Software Engineering

Syllabus

- Unit Content Hrs/Unit Marks/Unit
- 1 Overview of System Analysis & Design , Business System Concept, System Development Life Cycle, Waterfall Model ,
- Spiral Model, Feasibility Analysis, Technical Feasibility, Cost- Benefit Analysis, COCOMO
- model. **[10L]** 10
- 2 System Design – Context diagram and DFD, Problem Partitioning, Top-Down And Bottom-Up design; Decision tree,
- decision table and structured English; Functional vs.
- Object- Oriented approach. **[5L]** 5
- 3 Coding & Documentation – Structured Programming, OO Programming, Information Hiding, Reuse, System
- Documentation. **[4L]**
- Testing – Levels of Testing, Integration Testing, Test case Specification, Reliability Assessment, Validation & Verification 12
- Metrics, Monitoring & Control. **[8L]**
- 4. Software Project Management – Project Scheduling, Staffing, Software Configuration Management, Quality Assurance,
- Project Monitoring. **[7L]**
- Static and dynamic models, why modeling, UML diagrams: Class diagram, interaction diagram: collaboration diagram,
- sequence diagram, state chart diagram, activity diagram, implementation diagram.

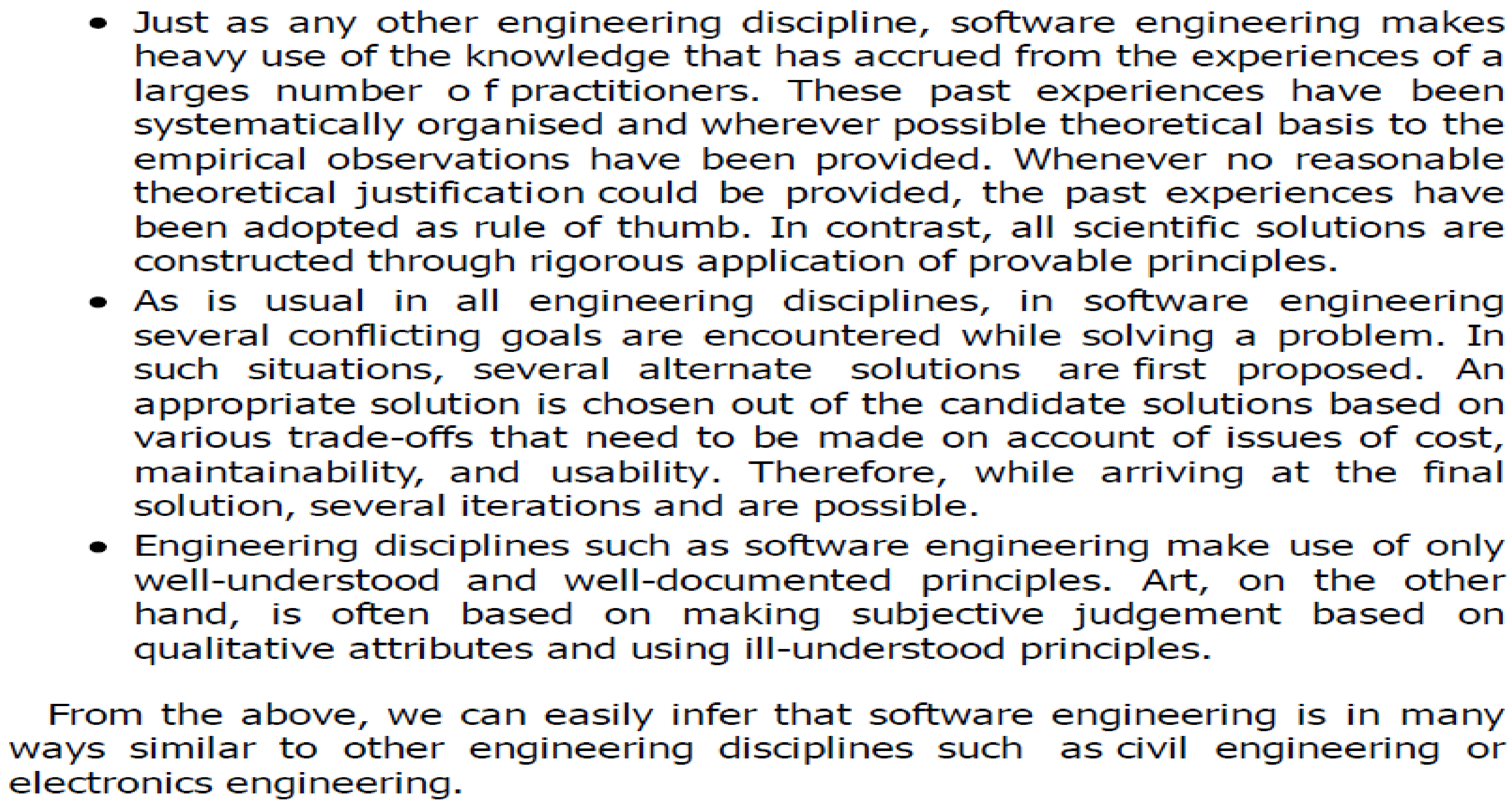
What is Software Engineering

- “A systematic collection of good program development practices and techniques”.
- “An *engineering approach* to develop software”. Based on these two point of views,

We can define software engineering as follows:

- Software engineering discusses systematic and cost-effective techniques for software development.
- These techniques help develop software using an engineering
- approach.

Is software engineering a science or an art?



