软件工程的发展

刘钦

Reference

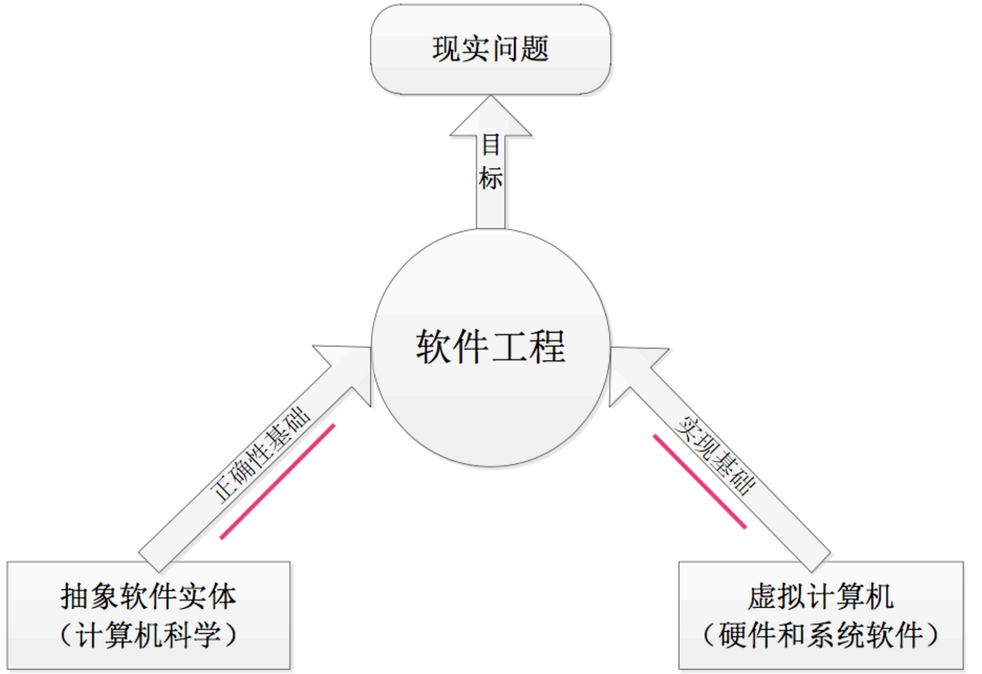
- 《软件工程通史:1930-2019》，Capers Jones
- 《Software Pioneers:Contributions to Software Engineering》，Springer

Outline

- Three Environment factors of SE
- 1950s
- 1960s
- 1970s
- 1980s
- 1990s
- 2000s
- 2010s later
- Summary

ThreeEnvironment Factors of SE

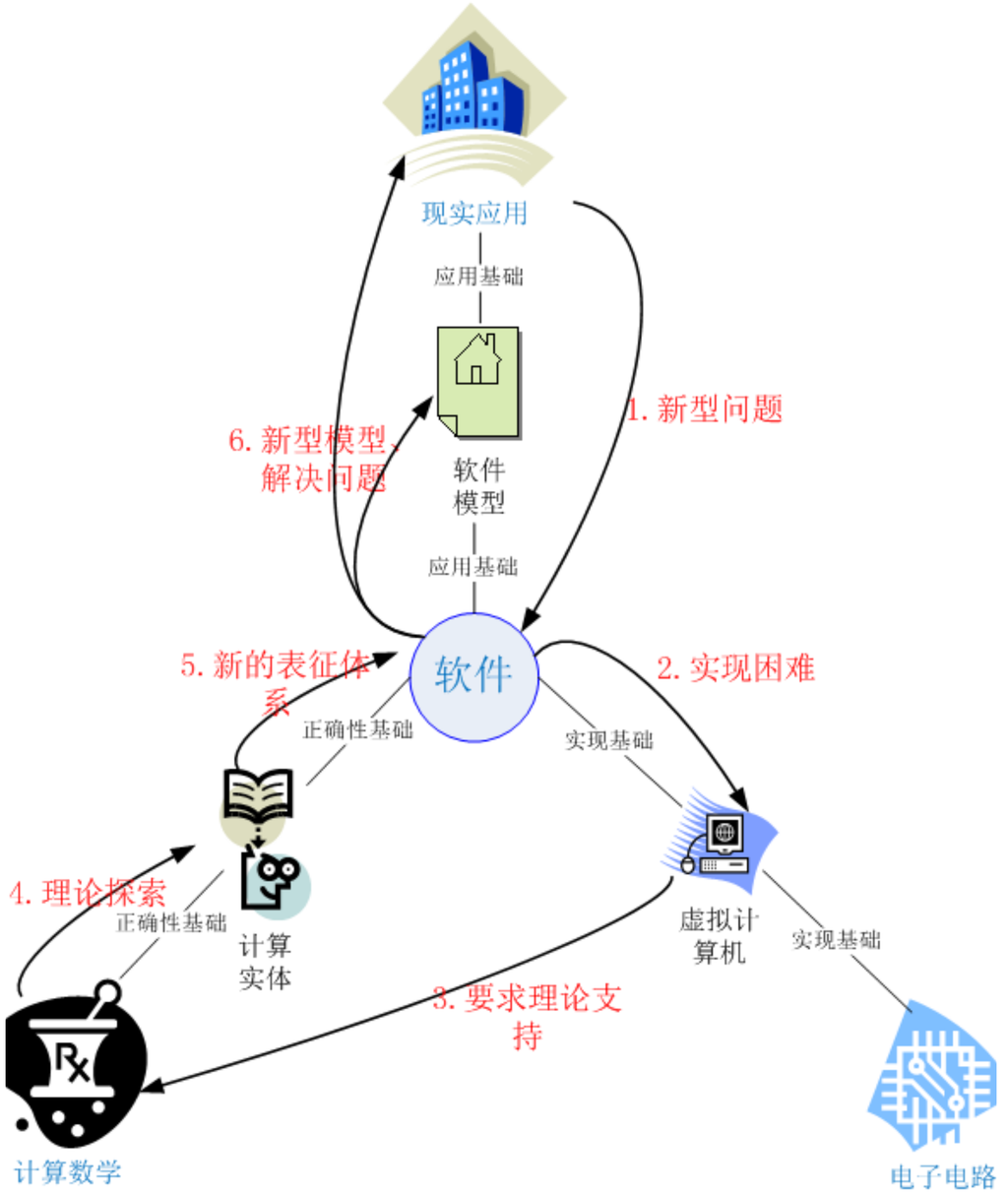
- Foundation
  - Abstract software entity
  - Virtual machine
- Target
  - Problems of reality



