软件技术发展的过去、现在和未来

李必信

bx.li@seu.edu.cn

东南大学软件学院/计算机学院

软件发展的过去、现在和未来

- 1 发展简史
- 2 基本内容
- 3 存在问题
- 4 未来趋势

计算机的发展史是软件和硬件协同演化的历史

计算模型的发展史是“待解问题”和“求解问题”协同演化的历史

计算机软件的发展史是人、语言和工具协同演化的历史

……

1 发展简史

- 1.1 History of Computing
- 1.2 History of Software

1.1 History of Computing

- 1620前—计算的史前史
  （300.000BC - 1619AD ）
  - The prehistory in computing is characterized by **simple methods of numbering, the invention of numerical systems and the birth of mathematics and calculus.** It will take hundreds of years before the concept of the 0 is introduced in the entire world

Clay tablet

(克莱写字板)

engraved with cuneiform

- 1620－1885：古代
  - Antiquity starts at 1620 AD. **The first aids to calculate** are being constructed, no intricate machinery that resembles a computer. But **logarithm and binary logic** (1847)is developed

Schickard calculator

(谢卡特计算器)

- 1886–1946: 前工业时代
  - Pre Industrial era starts at 1886. In this era **the first computing machines are constructed**, the first fully functional mechanical calculators are put on the market. The concept of computers is born but are fully **mechanical**

Difference engine
Babbage

- 1947–2010：工业时代/信息时代
  - Industrial era starts at 1947. This era the computer revolution will go in full gear. **Computers will be produced by the millions each year and becoming smaller than a human finger nail.**

IBM 360 system

1.2 History of Software

- 1.2.1 How It All Started
- 1.2.2 Enter the Information Age

1.2.1 How It All Started

- The earliest **practical form** of programming was probably done by **Jaquard (1804, France)**. He designed a loom that performed predefined tasks through feeding punched cards (穿孔卡) into a reading contraption(装置). This new technology allowed carpets (地毯) and tissues (纱织品) to be manufactured with lower skills and even with fewer people. The little kid sitting under the loom changing rods (杆) and other things vanished. One single person could now handle a loom.

- First there was **Ada Lovelace**, writing a rudimentary program (1843) for the Analytical Machine, designed by Charles Babbage in 1827, but the machine never came into operation.

- Then there was **George Boole** (1815-1864), a British mathematician, who proved the relation between mathematics and logic with his algebra of logic (BOOLEAN algebra or binary logic) in 1847.

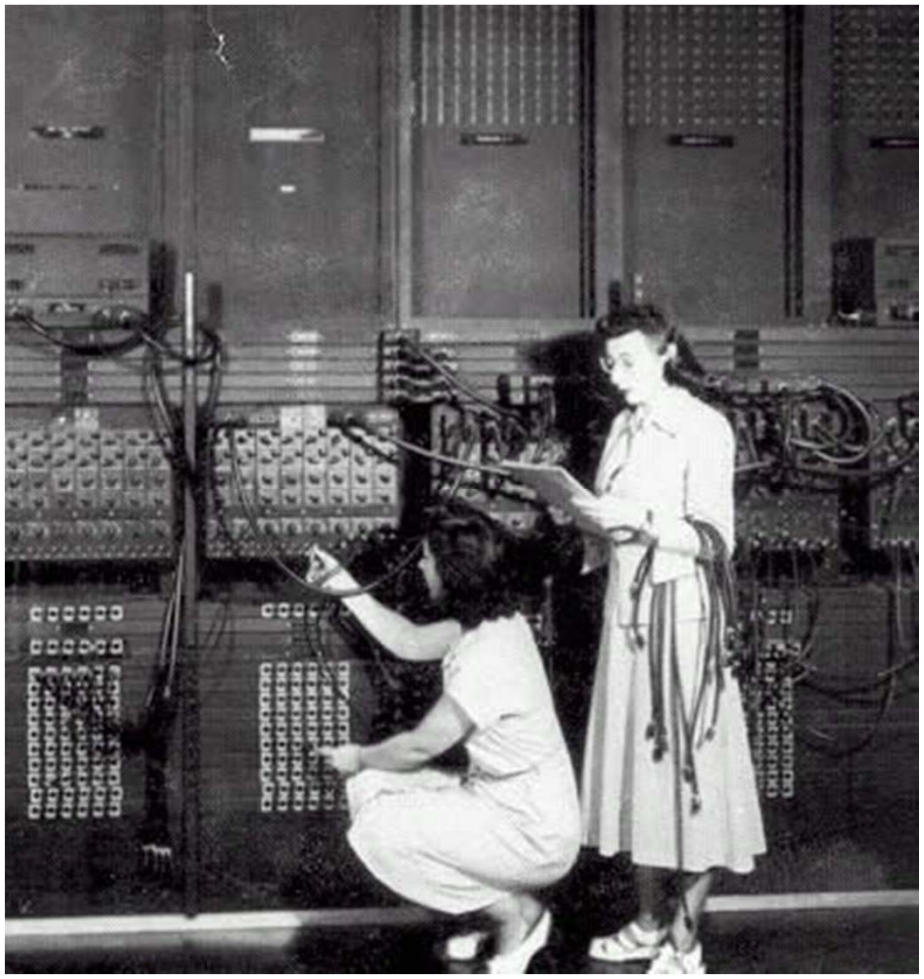
- This meant a breakthrough for mathematics. Boole was the first to prove that logic is part of mathematics and not of philosophy. A big step in thinking too.
- But it will take **one hundred years** before this algebra of logic is put to work for computing.
- It took Claude Shannon (1916-2001) who wrote a thesis (A Mathematical Theory of Communication in the Bell System Technical Journal -**1948**) on how binary logic could be used in computing to complete the **software concept of modern computing**.