- Just as any other engineering discipline, software engineering makes heavy use of the knowledge that has accrued from the experiences of a larges number o f practitioners. These past experiences have been systematically organised and wherever possible theoretical basis to the empirical observations have been provided. Whenever no reasonable theoretical justification could be provided, the past experiences have been adopted as rule of thumb. In contrast, all scientific solutions are constructed through rigorous application of provable principles.
- As is usual in all engineering disciplines, in software engineering several conflicting goals are encountered while solving a problem. In such situations, several alternate solutions are first proposed. An appropriate solution is chosen out of the candidate solutions based on various trade-offs that need to be made on account of issues of cost, maintainability, and usability. Therefore, while arriving at the final solution, several iterations and are possible.
- Engineering disciplines such as software engineering make use of only well-understood and well-documented principles. Art, on the other hand, is often based on making subjective judgement based on qualitative attributes and using ill-understood principles.

From the above, we can easily infer that software engineering is in many ways similar to other engineering disciplines such as civil engineering or electronics engineering.

EVOLUTION—FROM AN ART FORM TO A N ENGINEERING DISCIPLINE

- 1.1.1 Evolution of an Art into an Engineering Discipline
- The early programmers used an *ad hoc* programming style. This style of
- program development is now variously being referred to as *exploratory*, *build*
- *and fix*, and *code and fix* styles.

Evolution Pattern for Engineering Disciplines

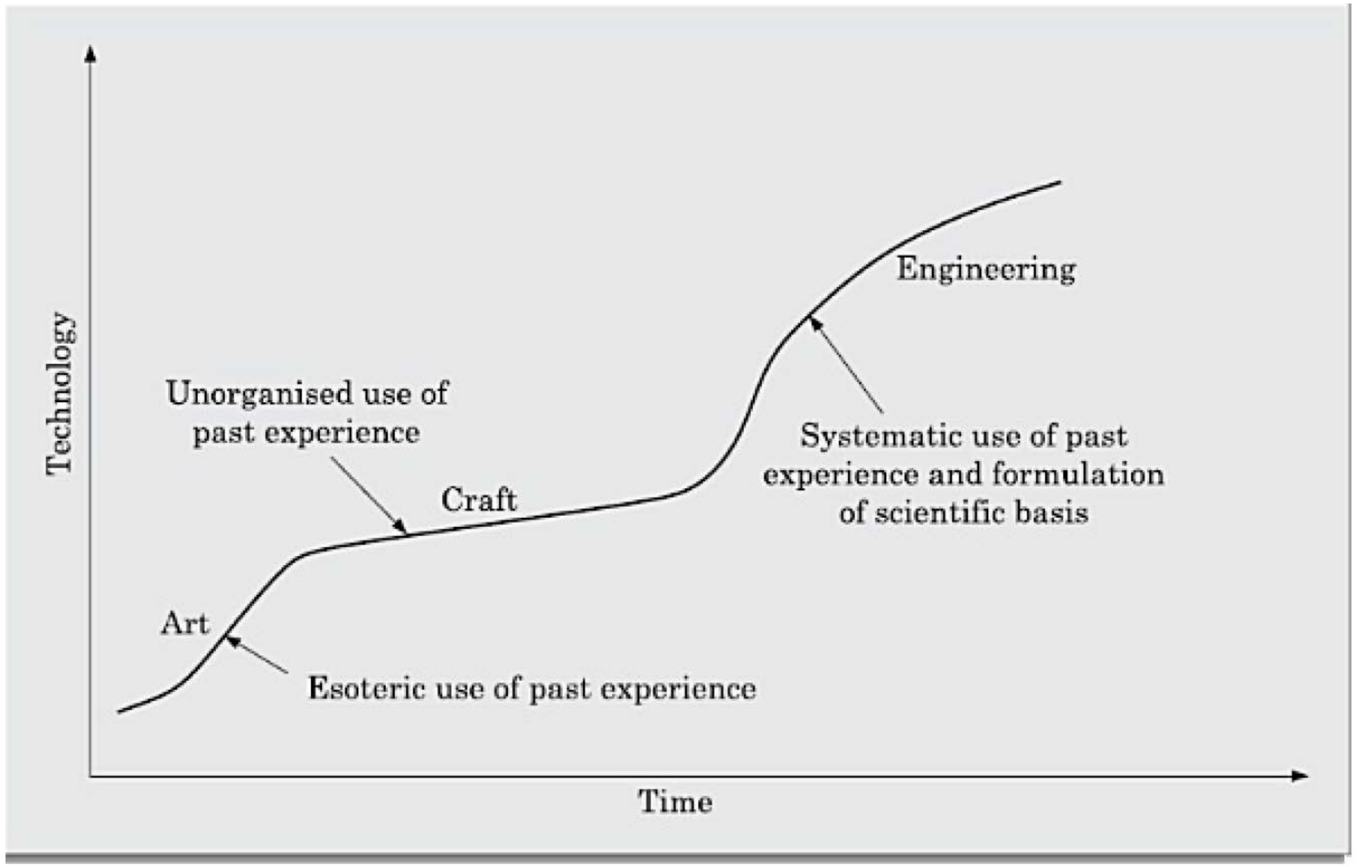

Technology
Unorganised use of past experience
Craft
Engineering
Systematic use of past experience and formulation of scientific basis
Art
Esoteric use of past experience
Time

Figure 1.1: Evolution of technology with time.

