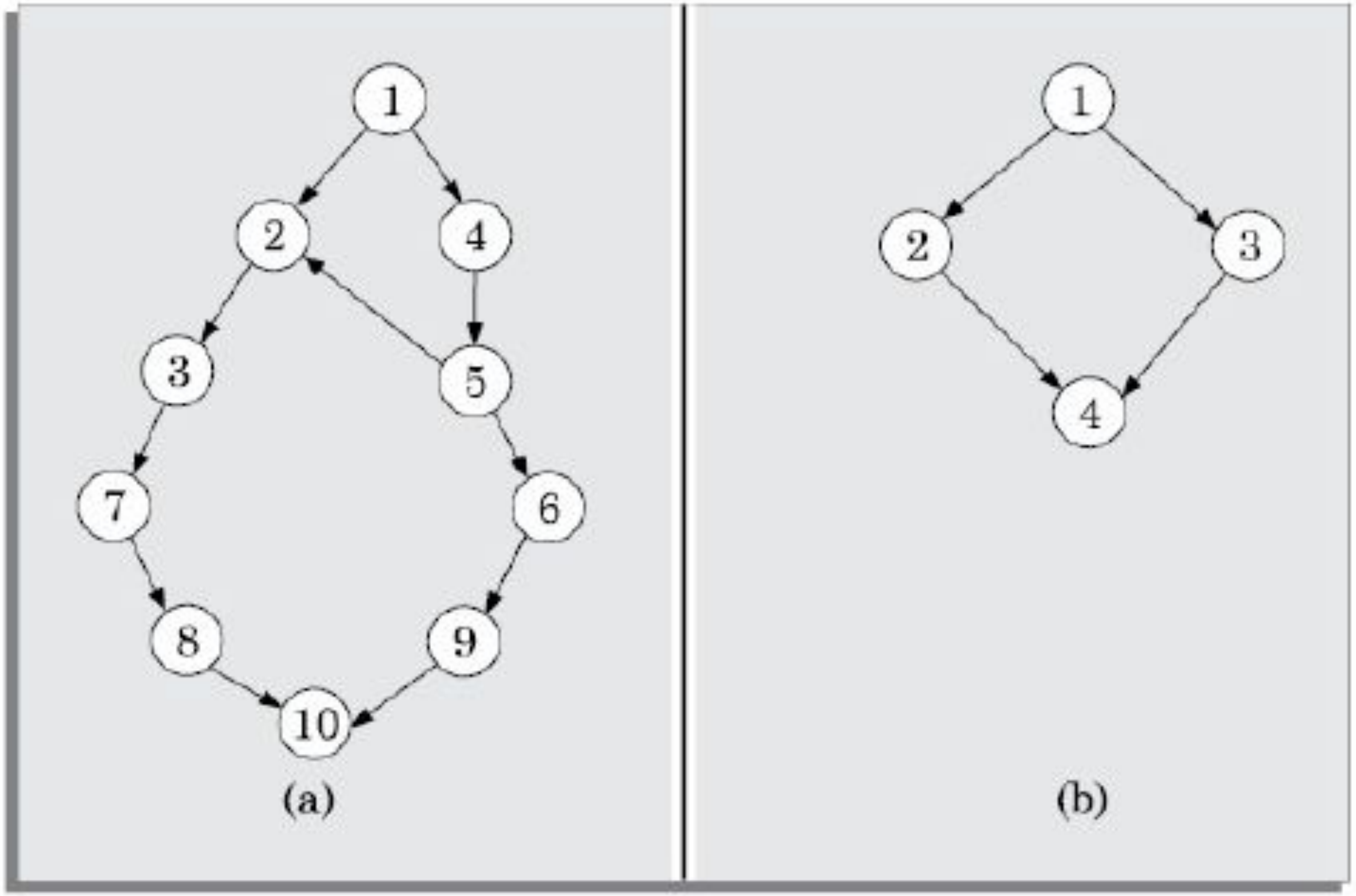


Figure 1.9: Control flow graphs of the programs of Figures 1.8(a) and (b).