Also the hardware needed to make jumps ahead

- 1935-1938: It took Konrad Zuse (1910-1995) to create the first binary programmable computer, **Z1** (world's first program-controlled computer.).

- 1938: The **Bomba**, originally built by Polish engineers to crack the **Enigma** code (German cipher machine), pushed the envelop again.

- 1937-1942: Atanasov and Berry designed the **ABC computer**, a binary computer, as Zuse's was, but now 100% electronic.
- 1944: The **colossus** (巨人) built by people from Bletchley Park (near London, UK) . The Worlds First Electronic Computer
- In 1946, John Mauchly and J Presper Eckert developed the **ENIAC I** (**E**lectrical **N**umerical **I**ntegrator **A**nd **C**alculator). -- **Invention of the Modern Computer**



- Two women wiring the right side of the ENIAC with a new program (US Army photo, from archives of the ARL Technical library, courtesy of Mike Muuss)
- Programming like this was nothing else but rewiring these huge machines in order to use all the options, possibilities and calculations. **Reprogramming always meant rewiring**.

1.2.2 Enter the Information Age

- **Machine Language**
  - First programming was done by typing in 1's or 0's that were stored on different information carriers.
  - By storing these 0's and 1's on a carrier (first used by Karl Suze's X1 in 1938) it was possible to have the computer read the data on any later time. But mistyping a single zero or one meant a disaster because all coding (instructions) should absolutely be on the right place and in the right order in memory. This technology was called absolute addressing.
  - An example: 10100101011101010 11

- In the early 50's programmers started to let the machines do a part of the job. This was called **automatic coding** and made live a lot easier for the early programmers.
- Soon the next step was to have the program to select the proper memory address in stead of using absolute addressing.
- The next development was to combine groups of instruction into so called words and abbreviations were thought up called: **opcodes** (Hopper 1948)

Assembly language

- Opcode works like a shorthand and represents as said a group of machine instructions. The opcode is translated by another program into zero's and one's, something a machine could translate into instructions.
- But the relation is still one to one: one code to one single instruction. However very basically this is already a programming language. It was called: assembly language.

A:=B+C

```
mov1  B,  d0;   move long-word B into register d0
add1  C,  d0;   add
mov1 d0,  A;    and store
```

- **Subroutines**
  - Soon after developing machine languages and the first crude (粗糙的) programming languages began to appear, the danger of inextricable (无法解脱的) and thus unreadable coding became apparent. Later this messy(凌乱的) programming was called: “spaghetti(意大利面条式的) code".
  - One important step in unraveling or preventing spaghetti code was **the development of subroutines**.

- And it needed Maurice Wilkes, when realizing that "a good part of the remainder of his life was going to be spent in finding errors in ... programs", to develop the concept of subroutines in programs to create reusable modules. Together with Stanley Gill and David Wheeler he produced the first textbook on "The Preparation of Programs for an Electronic Digital Computer".
- **The formalized concept of software development** (not named so for another decade) **had its beginning in 1951**.
- Below is an example of how subroutines would work

Start of program	Begin program;
the main "menu"	***Main***;
	Printf ("Hello World");
	DoSomethingElse()
	Printf ("Hello World");
	(end of program)
first subroutine	Function **DoSomethingElse;**
	Add two numbers;
back to the main menu	Return OK
second subroutine	Function ***Printf***(what_to_print)
with a parameter (contents	Open channel to printer interface;
of what to print)	Initialize printer;
	Send "what_to_print" to printer;
	Send page feed to printer;
back to procedure: main	Close printer interface;
	Return OK

- **Fortran**
  - The next big step in programming began when an IBM team under John W. Backus created FORTRAN - FORmula TRANslator **1952**.
  - It could only be used on their own machine, the: IBM 704. But later versions for other machines, and platforms were sold soon after.

- **Programming language**
  - **FORTRAN** soon became called a programming language.
  - So why calling a set of some predefined instructions a programming language?
  - Because some characteristics of a language are met:
    - It must have a vocabulary - list of words
    - It must have a grammar - or syntax
    - It must be unique, both in words and in context

- **Cobol**
  - And by the time when **Cobol**, Common Business Oriented Language, was published in **1960** by the CODASYL (Conference on Data System Languages) committee, (Hopper was a member) the term Computer Language was a fact.
- **Enter "C"**
  - C came into being in the years **1969-1973** and was developed by Dennis Richey and David Kerningham both working at the Bell laboratories. And the magic word was **portability**

A C example program

```
#include <stdio.h>


 int main()
{
  printf( “Hello world!\n" );
}
```

- **Operating Systems (OS)**
  - Parallel at the development of computer languages, Operating Systems (OS) were developed. These were needed because to create programs and having to write all machine specific instructions over and over again was a "waste of time" job.

- So by introducing the OS 'principle' almost all input and output tasks were taken over by the OS. Such as:
  - writing data to and from memory,
  - saving data on disks,
  - writing data to screens and printers,
  - starting tapes,
  - refreshing memory,
  - scheduling specific tasks
  - etcetera.