1.1.2 Evolution Pattern for Engineering Disciplines

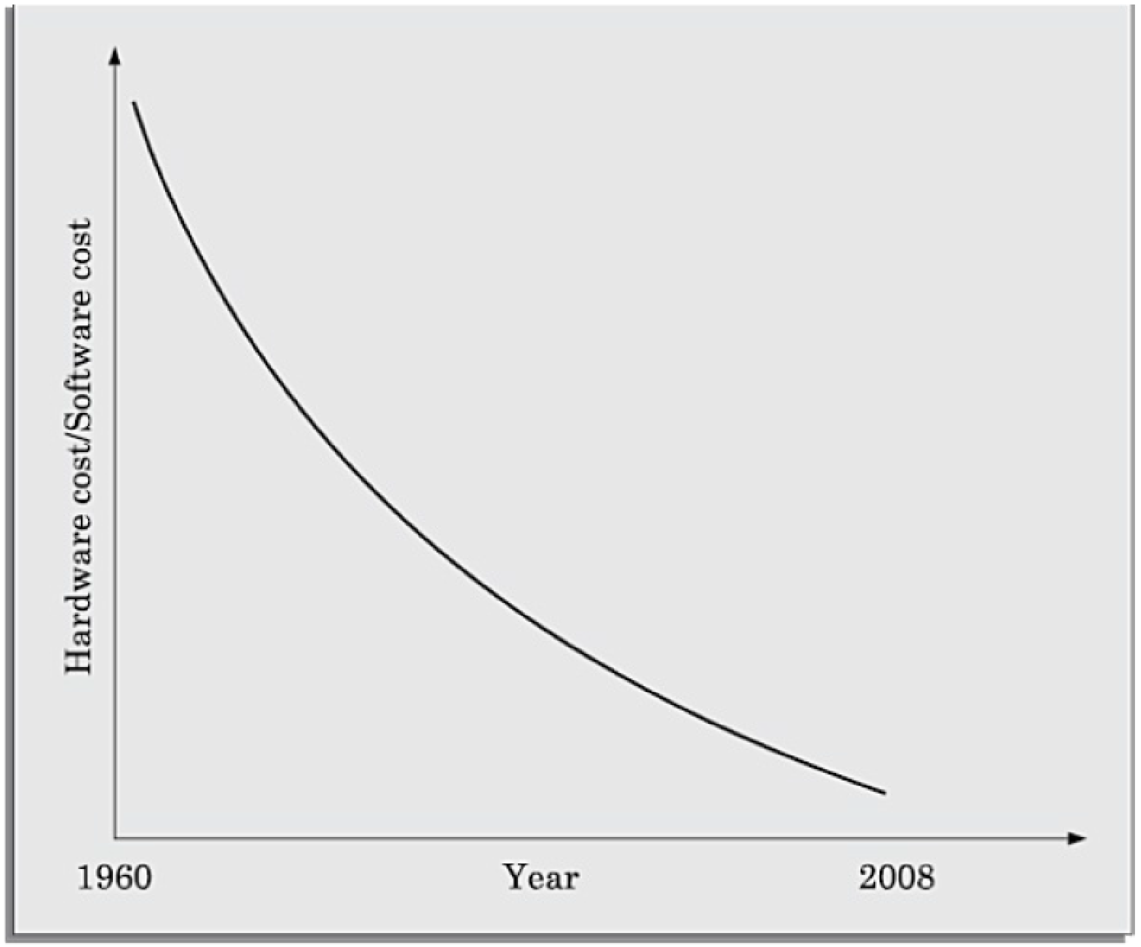


Figure 1.2: Relative changes of hardware and software costs over time.

A Solution to the Software Crisis

- Not only are the software products becoming progressively more expensive than hardware, but they also present a host of other problems to the customers
- —software products are difficult to alter, debug, and enhance; use resources non-optimally; often fail to meet the user requirements; are far from being reliable; frequently crash; and are often delivered late.