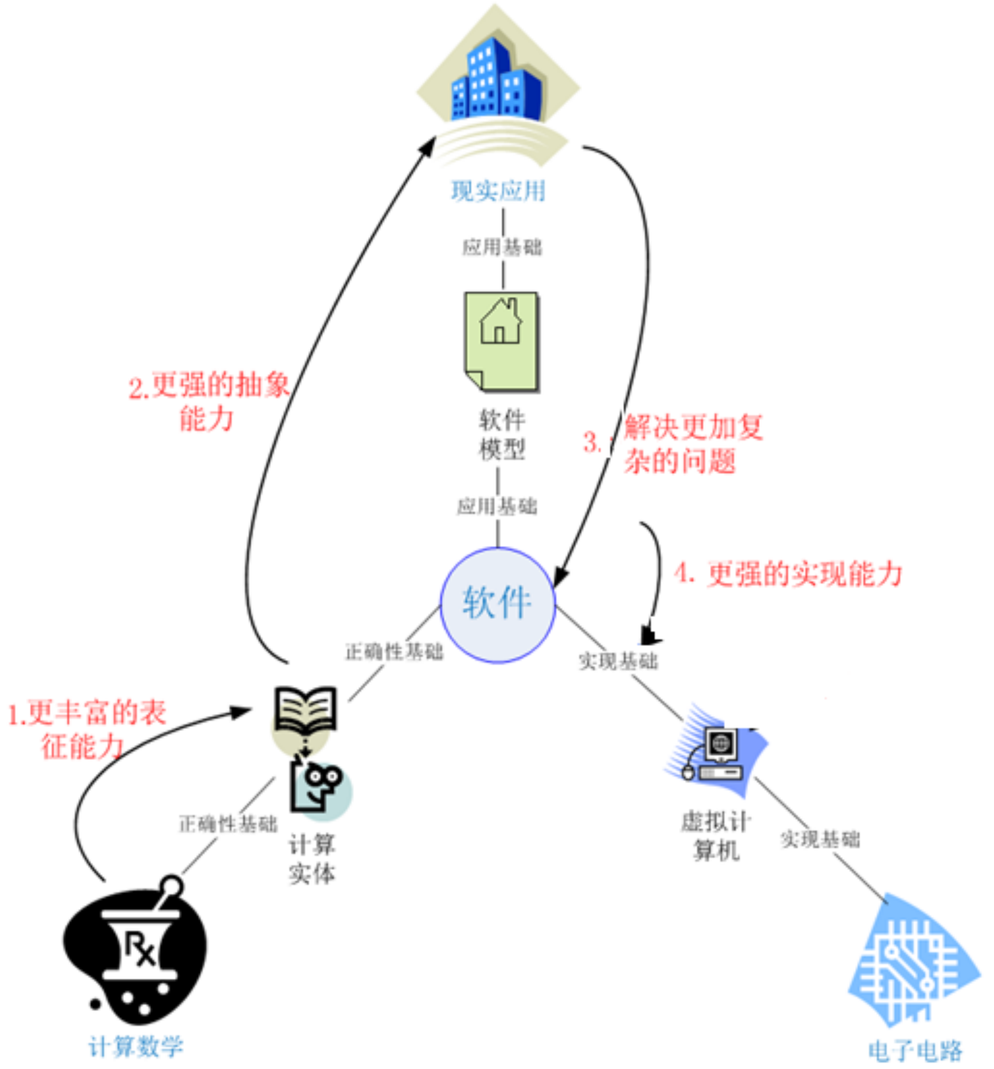
计算指令
汇编指令
函数级程序单位
面向对象程序单位
构件、体系结构、服务…

裸机硬件
基本操作平台（OS）
系统辅助环境(编译器)
应用辅助环境…

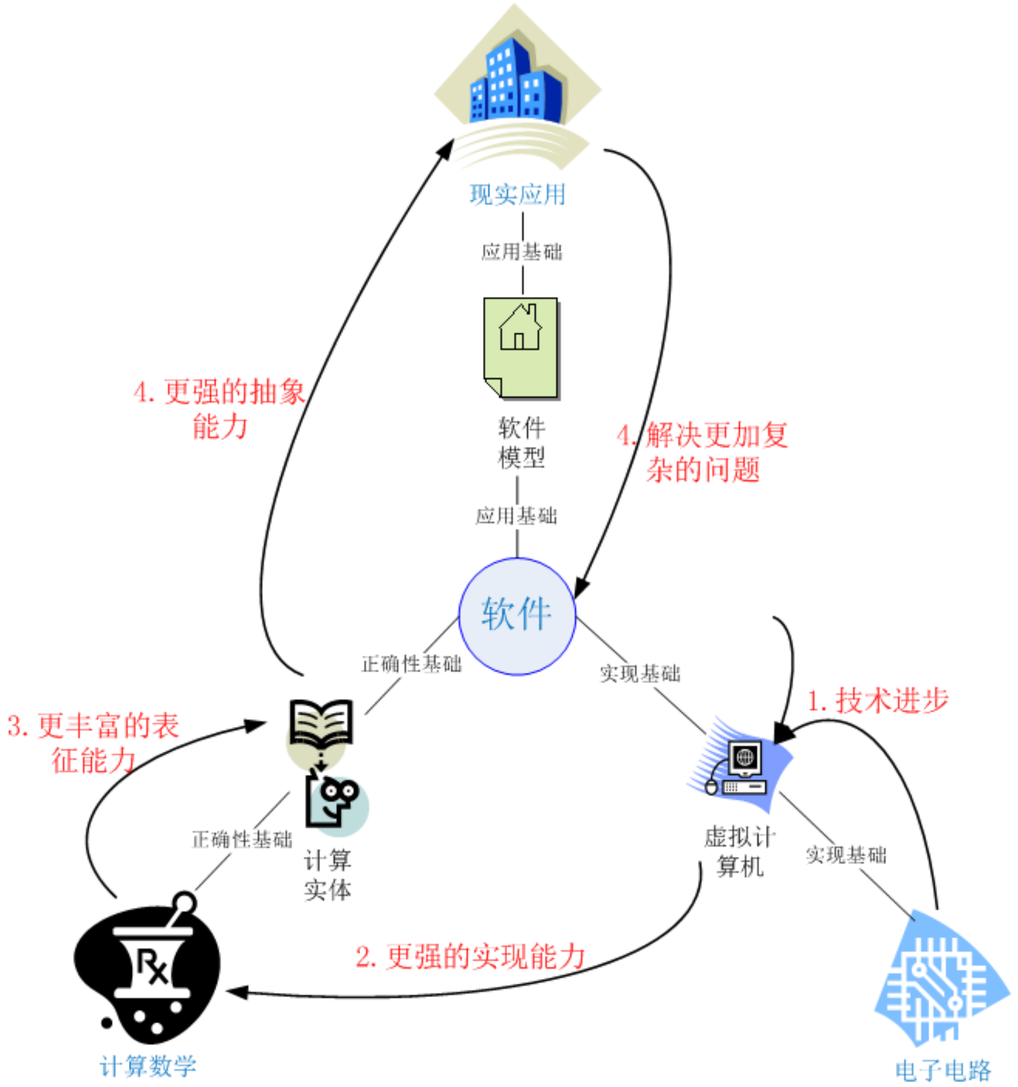
Driving Force of SE -- Problems of Reality

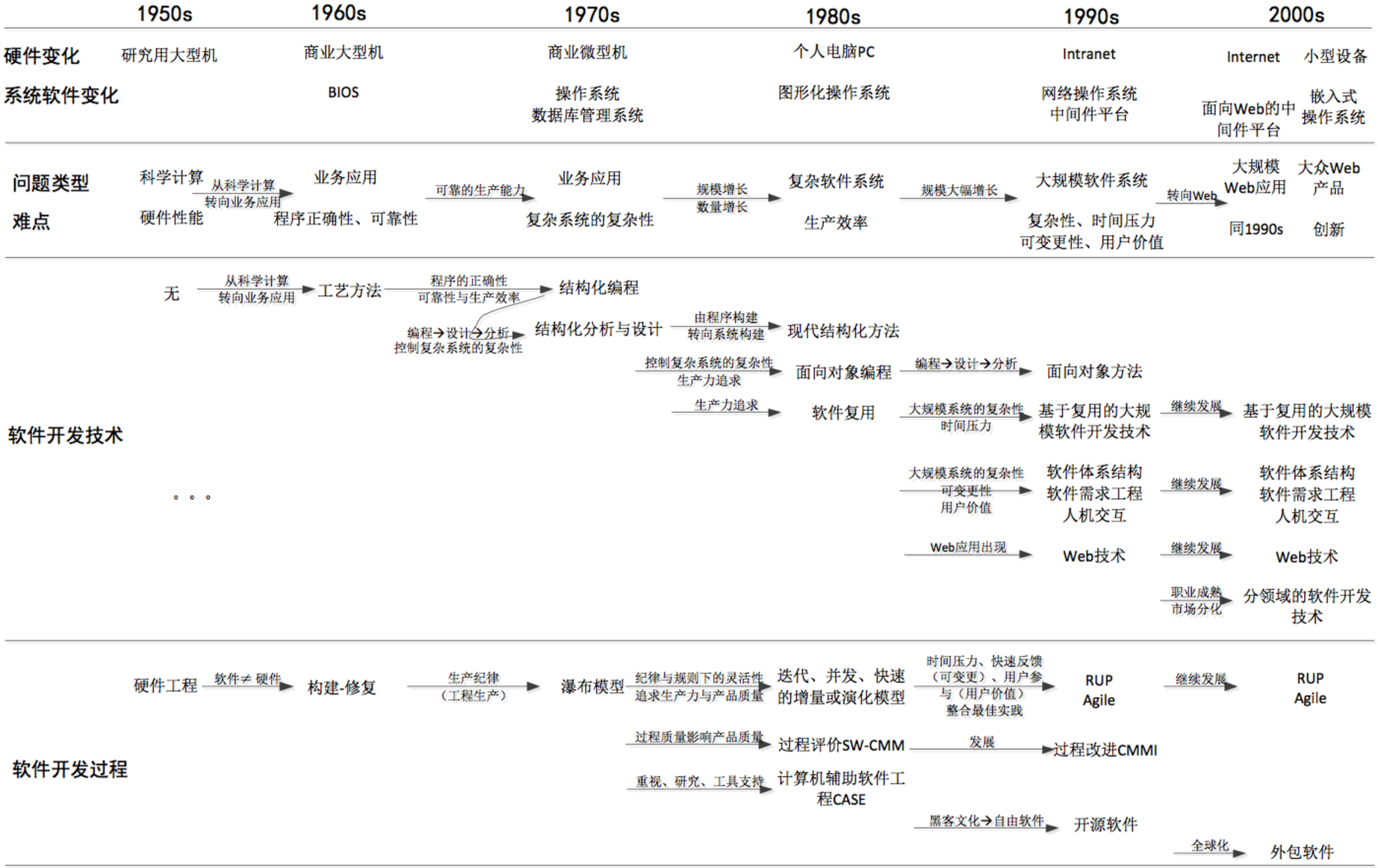


Driving Force of SE
-- Abstract Software



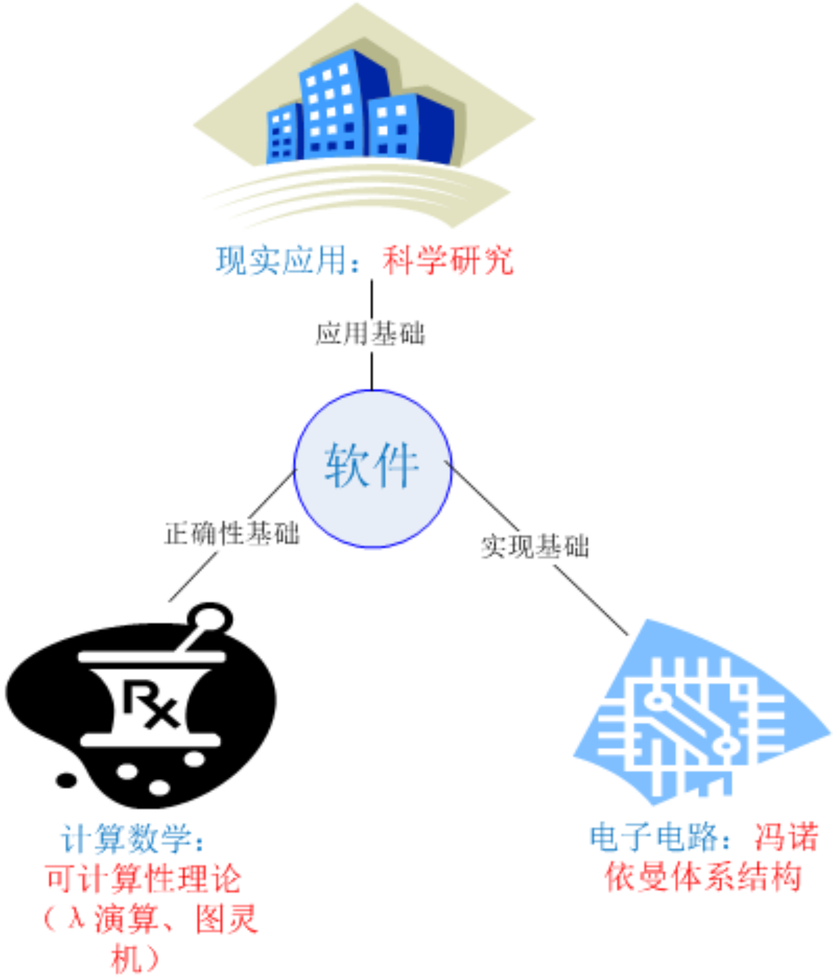
Driving Force of SE
-- Virtual Machine




1950s
1960s
1970s
1980s
1990s
2000s
硬件变化
研究用大型机
商业大型机
商业微型机
个人电脑PC
Intranet
Internet
小型设备
系统软件变化
BIOS
操作系统
数据库管理系统
图形化操作系统
网络操作系统
中间件平台
面向Web的中间件平台
嵌入式操作系统
问题类型
科学计算
从科学计算转向业务应用
业务应用
可靠的生产能力
业务应用
规模增长
数量增长
复杂软件系统
规模大幅增长
大规模软件系统
转向Web
大规模Web应用
大众Web产品
难点
硬件性能
程序正确性、可靠性
复杂系统的复杂性
生产效率
复杂性、时间压力
可变更性、用户价值
同1990s
创新
软件开发技术
无
从科学计算转向业务应用
工艺方法
程序的正确性
可靠性与生产效率
结构化编程
编程→设计→分析
控制复杂系统的复杂性
结构化分析与设计
由程序构建
转向系统构建
现代结构化方法
控制复杂系统的复杂性
生产力追求
面向对象编程
编程→设计→分析
面向对象方法
生产力追求
软件复用
大规模系统的复杂性
时间压力
基于复用的大规模软件开发技术
继续发展
基于复用的大规模软件开发技术
。。。
大规模系统的复杂性
可变更性
用户价值
软件体系结构
软件需求工程
人机交互
继续发展
软件体系结构
软件需求工程
人机交互
Web应用出现
Web技术
继续发展
Web技术
职业成熟
市场分化
分领域的软件开发技术
软件开发过程
硬件工程
软件≠硬件
构建-修复
生产纪律
（工程生产）
瀑布模型
纪律与规则下的灵活性
追求生产力与产品质量
迭代、并发、快速的增量或演化模型
时间压力、快速反馈（可变更）、用户参与（用户价值）整合最佳实践
RUP
Agile
继续发展
RUP
Agile
过程质量影响产品质量
过程评价SW-CMM
发展
过程改进CMMI
重视、研究、工具支持
计算机辅助软件工程CASE
黑客文化→自由软件
开源软件
全球化
外包软件

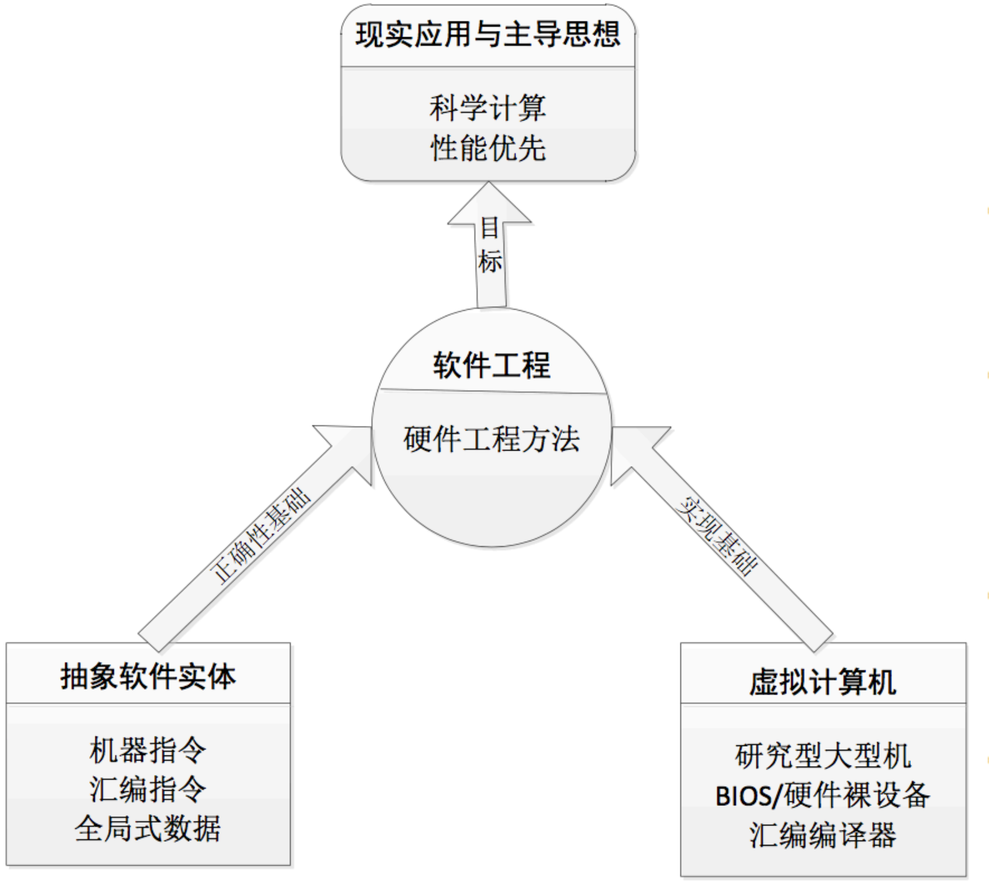
Before 1950s - Software is one part of hardware