- **Interpreters and Compilers**
  - An **interpreter** is a computer language that execute instructions that are written in the form of a program. The trick of an interpreter is that it loads the source code and translates the instruction into executable machine language **line by line**. And it does it over and over again any time that program is run.
  - In short a **COMPILER** is: the translator of the source code into computer language.

- **Artificial Intelligence**
  - One of the first and best known is LISP, developed in **1958/9** by the Artificial Intelligence Group at MIT under John McCarthy (McCarty also coined the term AI). Lisp is used in so called **expert systems**: you ask the program answers.
  - Other examples of languages that are used in the AI field are: Prolog (PROgramming LOGic 1970, Alain Colmerauer)， Smalltalk (1979), Algol (1960), and in lesser extend Simula(1967).

- **Enter "OOP"**
  - This is about rewriting routines and programming functionality over and over again for each different part of a program and for each new program again and again.
  - There was a need for **common shared parts** that acted on instructions like a black box. So after some years formal development of software was on its way, it was strongly felt that **portability was one thing but reusability was another**.
  - And history repeats itself: what was written about the subroutines above is also true for this idea of black boxes.

- OOP was introduced by the development of Smalltalk (in 1979) and became known as the Object Oriented Programming method another significant step into the right direction.

- **Which high-level languages are in popular use today?**
  - There are more than 300 other high-level languages such as PASCAL, FORTH, PL/I, LISP, SMALLTALK, APL, C/C++ and PROLOG to name but a few. Many of these were developed for particular applications, while others are developments or improvements of existing languages.

- 1957 FORTRAN
- 1958 ALGOL
- 1960 LISP
- 1960 COBOL
- 1962 APL
- 1962 SIMULA
- 1964 BASIC
- 1964 PL/I
- 1966 ISWIM
- 1970 Prolog
- 1972 C
- 1975 Pascal
- 1975 Scheme
- 1977 OPS5
- 1978 CSP
- 1978 FP
- 1980 dBASE II
- 1983 Smalltalk-80
- 1983 Ada
- 1983 Parlog
- 1984 Standard ML
- 1986 C++
- 1986 CLP(R)
- 1986 Eiffel
- 1988 CLOS
- 1988 Mathematica
- 1988 Oberon
- 1990 Haskell
- 1994 Java
- ...

- **Standardization**
  - A common question is: Is there a standard version of any higher level programming language? The answer is NO
  - However, an important problem with higher programming languages is that they are seldom portable. This means that a program written in FORTRAN or BASIC for a specific computer will not always run on another type of computer from a different brand. Even within a certain brand this could be a problem.
  - The cause of all this is because most manufacturers are creating their own standards in programming languages.

2. 软件开发的基本内容

- 2.0 基本术语回顾
- 2.1认识软件、软件开发、软件工程
- 2.2软件开发的文档（规约、代码）
- 2.3软件开发的语言（程序设计语言）
- 2.4软件开发的基本方法
- 2.5软件开发的生命周期模型（过程模型）
- 2.6软件开发的支撑环境和工具
- 2.7软件开发的其他技术

2.0 基本术语回顾

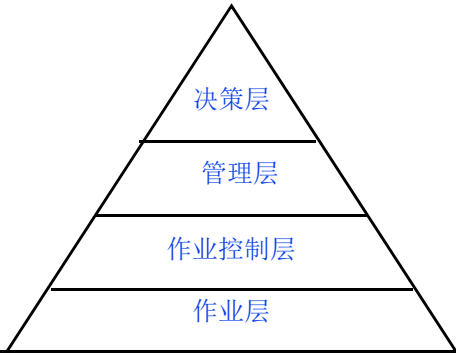
- 计算：解决问题的手段
- 算法：解决问题的步骤
- 指令：表示算法的符号
- 程序：程序＝算法＋数据结构
- 软件：软件＝程序＋文档
- 软件开发：构造软件的过程
- 软件工程：用工程方法构造软件

2.1 认识软件、软件开发、软件工程

- A **program** is a sequence of instructions that can be executed by a computer (CPU) that performs a certain task intentioned by the programmer.
- **Software** is the set of directions that enables computer hardware to perform useful work.
  - 计算观—计算的实质就是寻找一种可机械执行的算法。例如，著名的计算模型：图灵机（被公认为现代计算机的祖先）。
  - 组成观—软件可以用两个简单的算式来概括：
    - （1）软件＝程序＋文档
    - （2）程序＝算法＋数据结构

- 工具观—软件是一种代替、强化和延伸人类思维能力的工具。
- 逻辑观—计算机的思考方式是形式逻辑、形式逻辑通过建立一种可机械实现的符号演算机制，模拟了人类思维中的一类精确推理过程（如形式化语言、有限状态自动机理论）。
- 模型观—从模型论的角度看，整个软件系统是客户业务实现的一个模型，特别是软件的研制过程就是一个建模和模型间的演绎过程。
- 工程观—软件工程原理
- 对象观—OOA&D理论认为，现实世界是由对象构成的，对象是具有特殊属性和行为的实体，对象间具有共性、层次性、继承性和关联性，反应了客观世界从普遍到一般、从个性到共性及相互制约的运动规律。从这个角度，软件的开发可视为“类”的抽象及其关联的建模过程，软件的运行是对象的实例化及其状态的演变过程。