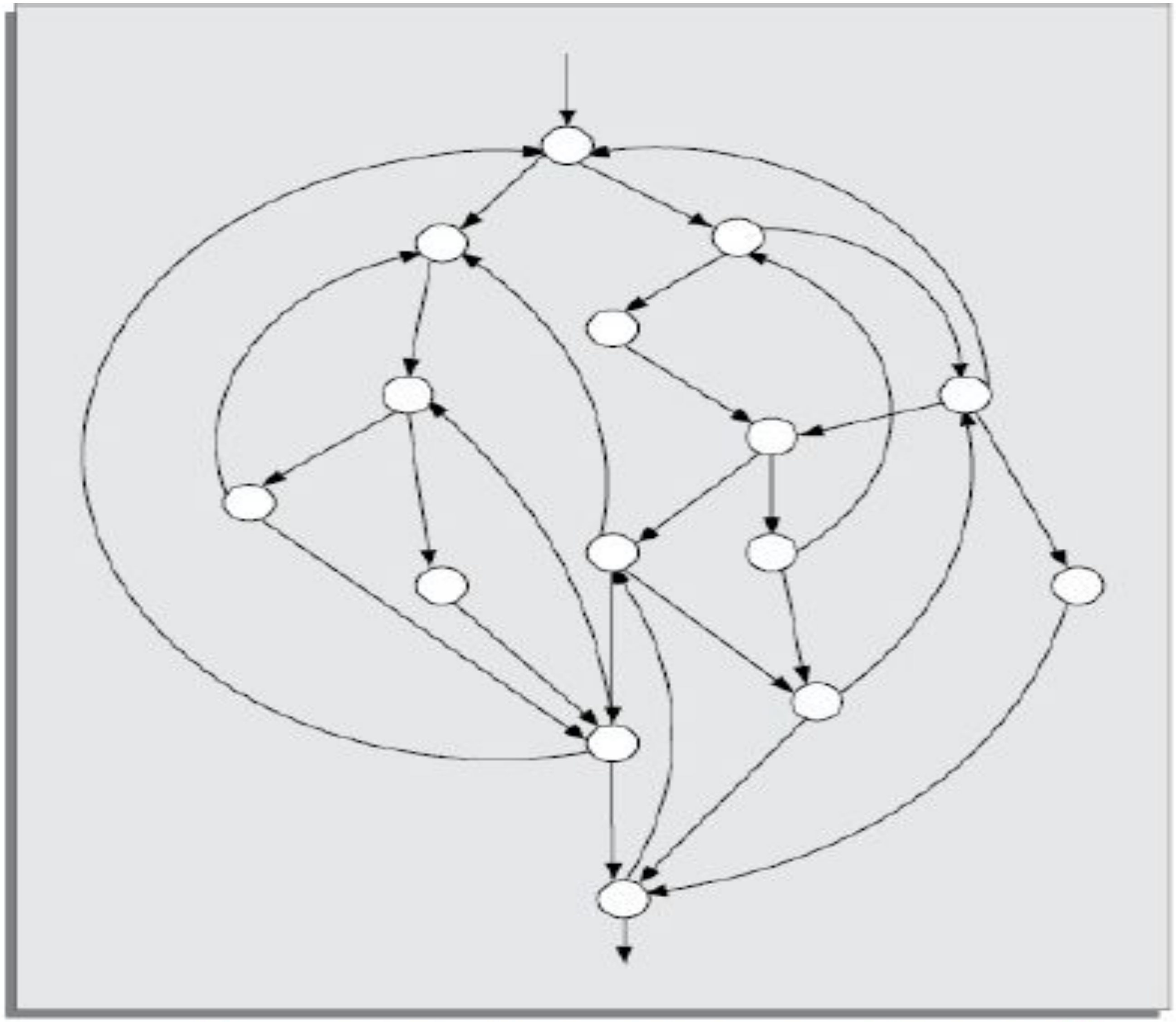


Figure 1.10: CFG of a program having too many GO TO statements.

Execution Paths: 1-2-3-7-8-10,
1-4-5-6-9-10, 1-4-5-2-3-7-8-10

Structured programming

It was conclusively proved by Bohm and Jacopini that only three programming constructs—sequence, selection, and iteration—were sufficient to express any programming logic.

- A program is called structured when it uses only the sequence, selection, and iteration types of constructs and is modular.

Structured programs avoid unstructured control flows by restricting the use of GO TO statements.

Data Structure-oriented Design

- Computers became even more powerful with the advent of *integrated circuits* (ICs) in the early seventies.
- Using data structure-oriented design techniques, first a program’s data structures are designed. The code structure is designed based on the data structure.
- Example: Jackson’s Structured Programming (JSP) technique, Warnier-Orr Methodology.
- However, they were replaced by the data flow based and the object-oriented techniques.

Data Flow-oriented Design

- As computers became still faster and more powerful with the introduction of *very large scale integrated* (VLSI) Circuits and some new architectural concepts, more complex and sophisticated software were needed to solve further challenging problems. Therefore, software developers looked out for more effective techniques for designing software and soon *data flow-oriented techniques* were proposed.
- The data flow-oriented techniques advocate that the major data items handled by a system must be identified and the processing required on these data items to produce the desired outputs should be determined.

Object-oriented Design

- Data flow-oriented techniques evolved into object-oriented design (OOD) techniques in the late seventies.

- Each object essentially acts as a *data hiding* (also known as *data abstraction*) entity.

Aspect-Oriented Programming*: web-based & embedded software*

Many of the present day software are required to work in a client-server environment through a web browser-based access.