SOFTWARE DEVELOPMENT PROJECTS

- Programs ***versus*** Products
- Software services

EXPLORATORY STYLE OF SOFTWARE DEVELOPMENT

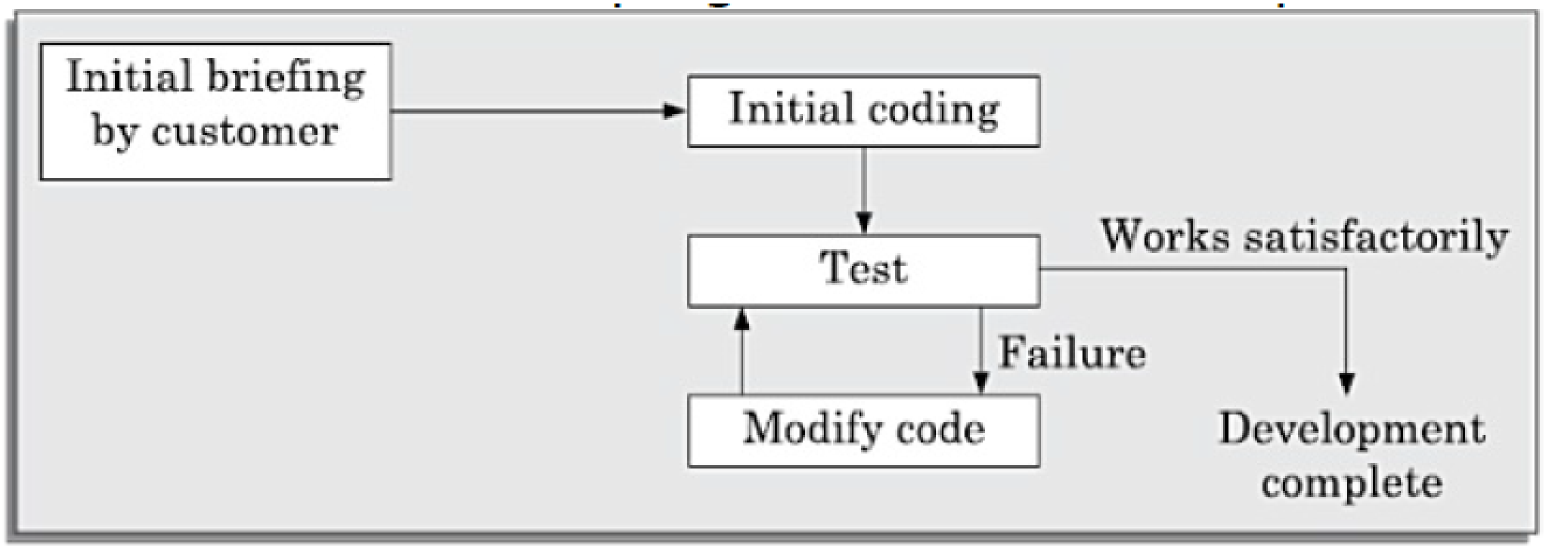