- Scientific research(Especially military purpose)
- Computability theory(Alonzo Church’s lambda calculus and Alan Turing’s Turing machines)
- Von Neumann Architecture
- Software is one part of hardware



1950s - Software Engineering is Same as Hardware Engineering

- Machine-Centric Application
- Machine language programs (1GL) & Assembly language (2GL)
- Research Mainframe & BIOS
- Hardware-Oriented Development Process



1950s - Progress

- Virtual Machine
  - First commercial computer (UNIVAC I)
  - Machine-Centric
- Abstract Software Entity
  - Grace Hopper, was one of the first programmers of the Harvard Mark I computer, a pioneer in the field, developed the first compiler, around 1952, for a computer programming language.
  - Programmers of early 1950s computers, notably UNIVAC I and IBM 701, used machine language programs, that is, the first generation language (1GL). 1GL programming was quickly superseded by similarly machine-specific, but mnemonic, second generation languages (2GL) known as assembly languages or "assembler".

1950s - Machine Centric

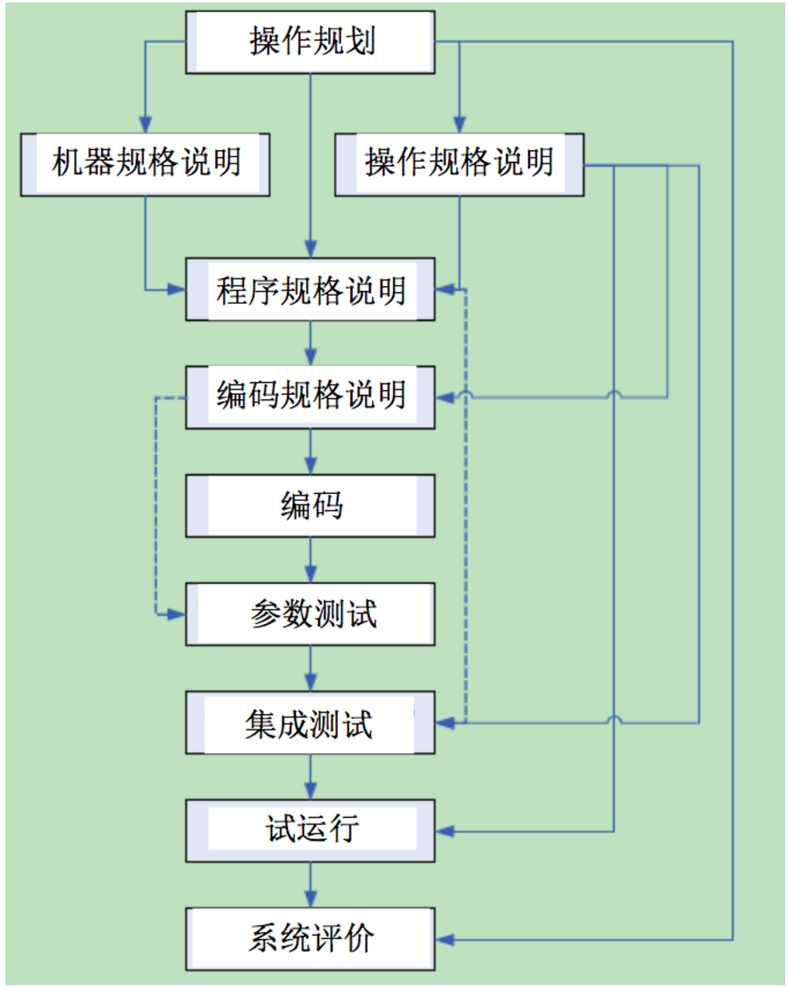
- 我工作的第一天（1950s），我的主管向我展示了通用电力ERA1103计算机，这个家伙足足占满了一个大房间。他对我说：“听着，我们每小时要为这台计算机支出600美元，而每小时只需为你支出2美元，我想你知道该怎么做了”
- -- [Boehm 2006]

1950s - Software development method

- 在这一时期，人们的主要集中力放在硬件上，所以没有出现对软件开发专门方法与技术的需求，也就没有出现被普遍使用的软件开发方法与技术。

1950s - Software Development Process

- SAGE Project -- Semi-Automated Ground Enviroment for U.S. and Canadian air defense
- 1 MLOC air defense system, real-time, 365*24，user-intensive
- Successful development of highly unprecedented system
- Hardware-oriented waterfall-type process

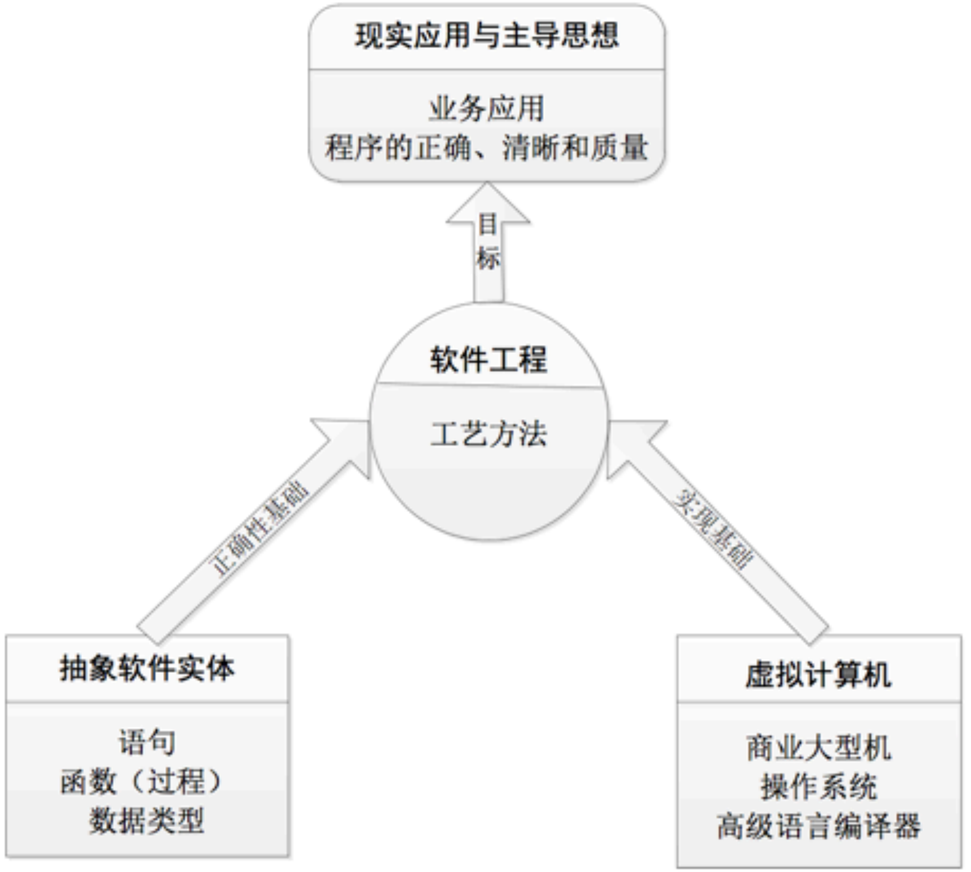


1950 - Summary

- Scientific computation
- Machine-centric Scientific mainframe
- 1GL, 2GL
- Software Engineering is same as Hardware Engineering
- Hardware-Oriented development process
- Emphasize review and test
- Scientist and Hardware Engineer

1960s - Software is not Hardware

- Business Application
- 3GL
- Business Mainframe
- Software Crafting



1960s - Progress

- Virtual Machine
  - Better infrastructure: OS, compilers (“function” concept), utilities
  - Product Families: OS-360, CAD/CAM, math/statistics libraries
  - Some large successes: Apollo, ESS, BofA check processing
  - 信用卡、ATM
- Abstract Software Entity
  - Later in the 1950s, assembly language programming, which had evolved to include the use of macro instructions, was followed by the development of "third generation" programming languages (3GL), such as FORTRAN, LISP, and COBOL.
  - ASCII美国信息交换标准码

两个重要的诉讼案

- 通过一项诉讼案裁定ENIAC专利不再有效（1973裁决）
  - 使得计算机体系结构从此进入公共领域
- 作为反垄断诉讼的结果，IBM同意分类定价软件
  - 使得软件不再依附于计算机硬件，独立存在

1960s - Application-Centric

- Business application
- Batch process

1960s - Software is not Hardware (1)

- 软件与现实世界关系更加密切，对需求的规格化更加困难
  - [Royce1970]提到“生产500万美元的硬件设备，30页的规格说明书就可以为生产提供足够多的细节，生产500万美元的软件，1500页的规格说明书才可以获取相当的控制。”

1960s - Software is not Hardware (2)

- 软件比硬件容易修改的多，并且不需要昂贵的生产线复制产品
  - 因为不需要昂贵的生产线，所以人力资源是软件开发的最大资源，人力成本是软件开发的主要成本，人的因素是软件开发中的最大因素。

1960s - Software is not Hardware (3)

- 软件没有损耗
  - 软件维护的主要工作是修改软件，主要成本是修改的人力成本。为了降低维护成本，要求开发者在开发时就要让软件产品设计的易于修改。

1960s - Software is not Hardware (4)

- 软件不可见
  - 很难辨别软件进度是否正常，需要开发者更多地使用模型手段以可视图形的方式反映软件生产，也需要开发者使用更深入的手段（例如度量）监控软件生产过程。

1960s - Software Crisis

- 软件危机
  - （1）对软件开发成本和进度的估计常常不准确。开发成本超出预算，实际进度比预定计划一再拖延的现象并不罕见。
  - （2）用户对“已完成”系统不满意的现象经常发生。
  - （3）软件产品的质量不可靠。
  - （4）软件的可维护程度非常之低。
  - （5）软件通常没有适当的文档资料。
  - （6）软件的成本不断提高。
  - （7）软件开发生产率无法满足人们对软件的生产要求，软件开发生产率的提高落后于硬件的发展。
- 这一情况迫使NATO科学委员会在1968和1969年召开两届里程碑式的“软件工程”会议，很多业界领先的研究者和实践者参加了这两届会议。1968年的会议主要分析了软件生产中的问题，提出了“软件危机”的说法。1969年的会议着重讨论了“软件危机”的解决方法，指出了“软件工程”的方向[Peter1969]，用工程的方法生产软件。

1960s - Software development method

- 因为缺乏正确科学知识的指导，也没有多少经验原则可以遵循，因此1960s的软件开发在总体上依靠程序员的个人能力，是“工艺式”的开发。
- 到了1960s后期，因为认识到“工艺式”开发的问题，很多研究者开始从编程入手探索解决软件危机的办法，这些促成了1970s结构化编程方法的建立。