- 生存期—根据ISO/IEC12207软件生命周期标准，一个通用的软件生命周期模型：（1）定义阶段（计划、需求分析）；（2）开发阶段（概要设计、详细设计、编码、集成测试、试运行和验收）；（3）运行维护
- 质量观—一种三维软件质量衡量:（1）软件运行（正确性、可用性、强壮性、安全性、性能）；（2）软件维护（可理解性、可扩展性、易维护性、灵活性）；（3）软件移植（可移植性、可复用性、互操作性）。
- 人机工程观—好的系统应该易于学习、理解和掌握；界面应该简洁。
- 集成观—各种软件、硬件资源的集成
- 结构观—



- **The software essence**
  - Complexity
  - Conformity[conformance to]
  - Changeability[software evolution]
  - Invisibility
- **The software accidents**
  - Stakeholders
  - Process
  - Modeling language and tools

Software Life Cycle

- Coarse granularity
  - 1 Analysis
  - 2 Design
  - 3 Implementation
  - 4 Maintenance
- Refined granularity
  - 1 Application Requirements determination
  - 2 Software Requirements specification
  - 3 Architectural design
  - 4 Detailed design
  - 5 Implementation
  - 6 Integration
  - 7 Maintenance

- **Software engineering** is the profession that creates and maintains software applications by applying technologies and practices from computer science, project management, engineering, application domains, and other fields.

- The term software engineering first was used in the late 1950s and early 1960s. Programmers have always known about civil, electrical, and computer engineering and debated what engineering might mean for software.

- The NATO Science Committee sponsored two conferences on software engineering in 1968 (Garmisch, Germany) and 1969, which gave the field its initial boost. Many believe these conferences marked the official start of the profession.

- History of Software Engineering
  - 1945 to 1965: The origins
  - 1965 to 1985: The software crisis
  - 1985 to present: No silver bullet

2.2 软件开发的文档（规约、代码）

- 调查报告（可行性分析）
- 应用需求分析文档 （SDL document）
- 软件需求规约 （use case, domain model）
- 软件设计规约（UML Interaction diagrams）
- 源程序代码 （Java code etc）
- 测试计划、测试数据和测试报告等

Modeling techniques

2.3 程序设计语言

- Procedural programming language（Fortran）
- Structured programming language （Algol 60, Pascal, C）
- Imperative programming language （Basic, C, SQL）
- Declarative programming language （MU-Prolog, NU-Prolog, CLP(R), Lygon, Mercury, and HAL）
- Logic programming language （Prolog）
- Functional programming language （Haskell, Lisp, ML, Scheme）

- Script programming language （Perl）
- Literate Programming language （WEB, CWEB, FWEB, MLWEB）
- Object oriented programming language （SmalltalkC++, Java）
- Aspect oriented programming language （AspectJ）
- Concurrent programming language （Occam, Ada, CSP, CCS）
- Component-oriented programming language （ADL,CDL, IDL）
- Interactive programming language （Basic, Python, B）

2.4 软件开发方法

- 模块化方法：Parnas方法(1972)
- 面向数据结构的软件开发方法: Warnier方法(1974), Jackson方法(1975),
- 结构化方法：Yourdon方法 (1978)
- Top-down programming
- Bottom-up programming
- 问题分析法PAM: 是80年代末由日立公司提出的一种软件开发方法。PAM方法希望能兼顾Yourdon方法、Jackson方法和自底向上的软件开发方法的优点，而避免它们的缺陷。
- OOA/OOD/OOP (1986)
- Component-based software development (1990)
- Extreme Programming (1999)