Figure 1.3: Exploratory program development.

What is wrong with the exploratory style of software development?

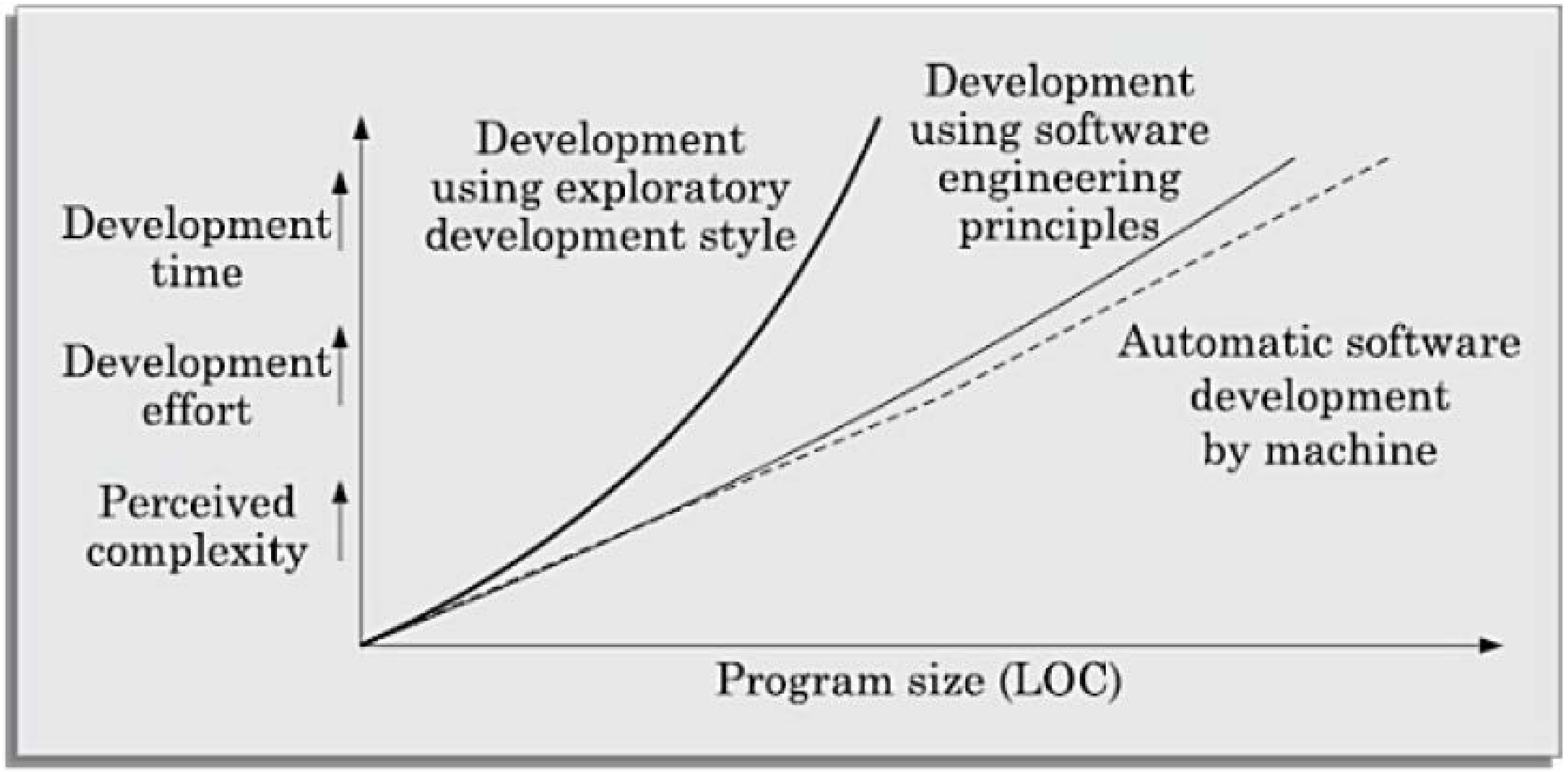
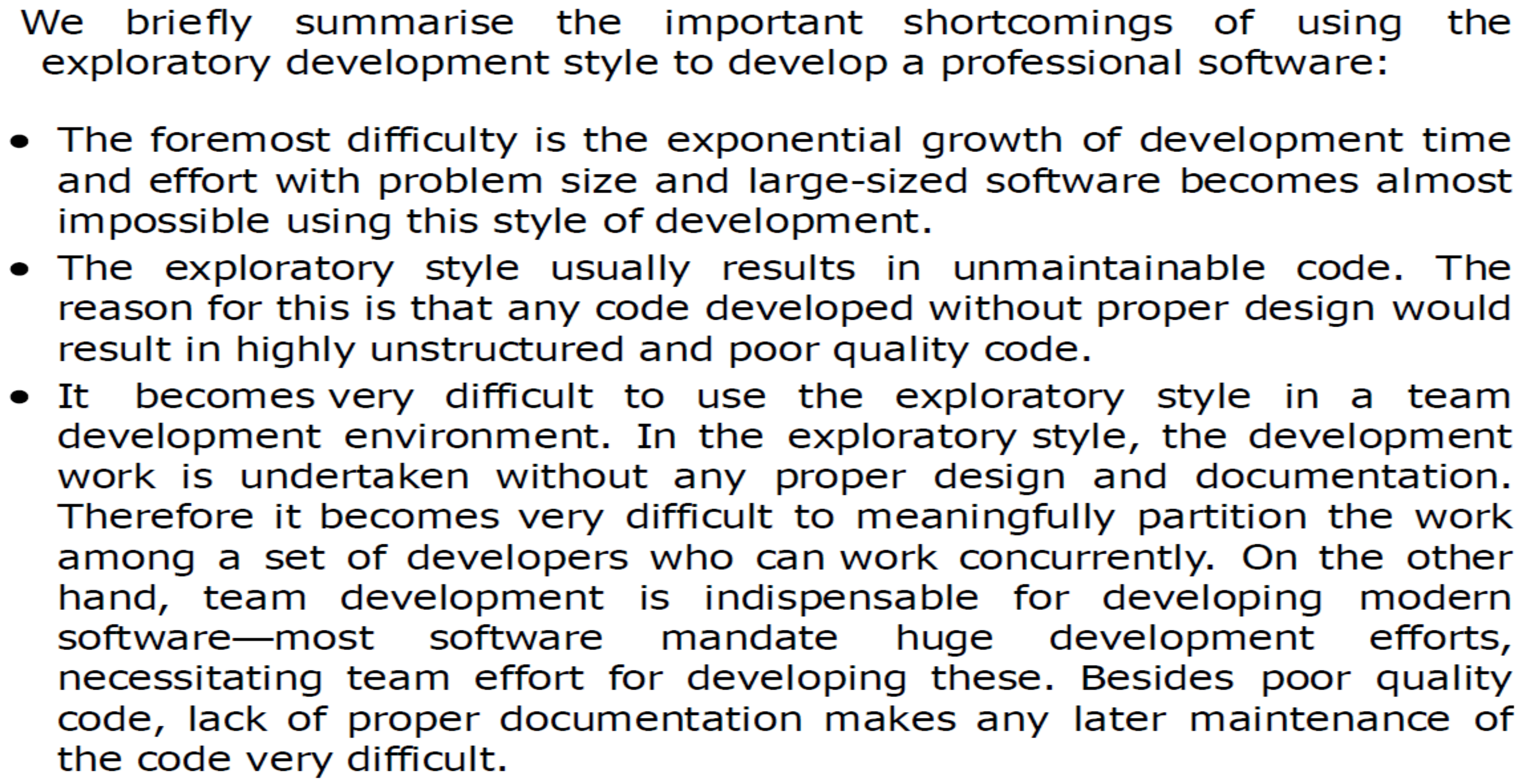