1960s - Software Development Process

- Build and Fix
- Software Crafting
- Hero and Cowboy

1960s - Problems

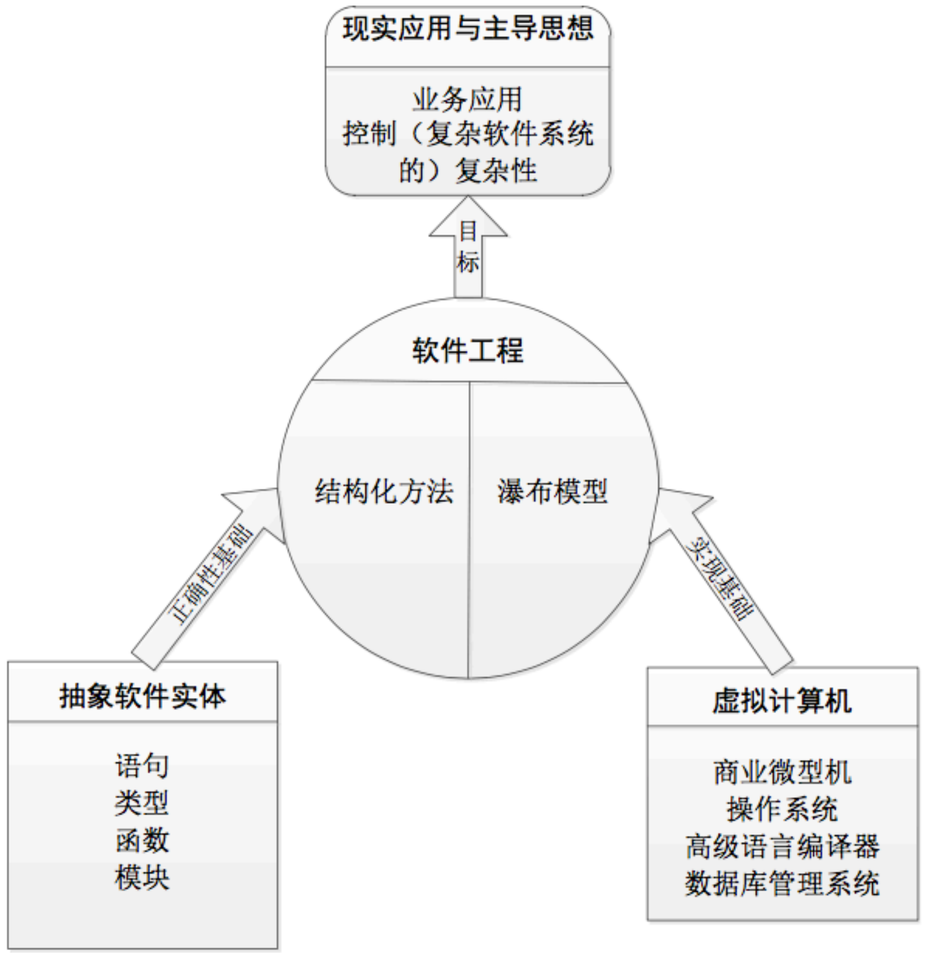
- Spaghetti code
- Many defects
- Larger project, weak plan & control

1960s - Summary

- Business Application(Batch Process)
- Application-Centric business mainframe
- 3GL
- Software is not hardware
- Software Crafting
- Computer Science Departments
- Software Crisis
- SW demand exceeded supply of SW engineer
- 图灵奖（1966）

1970 - Structured Method，WaterFall Model and Formal Method

- Software Product
- Commercial Microcomputer
- Structured Programming Language
- Structured Programming，WaterFall Model and Formal Method



1970s - Progress

- Virtual machine
  - DBMS
  - Compiler
  - Development Environment
  - 软盘，软盘驱动器
- Abstract software entity
  - C was developed between 1969 and 1973 as a system programming language, and remains popular
  - Pascal（1970），Smalltalk（1972），Prolog，Scheme（1975），SQL（1978）
  - function, block structures, sequence, branch, loop

1970s - Software development method

- Structured Programming Language
  - function
  - sequence, branch, loop
- Stepwise Refinement, Top-Down
- Decomposition and Hierarchy
- DFD, ERD, Structure Chart
- Modularity, Information Hiding
- Abstract Data Type
- Structured Method

Structured Method

- Structured programming (Bohm-Jacopini: GO TO unnecessary)
    - Formal programming calculus: Dijkstra, Hoare, Floyd
    - Formalized Top-Down SP: Mills, Baker
- Organized programs with
    - Function
        - Algorithm
    - Three control logic
    - Type and type check
- When compiler deal with functions well, organized programs with
    - Modularity
- This things gets importance with modularity
    - Information hiding
    - Data abstraction

1970s - Formal methods

- “formal methods” branch that focused on program correctness, either by mathematical proof, or by construction via a “programming calculus”

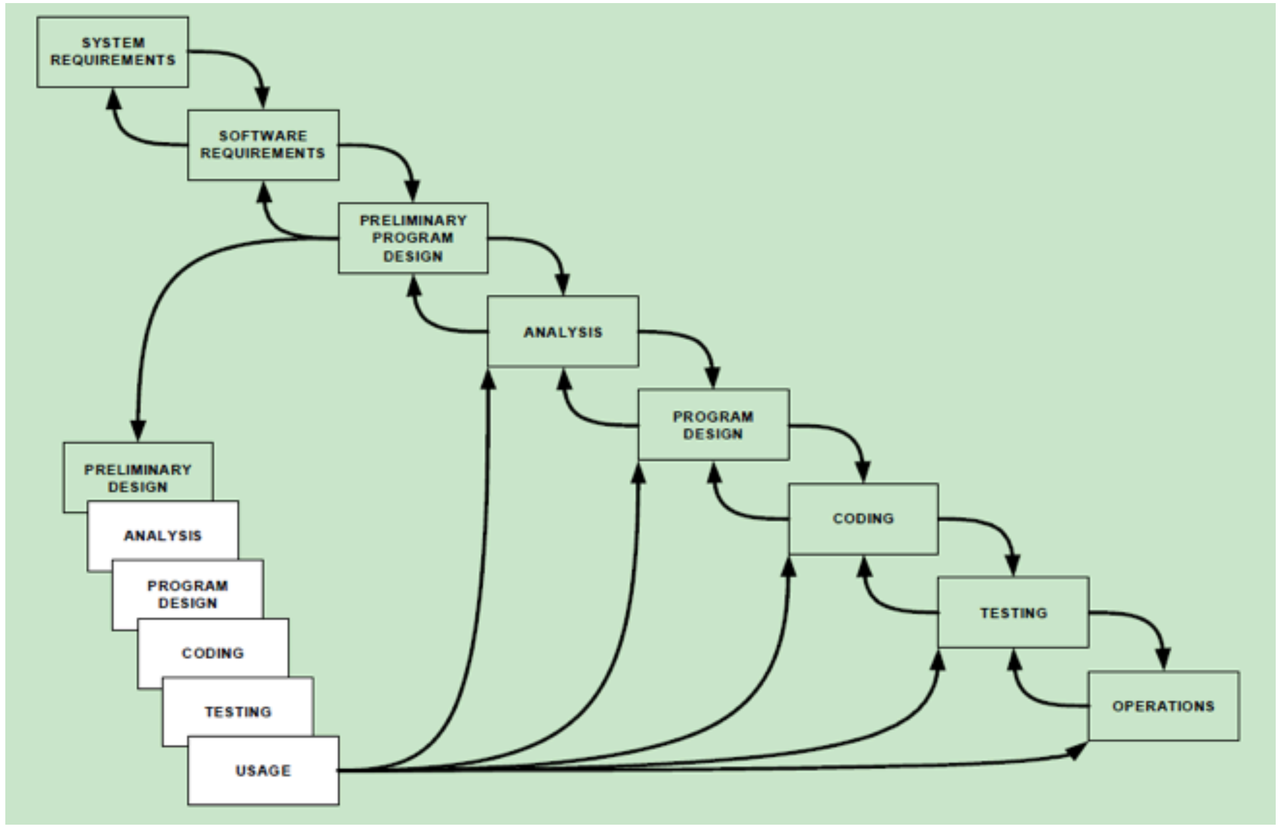
- Over the decade of the '60s theoretical computer science achieved standing as a discipline recognized by both the mathematical and the computing communities, and it could point to both applications and mathematical elegance

1970s - Problems with Formal Method

- Successful for small, critical programs
- Largest proven programs around 10 KSLOC
- Proofs show presence of defects, not absence
    - Defects in specification, proofs happen
- Scalability of programmer community
    - Techniques require math expertise, $500/SLOC
    - Average coder in 1975 survey:
        - 2 years of college, SW experience
        - Familiar with 2 languages, applications
        - Sloppy, inflexible, in over his head, and undermanaged