For example: Structured vs. Object Oriented

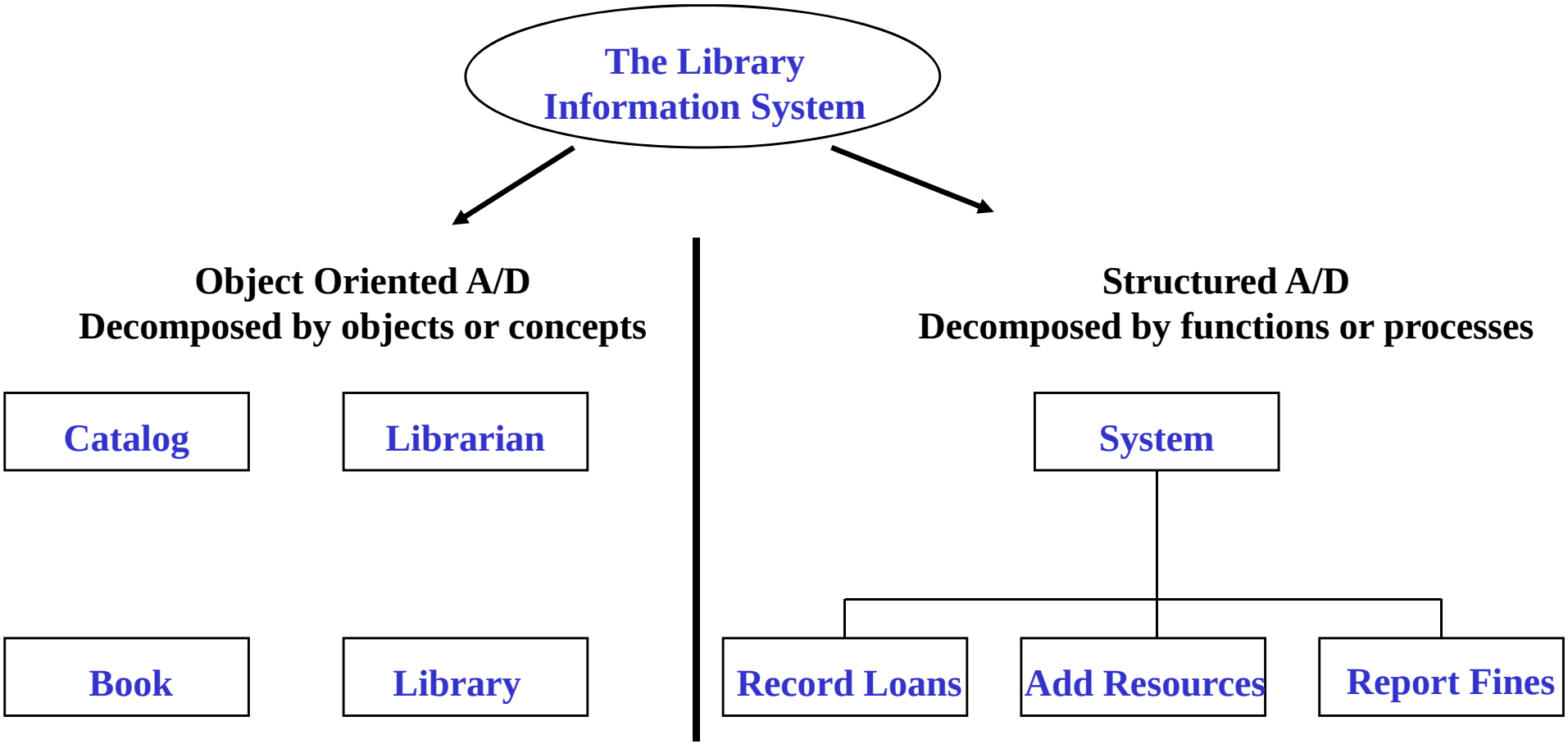
- **结构化方法：**强调结构的合理性，以及所开发的软件结构合理性，由此提出了一组提高软件结构合理性的准则，如分解和抽象、模块的独立性、信息隐藏等。针对不同的开发活动、它有结构化分析、结构化设计、结构化编程和结构化测试。
- Modeling Techniques
  - Functional Decomposition
  - Data Flow Diagrams (DFDs)
  - Entity Relationship Diagrams (ERDs)
  - State Diagrams (STDs)
  - Data Dictionaries
  - Structure Charts

- Problems
  - Sequential and transformational
  - Inflexible solutions
  - No reuse

- 面向对象方法：The focus is on:
  - Responsibility and Roles
  - Co-operation and Collaboration between objects
- Modeling Techniques
  - OOD G. Booch(1991)
  - HOOD HOOD Technical Group(1993)
  - OOA S. Shlaer and S. Mellor(1988, 1992)
  - OOA/OOD T. Coad and Y. Yourdan(1991)
  - OMT J. Rambaugh, M. Blaha, …(1991)
  - OOSE I. Jacobson, …(1992)
  - FUSION D. Coleman, …(1994)
  - UML1.0-1.5: Booch, Rambaugh, Jacobson(1998-2003)

- In structured approach the system decomposition (divide-and-conquer) is primarily by function or process, resulting in a hierarchical breakdown of processes composed of sub-processes. This is a solution oriented approach.

- In Object Oriented approach the problem space is decomposed by objects. This is a problem domain oriented approach.


The Library
Information System
Object Oriented A/D
Decomposed by objects or concepts
Catalog
Librarian
Book
Library
Structured A/D
Decomposed by functions or processes
System
Record Loans
Add Resources
Report Fines