Figure 1.4: Increase in development time and effort with problem size.

Summary of the shortcomings of the exploratory style of software development:



We briefly summarise the important shortcomings of using the exploratory development style to develop a professional software:

- 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. The reason for this is that any code developed without proper design would result in highly unstructured and poor quality code.
- It becomes very difficult to use the exploratory style in a team development environment. In the exploratory style, the development work is undertaken without any proper design and documentation. Therefore it becomes very difficult to meaningfully partition the work among a set of developers who can work concurrently. On the other hand, team development is indispensable for developing modern software—most software mandate huge development efforts, necessitating team effort for developing these. Besides poor quality code, lack of proper documentation makes any later maintenance of the code very difficult.

Principles Deployed by Software Engineering to Overcome Human Cognitive Limitations

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

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

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.

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**

ad hoc & build and fix (or the *exploratory programming*) style

- **High-level Language Programming**

FORTRAN, ALGOL, and COBOL

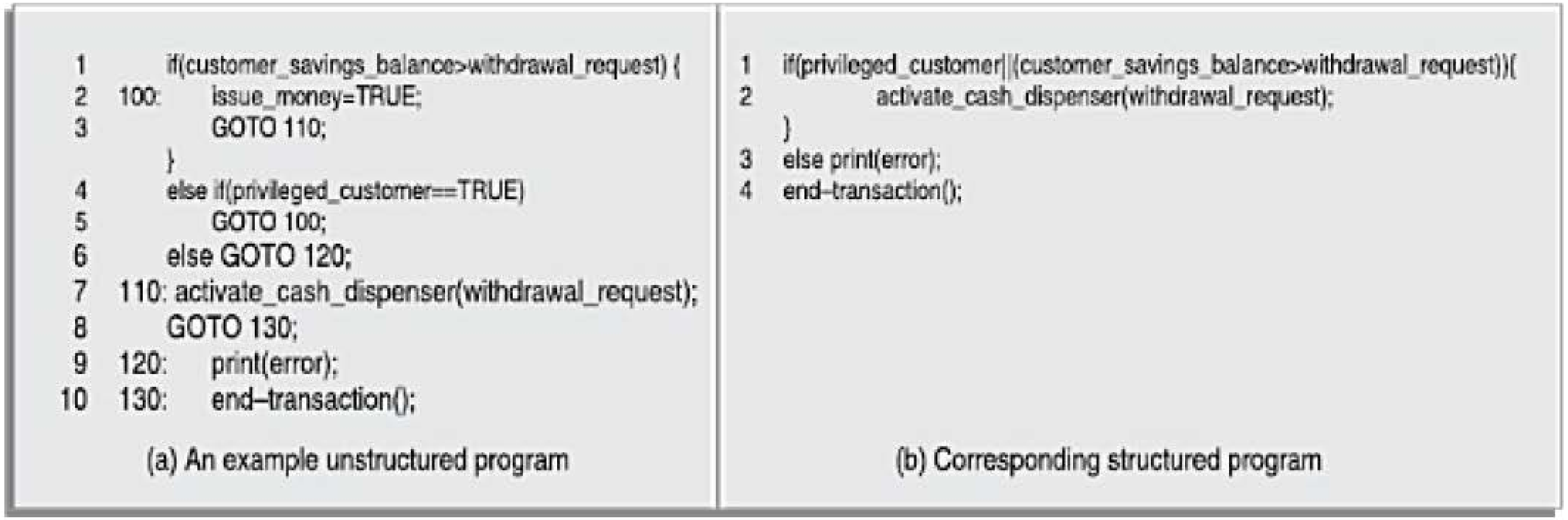
Control Flow-based Design

- ***Control flow structure*.**

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.