Embedded devices are experiencing an unprecedented growth and rapid customer acceptance in the last decade.

In the current decade, service orientation has emerged as a recent direction of software engineering due to the popularity of web-based applications and public clouds.

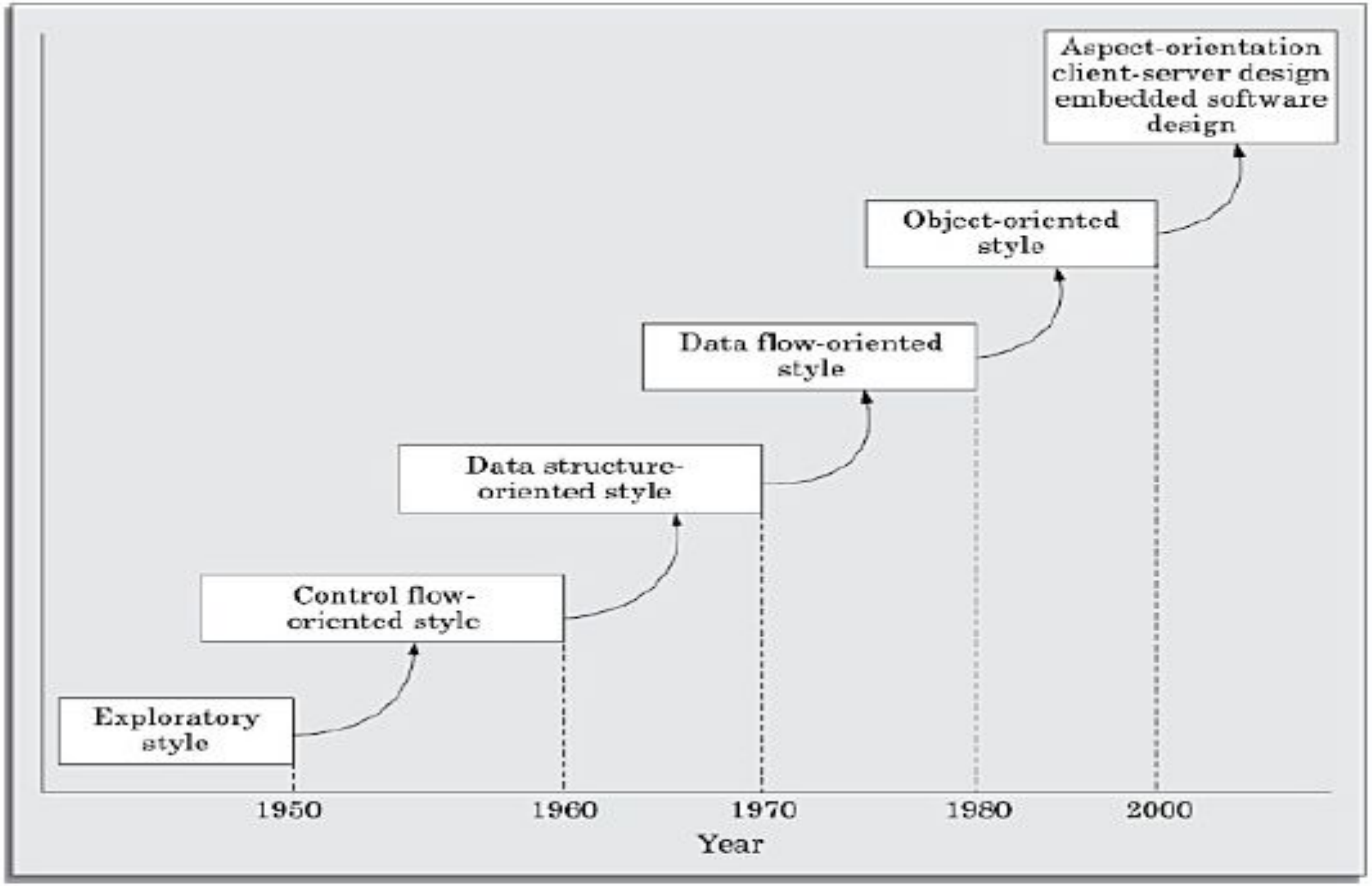

Aspect-orientation client-server design embedded software design
Object-oriented style
Data flow-oriented style
Data structure-oriented style
Control flow-oriented style
Exploratory style
1950
1960
1970
1980
2000
Year

Figure 1.11: Evolution of software design techniques.

NOTABLE CHANGES IN SOFTWARE DEVELOPMENT PRACTICES

- An important difference is that the exploratory software development style is based on *error correction (build and fix)* while the software engineering techniques are based on the principles of *error prevention*.
- In the exploratory style, coding was considered synonymous with software development. In the modern software development style, coding is regarded as only a small part of the overall software development activities.
- A lot of attention is now being paid to requirements specification.
- Periodic reviews are being carried out during all stages of the development process for *phase containment of errors.*

NOTABLE CHANGES IN SOFTWARE DEVELOPMENT PRACTICES

- Today, software testing has become very systematic.
- In the past, very little attention was being paid to producing good quality and consistent documents.
- Now, projects are being thoroughly planned. The primary objective of project planning is to ensure that the various development activities take place at the correct time and no activity is halted due to the want of some resource.