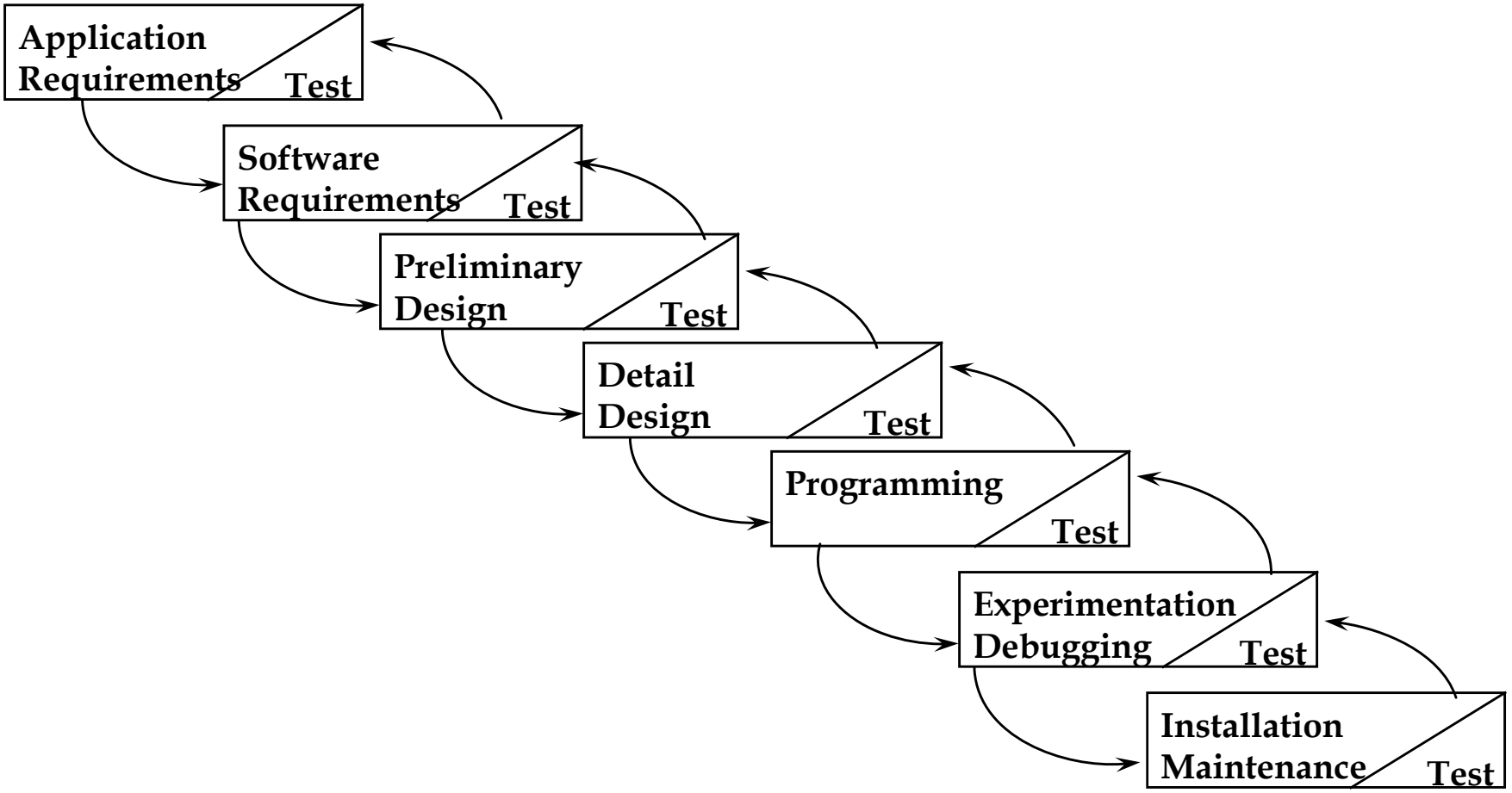
2.5 软件开发的过程模型

- Top-Down Model
- Bottom Up
- **Waterfall model**
- **V model**
- **Spiral model**
- Chaos model
- **Prototyping**
- Evolutionary prototyping
- **Iterative and Incremental development**
- **Extreme Programming**
- **Rational Unified Process**

Process model

- Linear
  - The Waterfall
  - The W
- Spiral

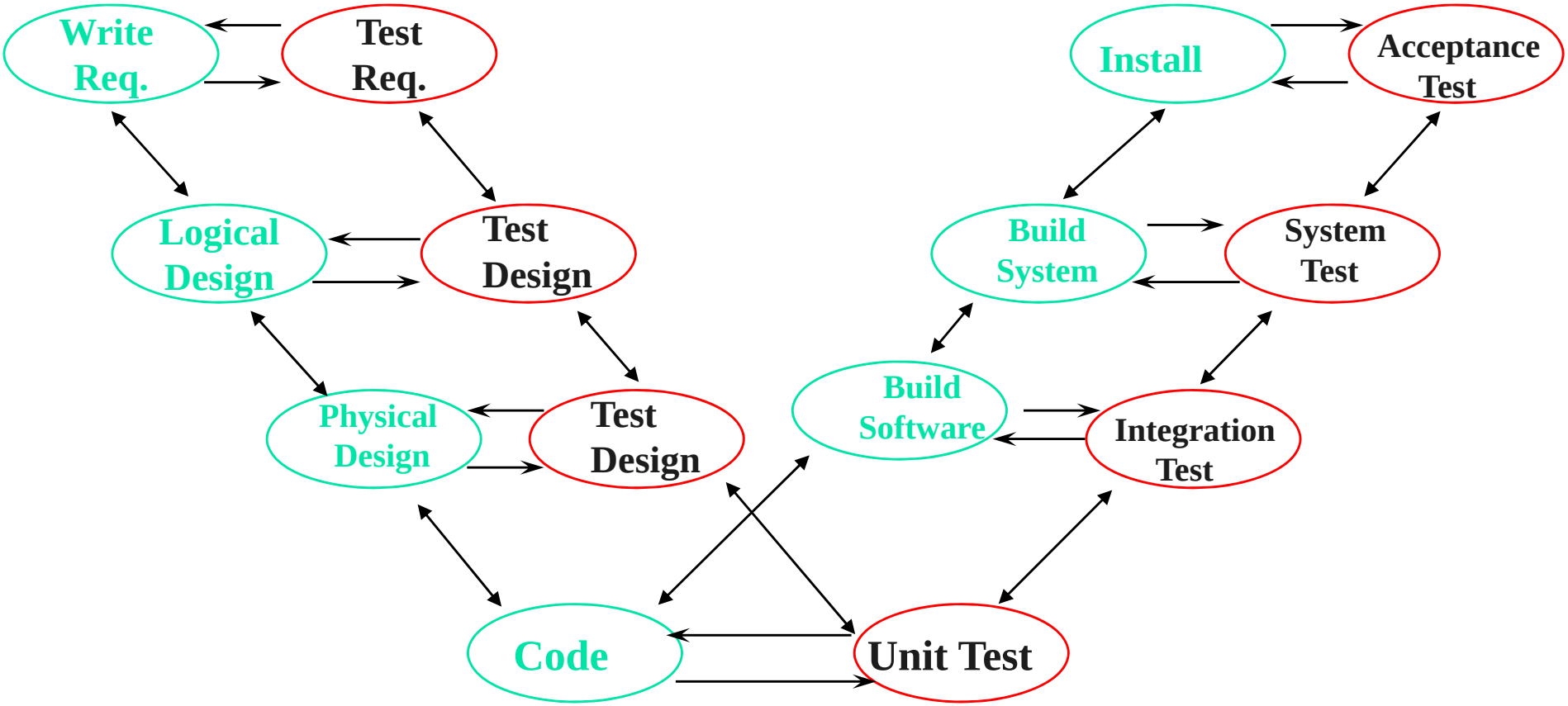
The Waterfall Model (B.W. Boehm)