Examples



```
      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

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

Control flow Graph

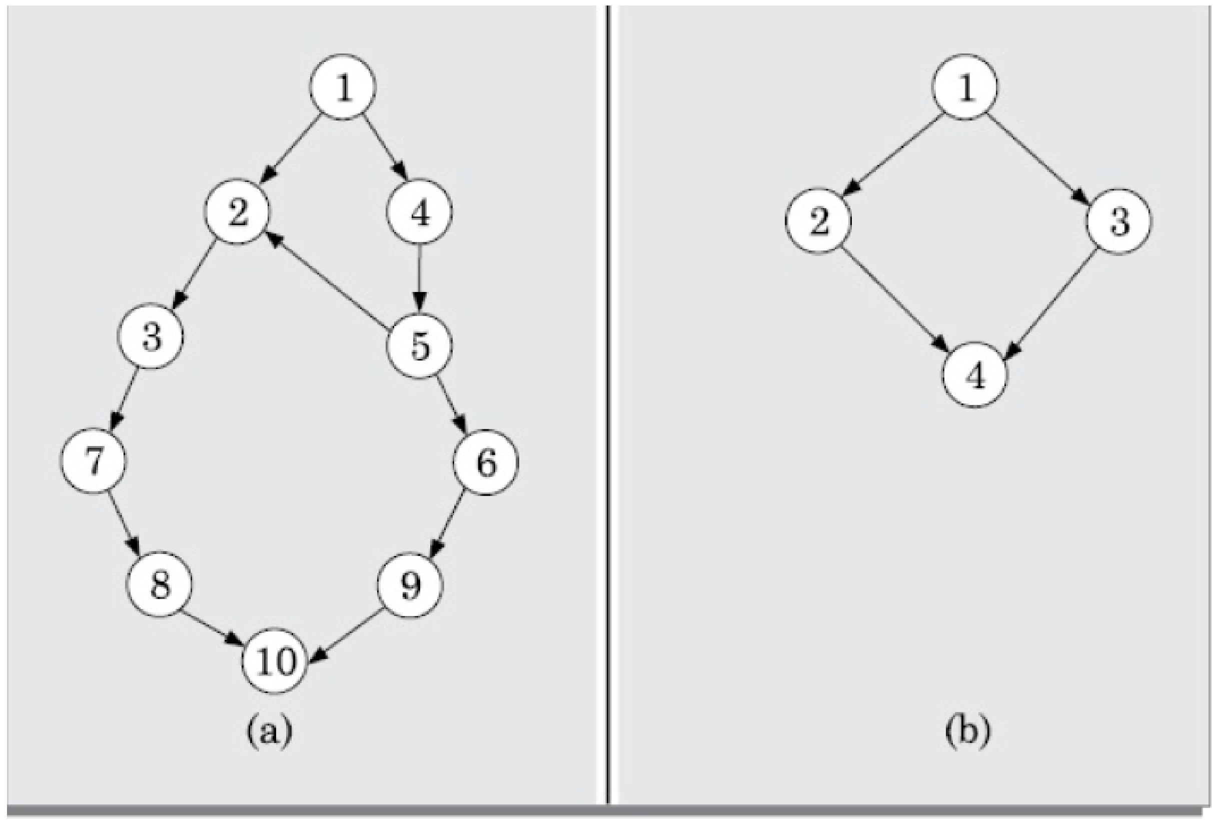


Figure 1.9: Control flow graphs of the programs of Figures 1.8(a) and (b).

CFG of many Go to statements

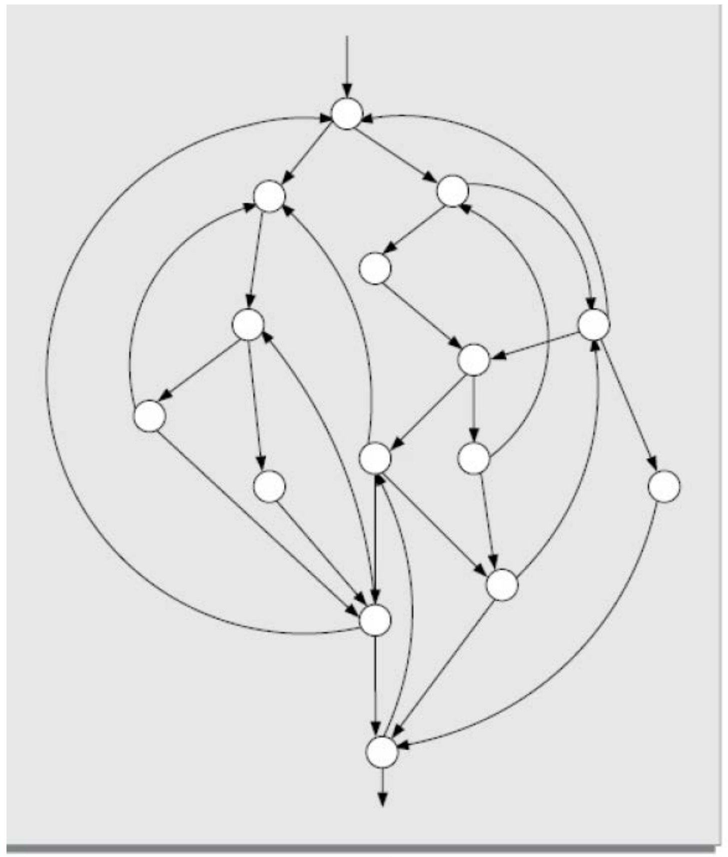


Figure 1.10: CFG of a program having too many GO TO statements.

Data Structure-oriented Design

Using data structure-oriented design techniques, first a program’s data structures are designed. The code structure is designed based on the data structure.

Data Flow-oriented Design

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.