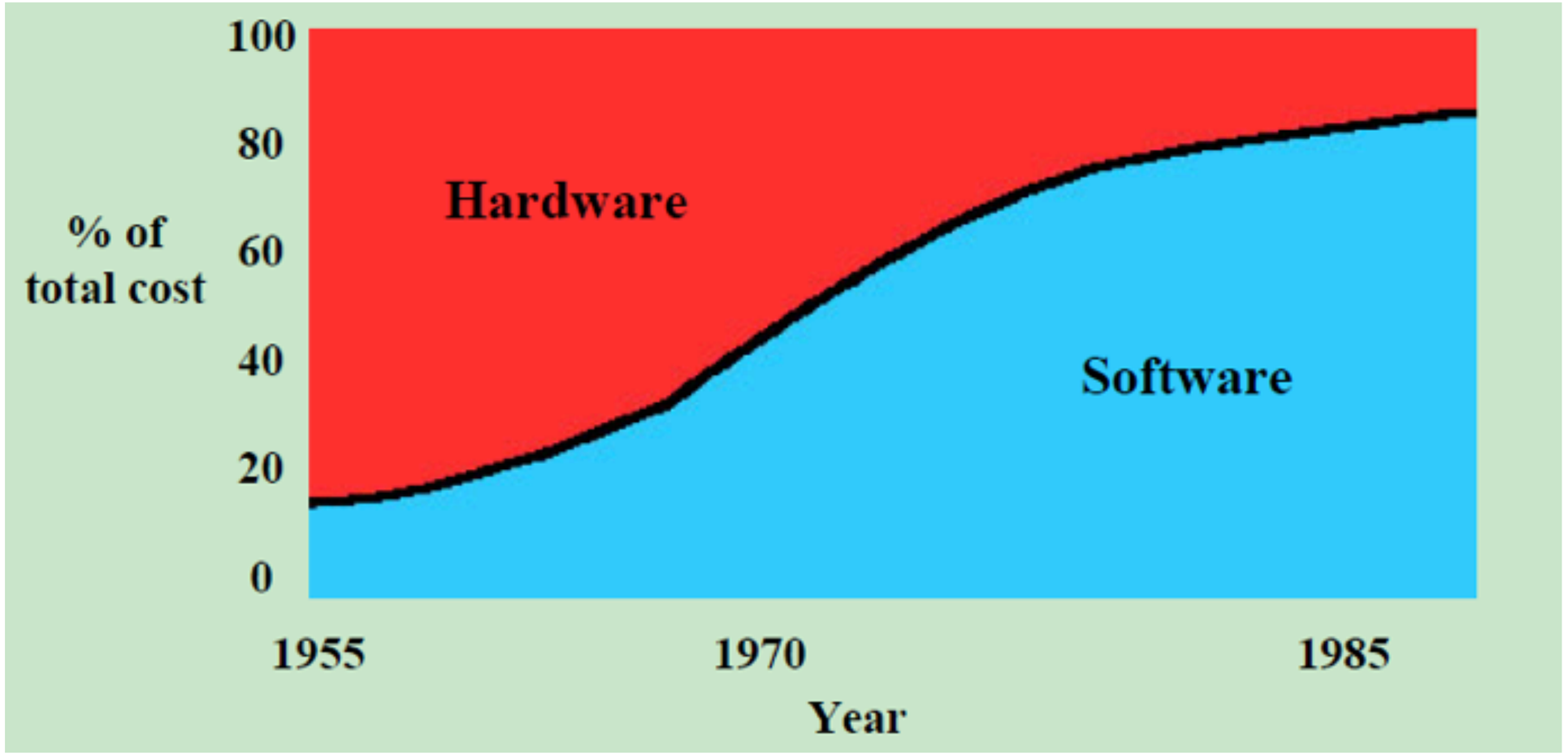
1970s - Software Development Process

- Water fall model
- Do it twice

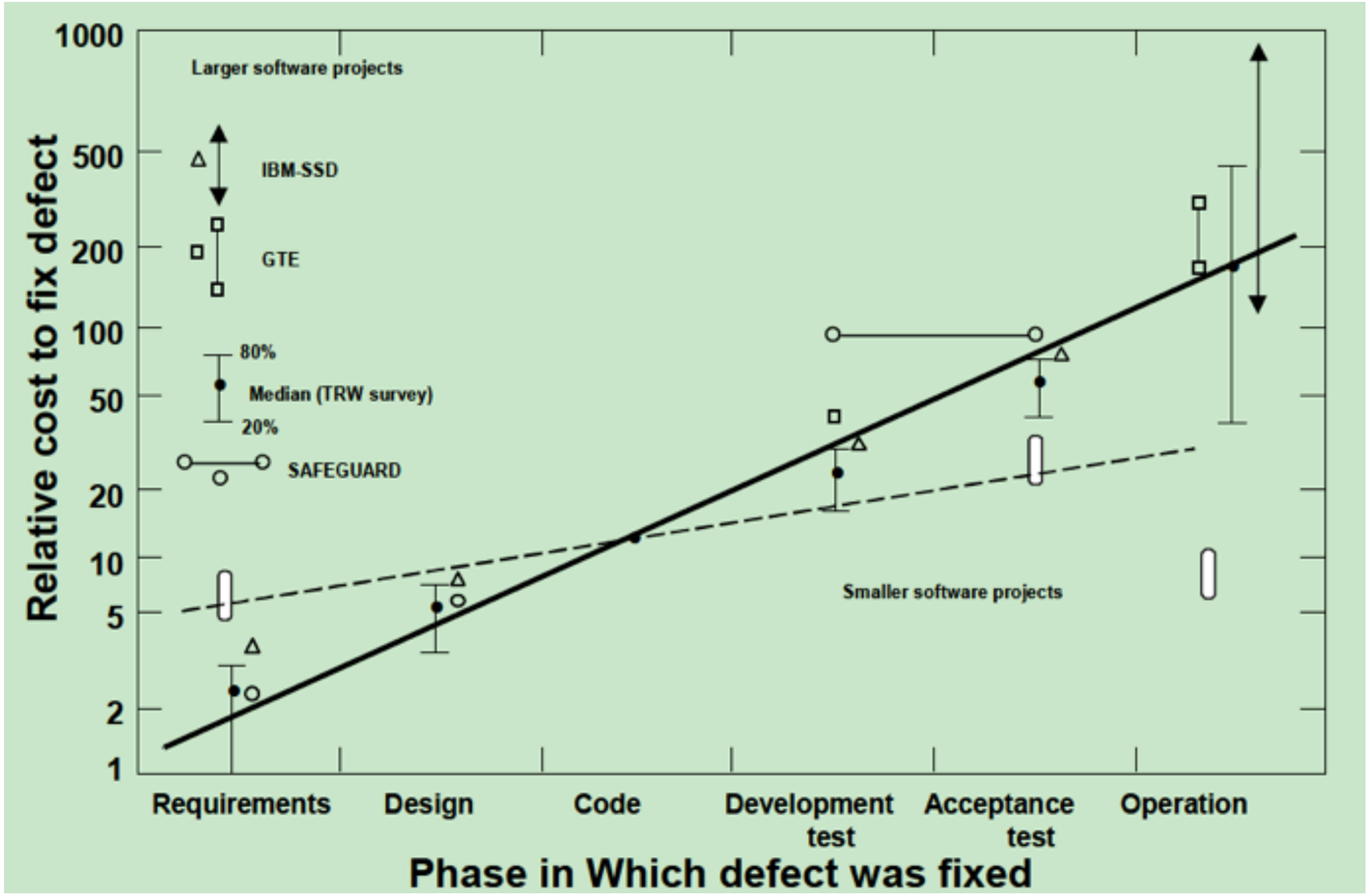


1970s - Problems with Waterfall Model

- Overly literal interpretation of sequential milestones
  - Prototyping is coding before Critical Design Review
  - Mismatch to user-intensive systems
- Heavyweight documentation hard to review, maintain
  - 7 year project with 318% requirements change
  - Milestones passed but not validated
- Mismatch to COTS, reuse, legacy SW
  - Bottom-up vs. top-down processes
- Scalability, cycle time, and obsolescence
  - Months = $5*\sqrt[3]{KSLOC}$ for sequential development
  - 3000 KSLOC => 5*14.4 = 72 months : 4 computer generations



大型机构的软硬件成本趋势图（1973），源自[Boehm1973]



Increase in Software Cost-to-fix vs. Phase (1976)

Quantitative Method

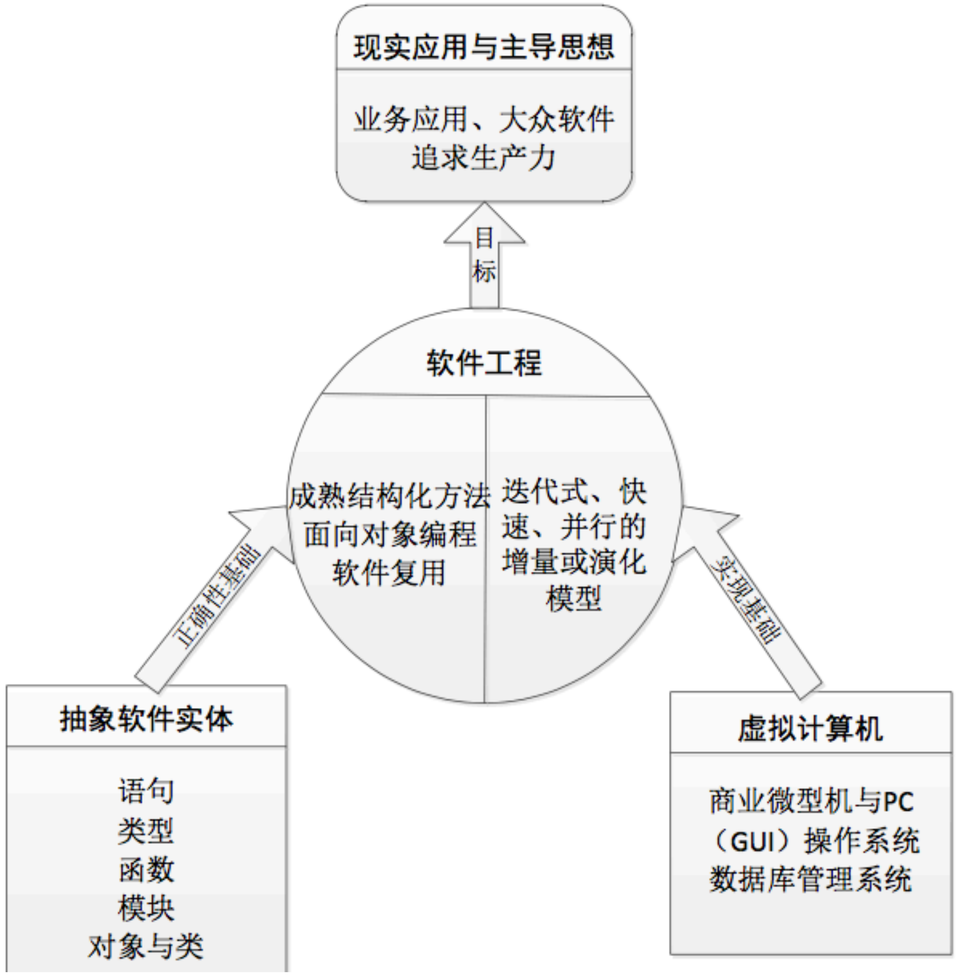
- Software productivity data
  - 26:1 productivity difference between programmers
- Software defects by phase and type
- Complexity metrics
- Software quality
- Cost and schedule estimation
- .....

1970s - Summary

- Software product
- Commercial microcomputer
- Structured programming language
- Structured method
- Waterfall model
- Formal method
- Quantitative method

1980s - Productivity, OO, Reuse, Software Process Model

- Application for individual consumer market
- OO
- Person computer & GUI



1980s - Progress

- Virtual machine
  - Personal computer
  - Better compiler
  - GUI programming environment
  - CD-ROM
- Abstract software entity
  - GUI
  - OO programming language

1980s - Application

- Exponential growth in
  - complexity of software
  - size of individual consumer market
- Worldwide concern with productivity, competitiveness
  - Japanese example: autos, electronics, Toshiba SW reuse

1980s - Software Development Method

- Modern Structured Analysis and Design
- OO Analysis and Design
- Software Reuse

1980s - Modern Structured Analysis and Design

- 1970s中后期基于结构化编程建立了早期的结构化方法，包括结构化分析与结构化设计。
  - 更多地还在关注于软件程序的构建。
- 到了1980s，人们逐步开始将结构化分析与设计的关注点转向问题解决和系统构建，产生了现代结构化方法
  - 信息工程（Information Engineering）[Martin1981]、JSD（Jackson System Development）[Jackson1983]、SSADM（Structured systems analysis and design method ）、SADT（ Structured Analysis and Design Technique）[Marca1987]和现代结构化分析（Modern Structured Analysis）[Yourdon1989]。
  - 更注重系统构建而不是程序构建，所以更重视问题分析、需求规格和系统总体结构组织而不是让分析与设计结果符合结构化程序设计理论，更重视阶段递进的系统化开发过程，而不是一切围绕最后的编程进行。

1980s - OO Programming

- 最早的面向对象编程思想可追溯到1960s的Simular-67语言[Nygaard1978]
  - 使用了类、对象、协作、继承、多态（子类型）等最基础的面向对象概念。
- 1970s的Smalltalk[Kay1993]就是完全基于面向对象思想的程序设计语言
  - 它强化了一切皆是对象和对象封装的思想，发展了继承和多态。
- 到了1980s中后期，随着C++的出现和广泛应用，面向对象编程成为程序设计的主流。C++只是在C语言中加入面向对象的特征，并不是纯粹的面向对象语言。
- 面向语言特性
  - 面向对象本身不像结构化一样有基于数学的程序设计理论的支撑
  - [Booch1997]认为模块化、信息隐藏等设计思想和数据库模型的进步都是促使面向对象概念演进的重要因素。
  - 与结构化方法相比，面向对象方法中的结构和关系（类、对象、方法、继承）能够为领域应用提供更加自然的支持，使得软件的复用性和可修改性更加强大。
  - 可复用性满足了1980s追求生产力的要求，尤其是提高了GUI编程的生产力，这也是推动面向对象编程发展的重要动力[Graham2001]。

1980s - Structured Programming vs OO Programming

- 结构化
  - 学术化
  - 严谨，充足的数学支持
- 面向对象
  - 商业化
  - 原则的必要性，流派的复杂化
    - 相当数量的程序员使用面向对象的语言写着过程式的程序
    - 面向对象是个好东西，但是只有专家级程序员才能用好它
- 工业界与学术界的鸿沟

Reuse and Object Orientation

- 1950’s: Math routines, utilities
- 1960’s: McIlroy component marketplace, Simula – 67
- 1970’s: Abstract data types, Parnas program families
- 1980’s: Smalltalk, Eiffel, C++, OO methods, reuse libraries
- 1990’s: Domain engineering, product lines, UML, pub-sub architectures
- 2000’s: Model driven development, service oriented architectures