Phases

- 1 Application Requirements determination
- 2 Software Requirements specification
- 3 Architectural design
- 4 Detailed design
- 5 Implementation
- 6 Integration
- 7 Maintenance

W - Whole Life Cycle

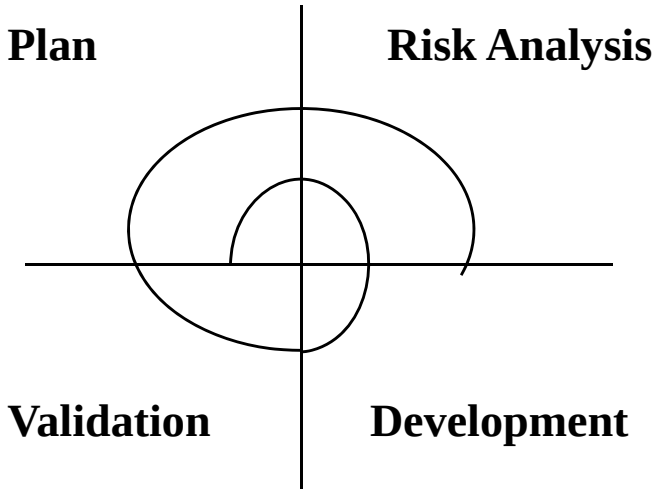


Characteristics and Problems of the Process

- Linear, sequential phases
- No overlap between phases
- All requirements ⇒all designs ⇒ all implementations ⇒ all maintenace…
- More later the error found, the more expensive the maintenance cost.

Spiral (I ** 3) Life Cycle

- Antithesis of Waterfall
- Iterative - many refinements
- Incremental - moving forward:
  - functions, features, quality
- Integrative - continual integration of work products
- Non-linear, evolving prototypes
- More time spent in Analysis & Design
- Less time spent in Development