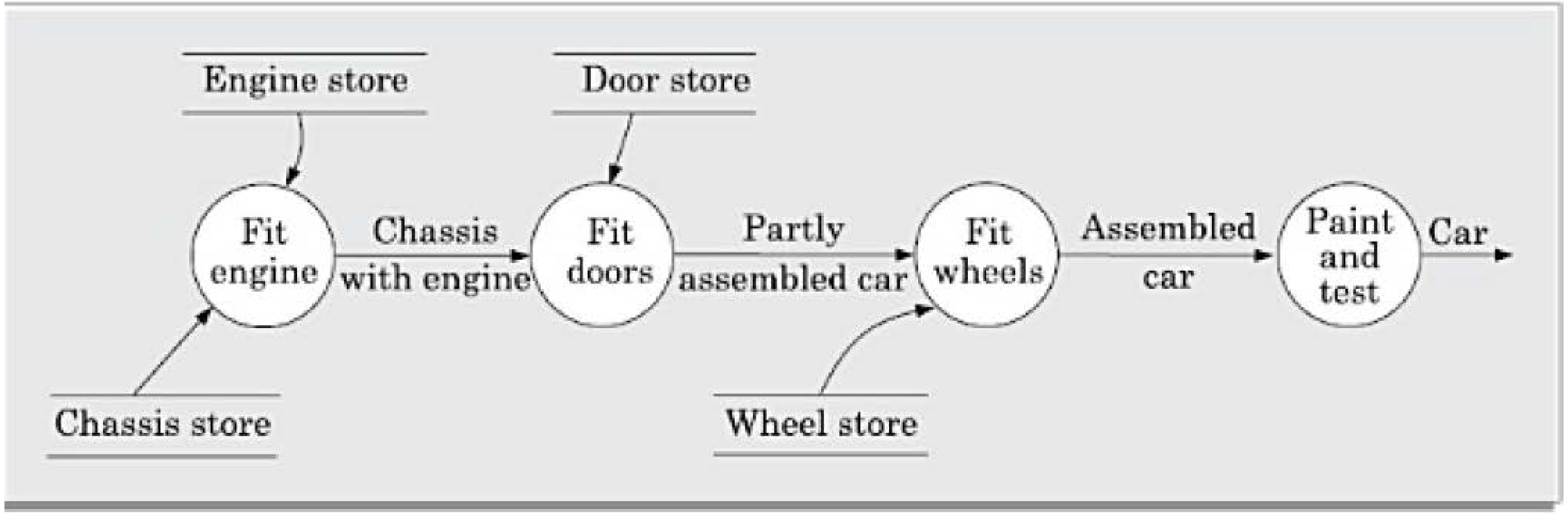


Figure 1.11: Data flow model of a car assembly plant.

Object-oriented Design

- *data hiding* (also known as *data abstraction*)

Evolution of software design techniques

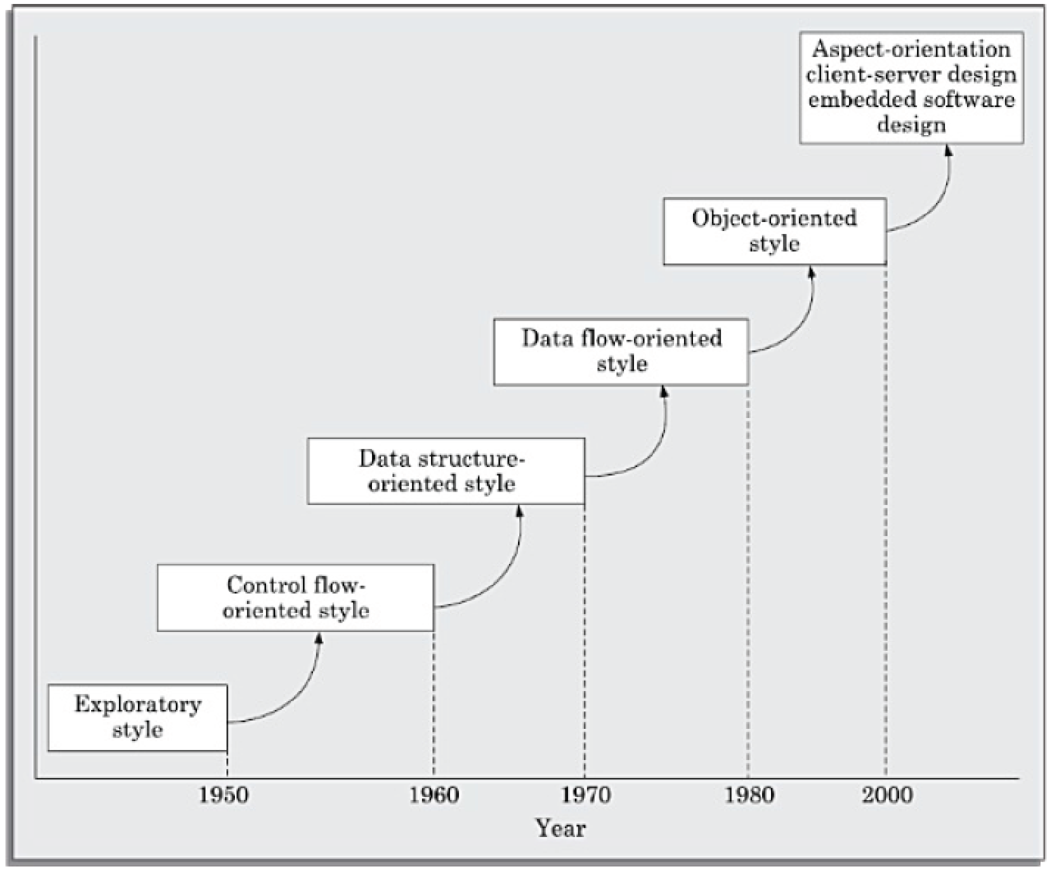

Aspect-orientation client-server design embedded software design
Object-oriented style
Data flow-oriented style
Data structure-oriented style
Control flow-oriented style
Exploratory style
1950
1960
1970
1980
2000
Year

Figure 1.12: Evolution of software design techniques.

Computer systems engineering

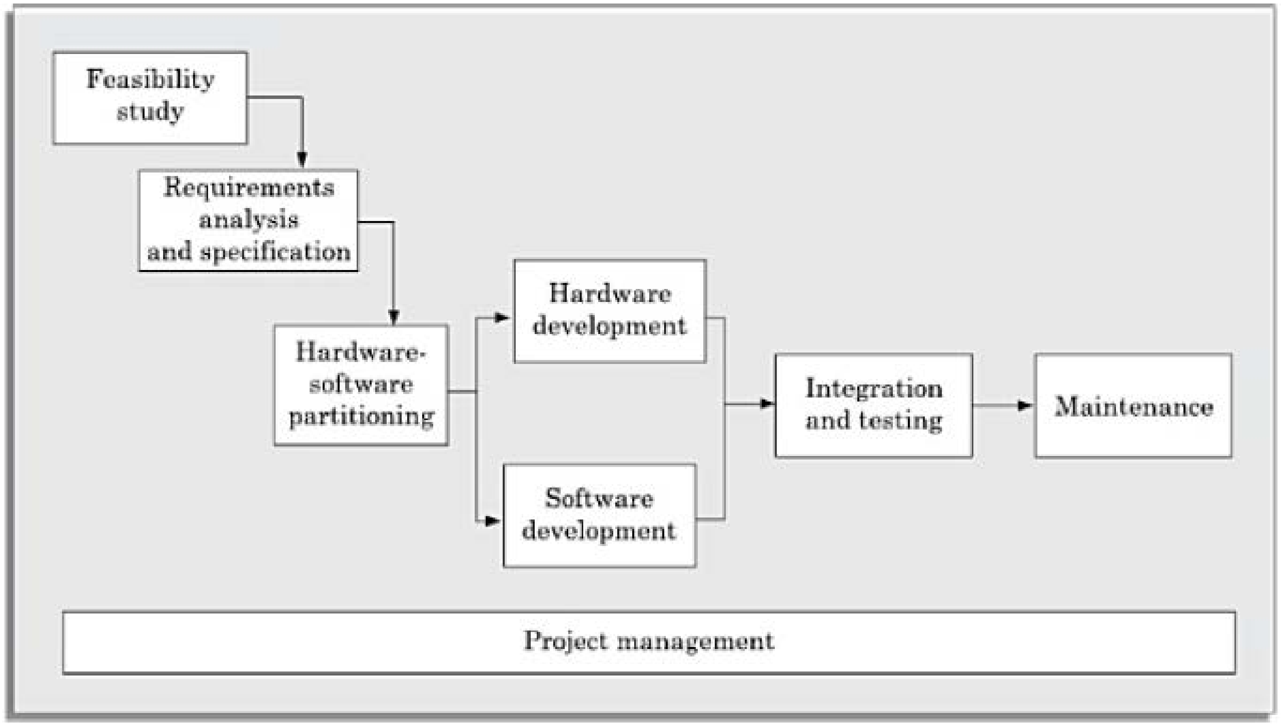


Figure 1.13: Computer systems engineering.