1980s - Software Reuse

- OO
- 第4代语言
- 购买商用组件
- 程序产生器（自动化编程）
- 。。。。。。

1980s - Software Process

- 《Software Processes are Software Too》
- 过程模型
  - 原型[Budde1984]
  - 渐进交付[Basili1975]
  - 演化式开发[Gilb1981]
  - 螺旋[Boehm1988]
- 过程评价
  - CMU SEI建立了软件能力成熟度模型（SW-CMM）[Humphrey1988]，它通过评价企业的开发过程管理来反映企业的生产能力。
  - 国际标准化机构也完成一个类似的软件过程评价标准ISO-9001。
- 使用工具支持过程
  - 计算机辅助软件工程（CASE）

1980s - No Silver Bullet

- Frederick P. Brooks经过分析后认为，软件开发的根本困难在于软件产品的内在特性，它们是无法回避的，也不是可以轻易解决的，没有技术能够起到银弹的作用——没有银弹。
- Brooks’ primary candidate for addressing the essential challenges
  - great designers
  - rapid prototyping
  - evolutionary development
  - work avoidance via reuse

1980s - People

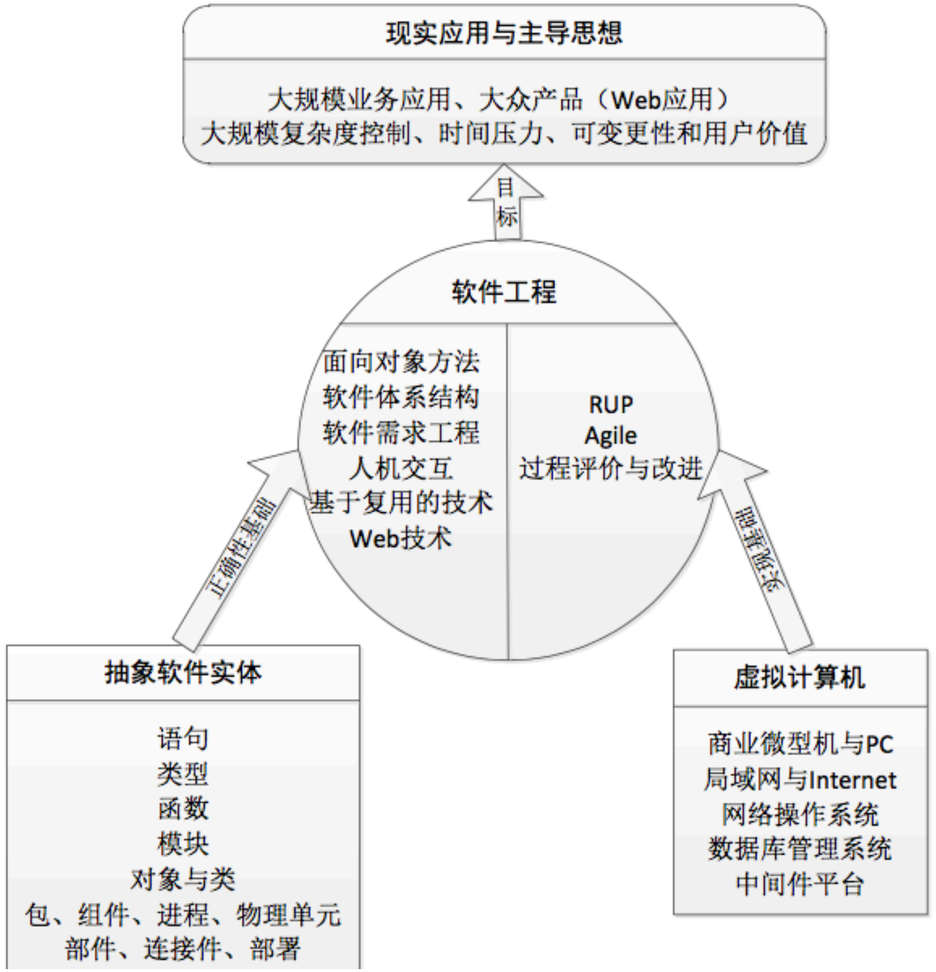
- The Most Important Factor
  - SW engineering is of the people, by the people, and for the people
- 1970’s: Weinberg Psychology of Computer Programming
- 1980’s: DeMarco-Lister Peopleware《人件》
- 1990’s – 2000’s: Importance emphasized in both Agile and CMM cultures
  - Individuals and interactions over process and tools
  - People CMM, Personal Software Process, Team Software Process

1980s - Summary

- Productivity
- Individual consumer
- Personal computer
- GUI
- Modern structured method
- OO
- Reuse
- Software process
- No silver bullet
- People

1990s - Intranet, Software Architecture, RUP,Process Improvement

- Web Application
- Intranet
- Software Architecture
- RUP



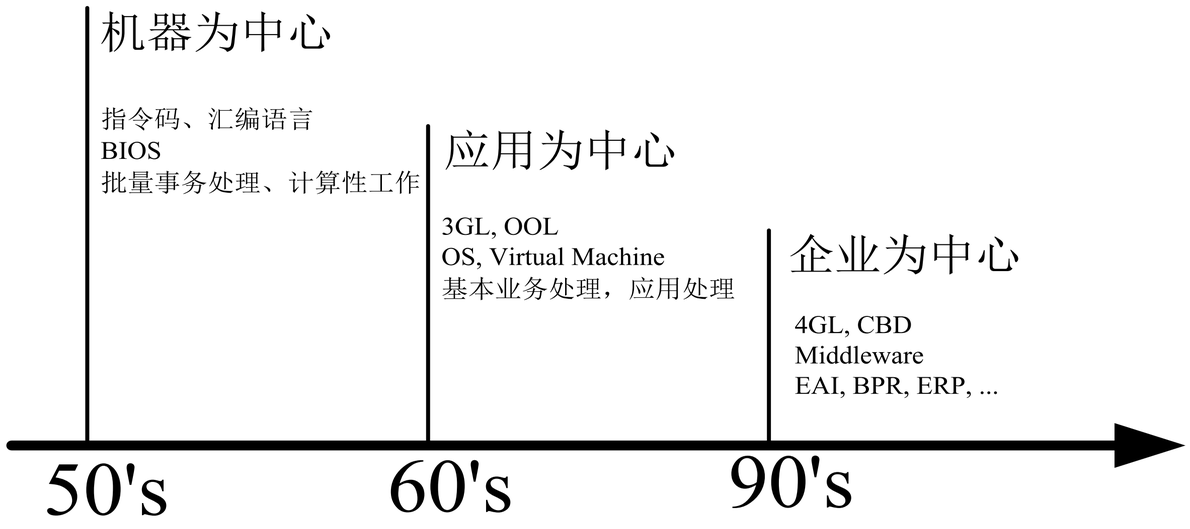
1990s - Progress

- Virtual Machine
  - Internet
  - Middleware(CORBA、DCOM、RMI)
- Abstract Software Entity
  - Software architecture
  - Java
  - ISBSG国际软件基准组织

1990s - Enterprise-Centric

- 整合以前建立的各个“应用孤岛”，实现对整个企业业务的全面整体化管理

- 以局域网为基础的软件系统可以覆盖企业的所有重要部门、重要人员和重要工作，实现它们之间的有效协同，从根本上提高企业的业务能力，价值和复杂度上都比孤立应用有着质的提高。



1990s - Large-Scale Software System

- 复杂度
  - 1990s的一个发展主题就是建立能够开发大规模软件系统的方法与技术。
- 可修改性
  - 1990s出现了对软件可变更性的持续追求，这种态势延续至今。
- 开发周期
  - 人们必须解决大规模软件系统开发周期过长的问题，并在1990s中后期提出了市场驱动开发（Market Driven Development）的口号
- 用户价值
  - 从1990s后期开始，人们认识到了用户价值的重要性，并持续探索至今，人机交互、涉众分析等很多新的方法与技术都是为了提高软件产品的用户价值而提出来的。

1990s - Software Development Method

- OO
- Software Architecture
- Human-Computer Interaction
- Requirement engineering
- Reuse based large-scale software development method
- Web application development method

1990s - OO

- 出现了OMT[Rumbaugh1991]、Booch方法[Booch1997]、OOSE[Jacobson1992]、CRC卡[Beck1989, Wirfs-Brock1990]等一系列面向对象的分析与设计方法。
- 一个统一的面向对象建模语言UML[UML]被建立和传播。
- 设计模式[Gamma1995]、面向对象设计原则[Martin2002]等有效的面向对象实践经验被广泛的传播和应用。

1990s - Reuse based large-scale software development method

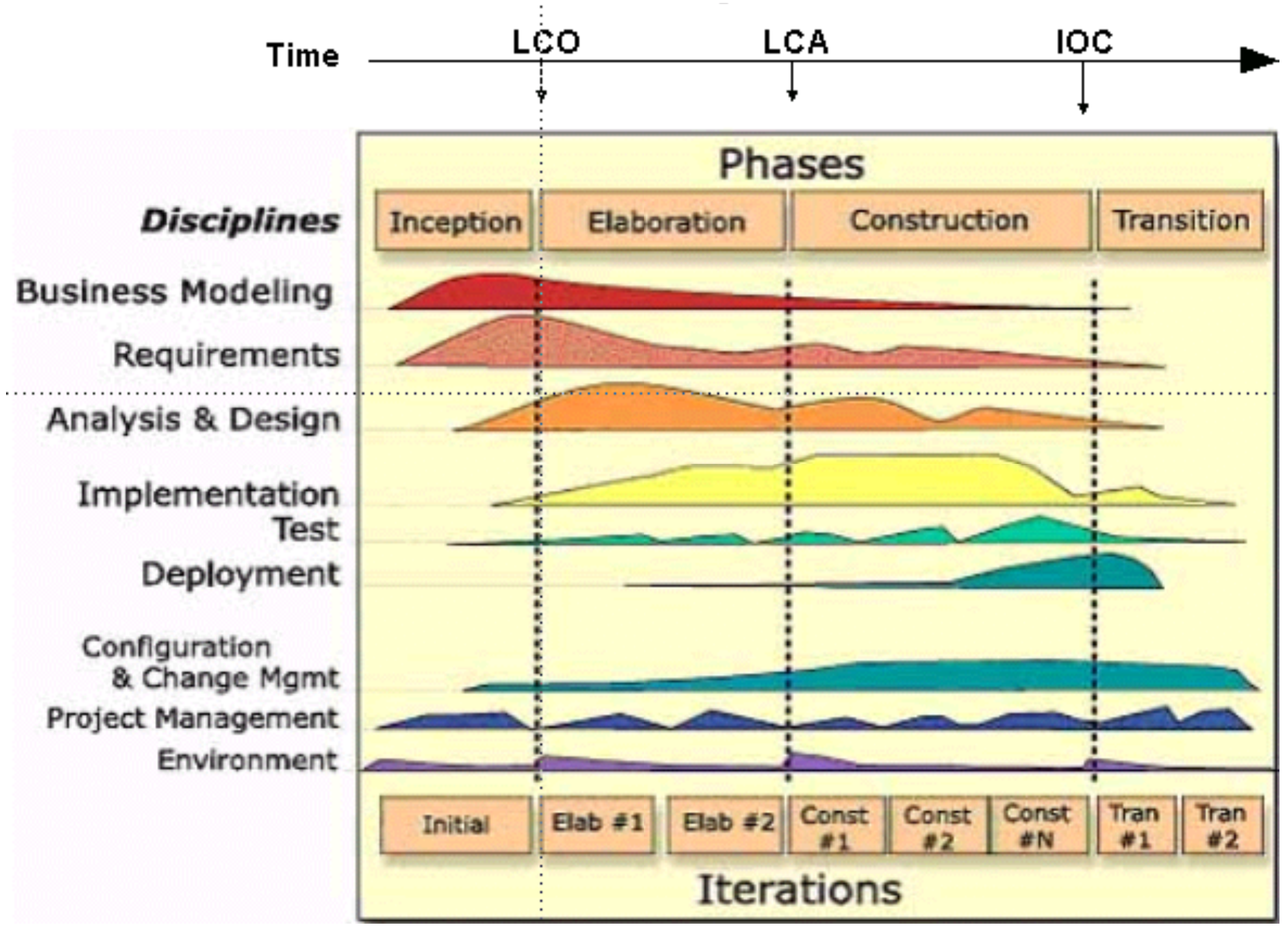
- 框架
  - 是领域特定的复用技术。它的基本思想是根据应用领域的共性和差异性特点，建立一个灵活的体系结构，并实现其中比较固定的部分，留下变化的部分等待开发者补充。
- 构件
  - 是在代码实现层次上进行复用的技术。它的基本思想是给所有的构件定义一个接口标准，就像机械工程定义螺丝和螺母的标准规格一样，这样就可以忽略每个构件内部的因素实现不同构件之间的通信和交互。COM和JavaBean就是1990s产生并流行起来的构件标准。