Quality of Good Software

- **Usability**
  - Easy to learn and use.
- **Efficiency**
  - Does not waste resources such as CPU time and memory.
- **Dependability**
  - Reliable, secure and safe.
- **Maintainability**
  - Easily evolved (modified) to meet changing requirement.
- **Reusability**
  - Parts can be reused, with minor or no modification.

COMPUTER SYSTEMS ENGINEERING

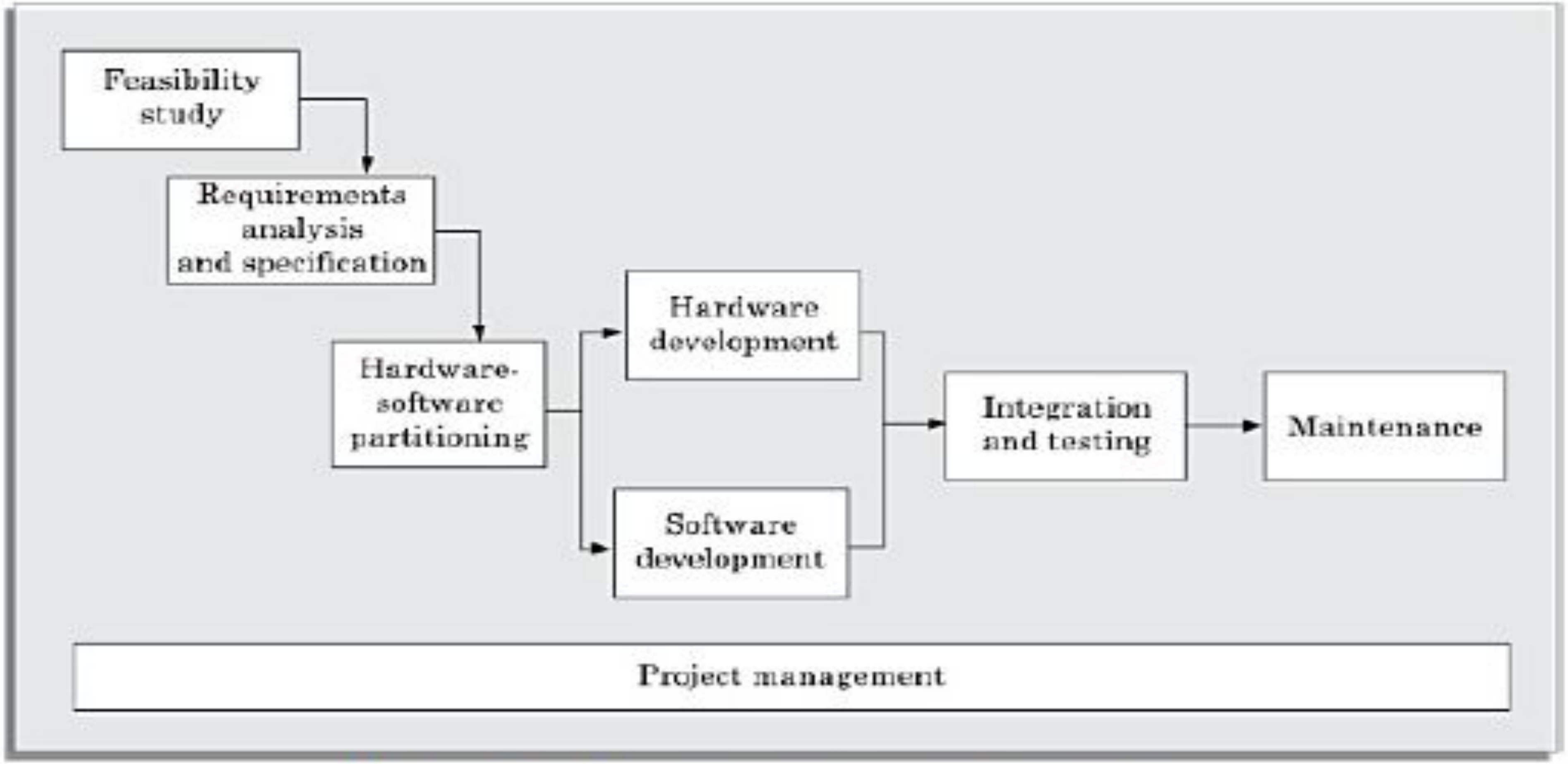


Figure 1.12: Computer systems engineering.

Thank you

If you have any doubts …. Feel free to ask