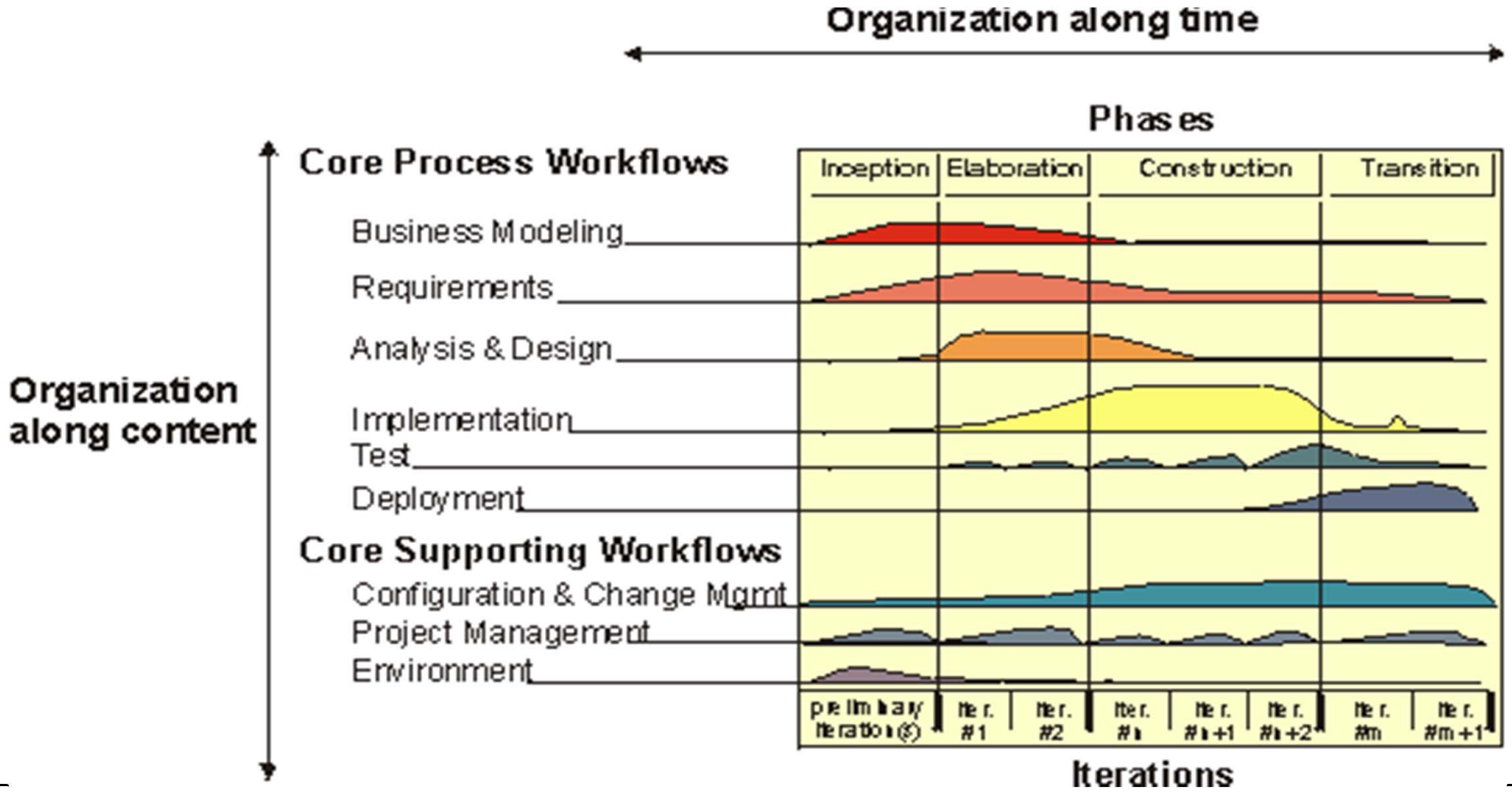
1:制定计划

2:风险分析

3:实施开发

4:客户评估

RUP：Rational Unified Process



- A phase is the span of time between two major milestones of the process in which a well-defined set of objectives are met, artifacts are completed, and decisions are made whether to move into the next phase. Four phases:
  - Inception - Establish the business case for the project
  - Elaboration - Establish a project plan and a sound architecture
  - Construction - Grow the system
  - Transition - Supply the system to its end users
- An iteration represents a complete development cycle, from requirements capture in analysis to implementation and testing, that results in the release of an executable project.

Characteristics of the Process

- Iterative Process
- Architecture Centric
- Software Modeling
- Use Case Driven
- Supports Object-Oriented Techniques
- Scalable
- Ongoing Quality Control & Risk Management

2.6 工具和环境

- **Software tools come in many forms**:
  - **Revision control**: SCM, SCCS, RCS, CVS, SVN, SourceSafe, PVCS, ClearCase, Bonsai
  - **Compilation and linking tools**: GNU toolchain (Make, automake, gcc), Microsoft Visual Studio
  - **Search**: grep, find
  - **Editors**: emacs, vi, Utraedit
  - **Scripts**: Shell, Perl
  - **Parsing**: Lex, Yacc
  - **Bug Databases**: gnats, Bugzilla, Trac
  - **Debuggers**: gdb, GNU Binutils
  - **Integrated development environments (IDEs)**: Eclipse

2.7 其他软件工程技术

- 框架技术(framework)
- 中间件技术(middleware)
- 软件体系结构(software architecture)
- 设计模式(design pattern)
- 软件配置管理(software configuration management)（DCM）
- 软件维护(software maintenance)
- 软件测试(software testing)
- 软件工程管理(software engineering management)

软件工程管理

- **Leadership** （领导阶层）
  - Coaching
  - Communication, SEs easily ignore this
  - Listening
  - Motivation
  - Vision, SEs are good at this
  - Example, everyone follows a good example best

- **Personnel management**
  - Hiring, getting people into an organization
  - Staffing, getting people onto a project
  - Training
  - Evaluation

- **Project management**
  - Goal setting
  - Customer interaction (Rethink)
  - Estimation
  - Risk management
  - Change management
- **Process management**
  - Processes
  - Software development processes
  - Methodologies
  - Metrics

3. 存在问题

- 技术更新太快：技术不稳定
- 缺乏统一标准：
- 生产效率低：
- 软件测试、维护困难：
- 软件质量不高：

4. 未来发展趋势

- 软件技术本身
- 软件生产
- 软件语言
- 软件支撑技术

4.1 软件技术本身

- 网络化：以Internet为中心的各种软件开发技术和方法将成为主流。web service, protocol engineering
- 智能化：人类的梦想还会继续。agent技术，语义Web，智能软件
- 本质化：基于Ontology开发才是本质
- 本地化：从国家安全和国家科学技术发展的角度。自主产权（操作系统、核心支撑软件）
- 人性化：以人为本
- 量子软件技术：…

4.2 软件生产

- **软件生产(大系统)**
  - 标准化：建模（UML）、语言（Java）、过程（RUP）、方法（OOA/OOD, AOP）
  - 智能化、自动化：形式化方法（Z，Refinement Calculus）
  - 规模化、批量化：软件重用技术、构件构架技术、SPL（Software Product Line）技术

- **软件生产(小系统)**
  - 快捷化：敏捷开发（XP）

4.3 软件语言

- PL：通用语言⇒专用语言
- SDL：规约语言
- ADL/CDL/IDL：软件体系结构、构件、接口语言
- 网络语言：
- 智能语言：

4.4 软件支撑技术

- 操作系统：Linux
- 软件开发环境和工具：Eclipse
- 软件过程改进：RUP的下一代
- 软件配置管理：动态重配置技术（DRCM）
- 软件分析与测试：组装测试、智能测试、演化测试和变异测试
- 软件质量提升：软件开发早期阶段的质量保证（model checking, specification testing, symbolic execution）

第1周作业

1. 说说你对软件工程的理解。
2. 软件工程在计算机学科的作用是什么？