1990 - Web Application Development Method

- 在1990s早期，人们主要使用HTML开发静态的Web站点。
- 到了1990s中后期，ASP、JSP、PHP、JavaScript等动态Web开发技术开始流行。
- 在1990s后期，人们建立了Web程序的数据描述标准XML。

1990s - Software process

- 过程模型
  - RUP
  - Agile
  - 产品线
- 过程改进
  - CMMI
- 开源软件, 遗留软件，商用货架产品（COTS）
- Reverse engineering
- Risk- driven concurrent engineering
  - Win-Win spiral with anchor points; Rational Unified Process



1990s - Spiral Model and Concurrent Engineering

1990s - Open Source

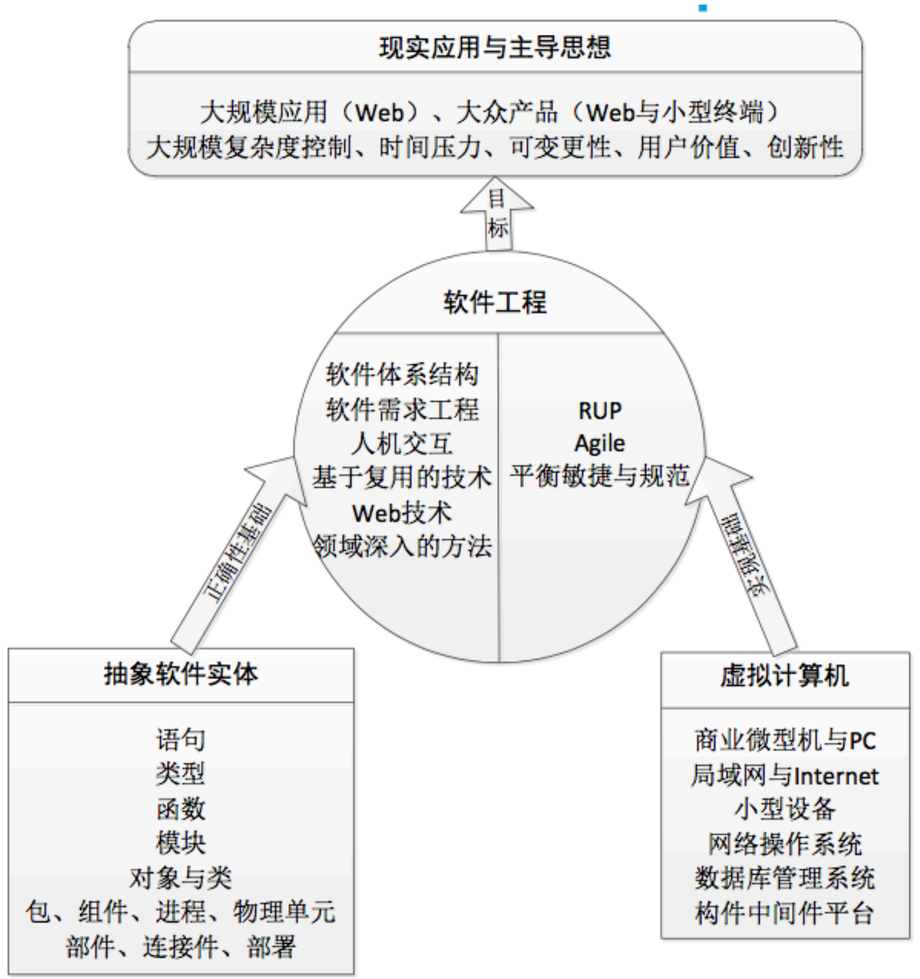
- Open source
  - From its roots in the hacker culture of the 1960’s
    - Collective code ownership
    - Free software, data, computing access
  - Free Software
  - Linux
  - Raymond’s “The Cathedral and the Bazaar”
- Components and COTS are their future

1990s - Summary

- Enterprise-Centric
- Large- Scale Software
- Web
- Software Architecture
- Domain Specific method
- product-line reuse
- Concurrent, risk-driven process
  - RUP
- CMMI

2000s - Internet, Agile, Hybrid agile and plan-driven

- Internet
- Agile



2000s - Progress

- Virtual machine
  - Internet
  - Embedded devices
  - .Net, J2EE, Web Service
  - Linux, WinCE, iOS, Android
  - Team-Development Environment
- Abstract software entity
  - UML 2.0
  - Model-driven development and service-oriented architectures
  - Hybrid agile and plan-driven product and process architectures
    - Encapsulate parts with rapid, unpredictable change
    - Concurrent build-to-spec, V&V, agile rebaselining

2000s - Consumer-Centric Application

- Web Application
- Mass Consumer

- Increasing integration of systems engineering and SW engineering
  - Increasing trend toward “soft systems engineering”
- Increasing focus on usability and value
  - Fit the software and hardware to the people, not vice versa
- Emergent vs. prespecifiable requirements

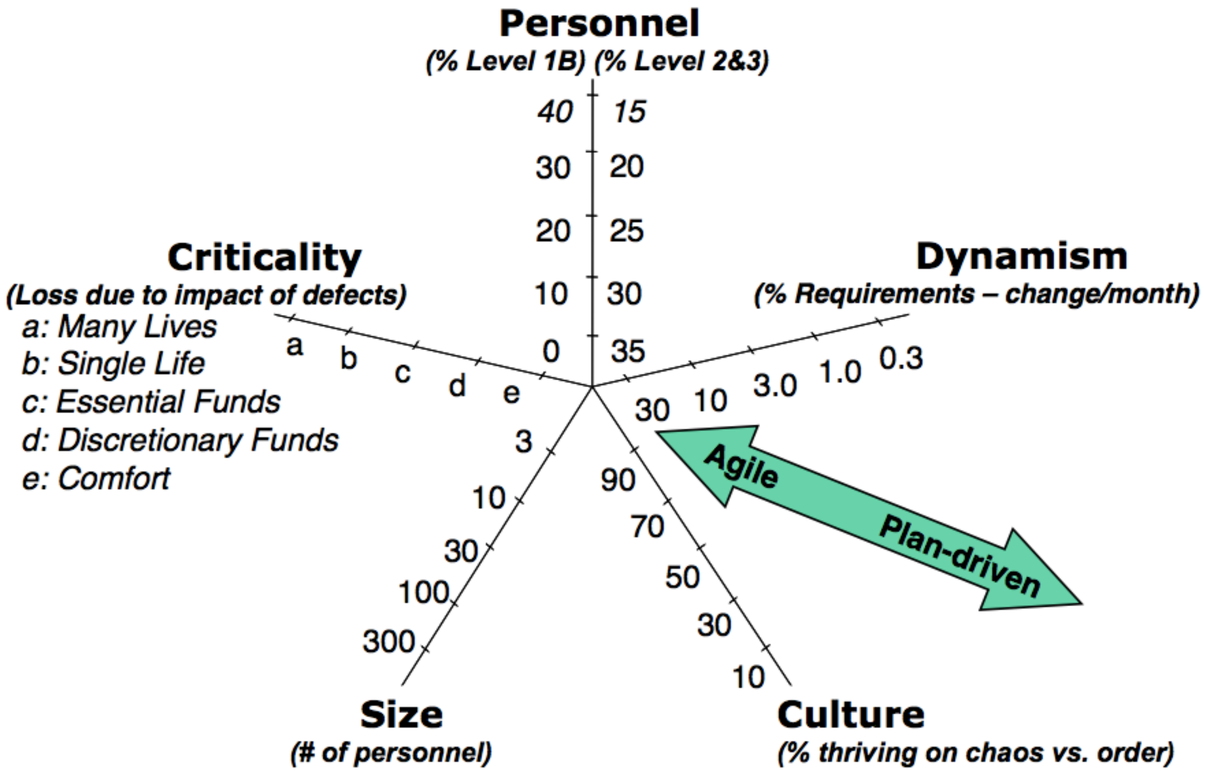
2000s - Software Development Method

- 延续1990s的技术进展
  - Integrated System and Software Engineering
  - Hybrid Agile, Plan-Driven Methods
  - Service-Oriented Architecture, Model-Driven Development
- Web技术发展
- 领域特定的软件工程方法
  - 以网络为中心的系统；
  - 信息系统；
  - 金融和电子商务系统；
  - 高可信系统；
  - 嵌入式和实时系统；
  - 多媒体、游戏和娱乐系统；
  - 小型移动平台系统。

2000s - Software Process

- Agile
- Balance

- **Size, Criticality, Dynamism, Personnel, Culture**

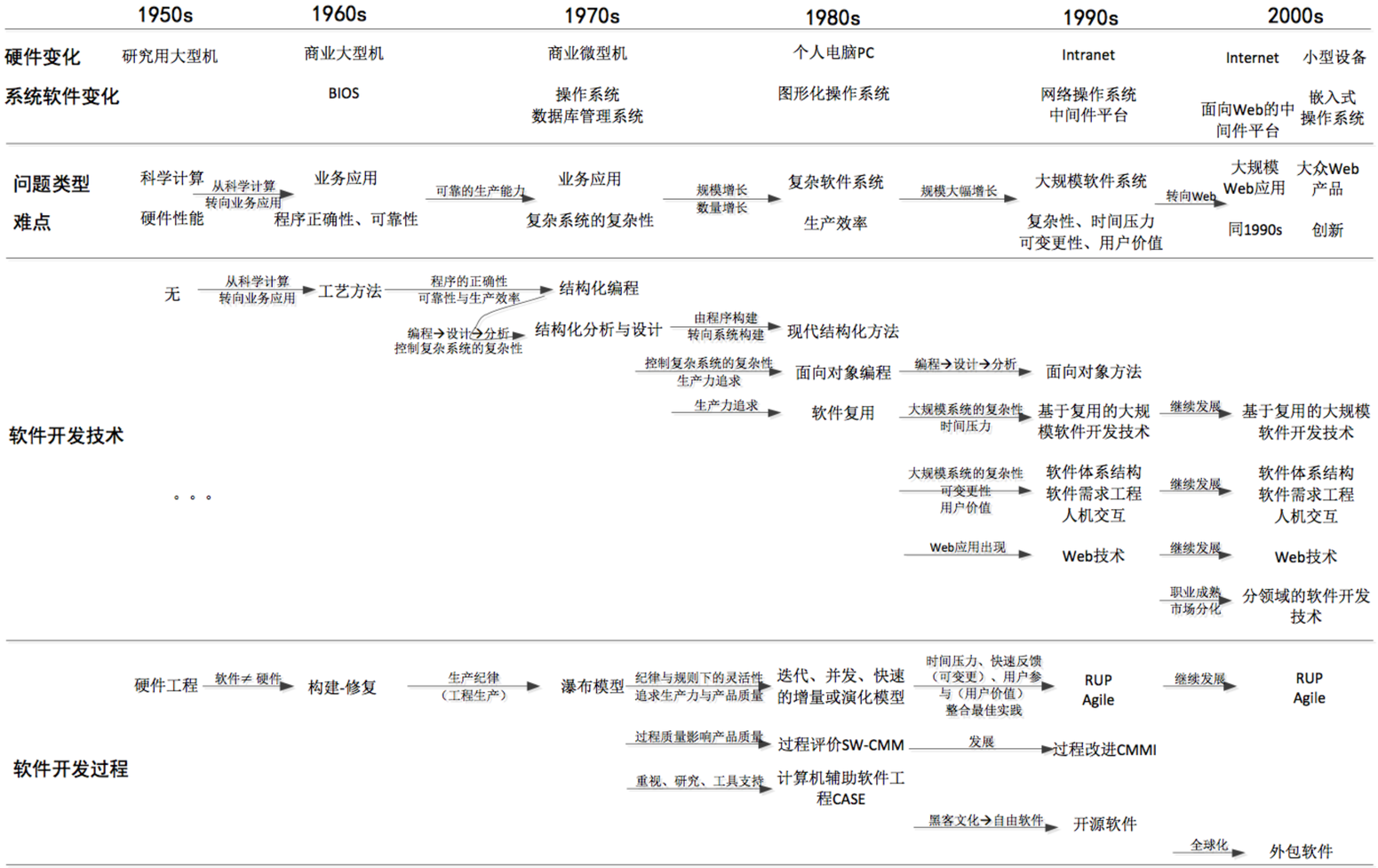


2000s - SE: From commercial to profession

- Body of knowledge
- Accredited university education

2000s - Summary

- Large-Scale Web Application
- Mass-Consumer Application
- Domain-Specific Software Engineering
- Agile


1950s
1960s
1970s
1980s
1990s
2000s
硬件变化
研究用大型机
商业大型机
商业微型机
个人电脑PC
Intranet
Internet
小型设备
系统软件变化
BIOS
操作系统
数据库管理系统
图形化操作系统
网络操作系统
中间件平台
面向Web的中间件平台
嵌入式操作系统
问题类型
科学计算
从科学计算转向业务应用
业务应用
可靠的生产能力
业务应用
规模增长
数量增长
复杂软件系统
规模大幅增长
大规模软件系统
转向Web
大规模Web应用
大众Web产品
难点
硬件性能
程序正确性、可靠性
复杂系统的复杂性
生产效率
复杂性、时间压力
可变更性、用户价值
同1990s
创新
软件开发技术
无
从科学计算转向业务应用
工艺方法
程序的正确性可靠性与生产效率
结构化编程
编程→设计→分析
控制复杂系统的复杂性
结构化分析与设计
由程序构建转向系统构建
现代结构化方法
控制复杂系统的复杂性
生产力追求
面向对象编程
编程→设计→分析
面向对象方法
生产力追求
软件复用
大规模系统的复杂性
时间压力
基于复用的大规模软件开发技术
继续发展
基于复用的大规模软件开发技术
。。。
大规模系统的复杂性
可变更性
用户价值
软件体系结构
软件需求工程
人机交互
继续发展
软件体系结构
软件需求工程
人机交互
Web应用出现
Web技术
继续发展
Web技术
职业成熟
市场分化
分领域的软件开发技术
软件开发过程
硬件工程
软件≠硬件
构建-修复
生产纪律（工程生产）
瀑布模型
纪律与规则下的灵活性
追求生产力与产品质量
迭代、并发、快速的增量或演化模型
时间压力、快速反馈（可变更）、用户参与（用户价值）整合最佳实践
RUP
Agile
继续发展
RUP
Agile
过程质量影响产品质量
过程评价SW-CMM
发展
过程改进CMMI
重视、研究、工具支持
计算机辅助软件工程CASE
黑客文化→自由软件
开源软件
全球化
外包软件

Summary

2010s - Cloud Native

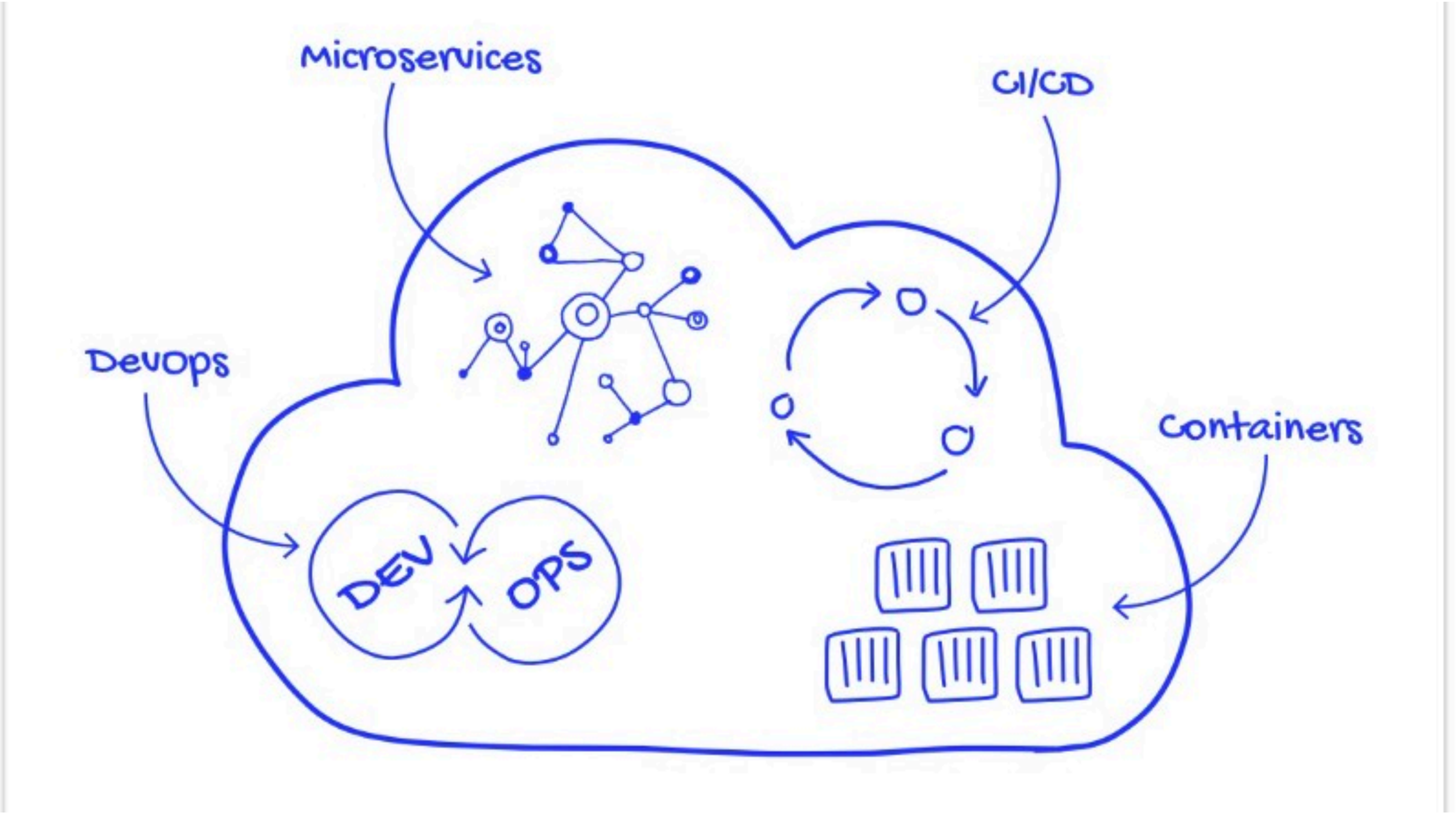
- 云原生
- 众包
- Facebook,twiter,wechat,tiktok
- Big data
- AI
- 可穿戴式设备

2010s later - Barry Bohem's view

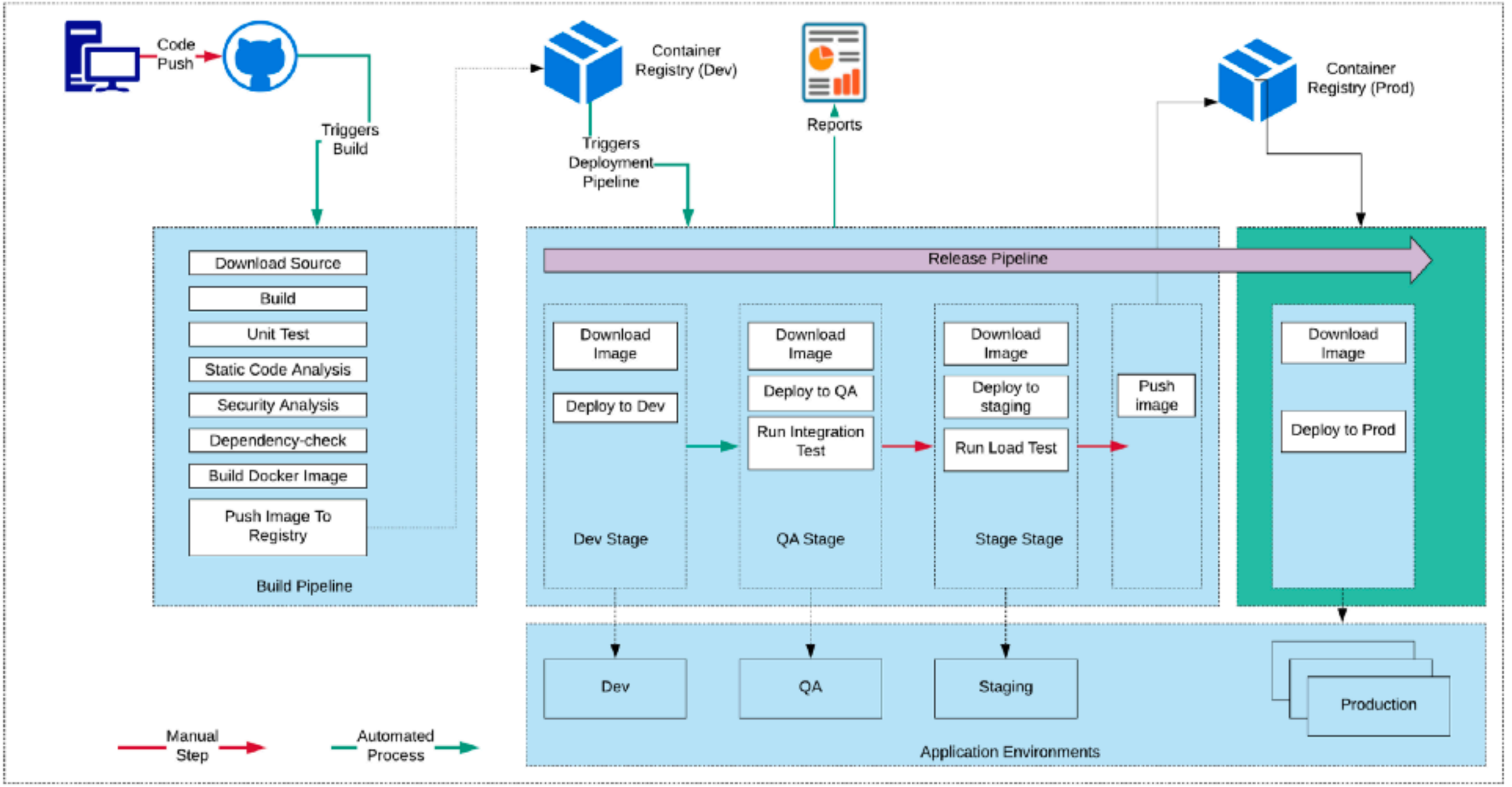
- Eight surprise-free trends
  - Increasing integration of SysE and SwE
  - User/Value focus
  - Software Criticality and Dependability
  - Rapid, Accelerating Change
  - Distribution, Mobility, Interoperability, Globalization
  - Complex Systems of Systems
  - COTS, Open Source, Reuse, Legacy Integration
  - Computational Plenty
- Two wild-card trends
  - Autonomy Software
  - Combinations of Biology and Computing

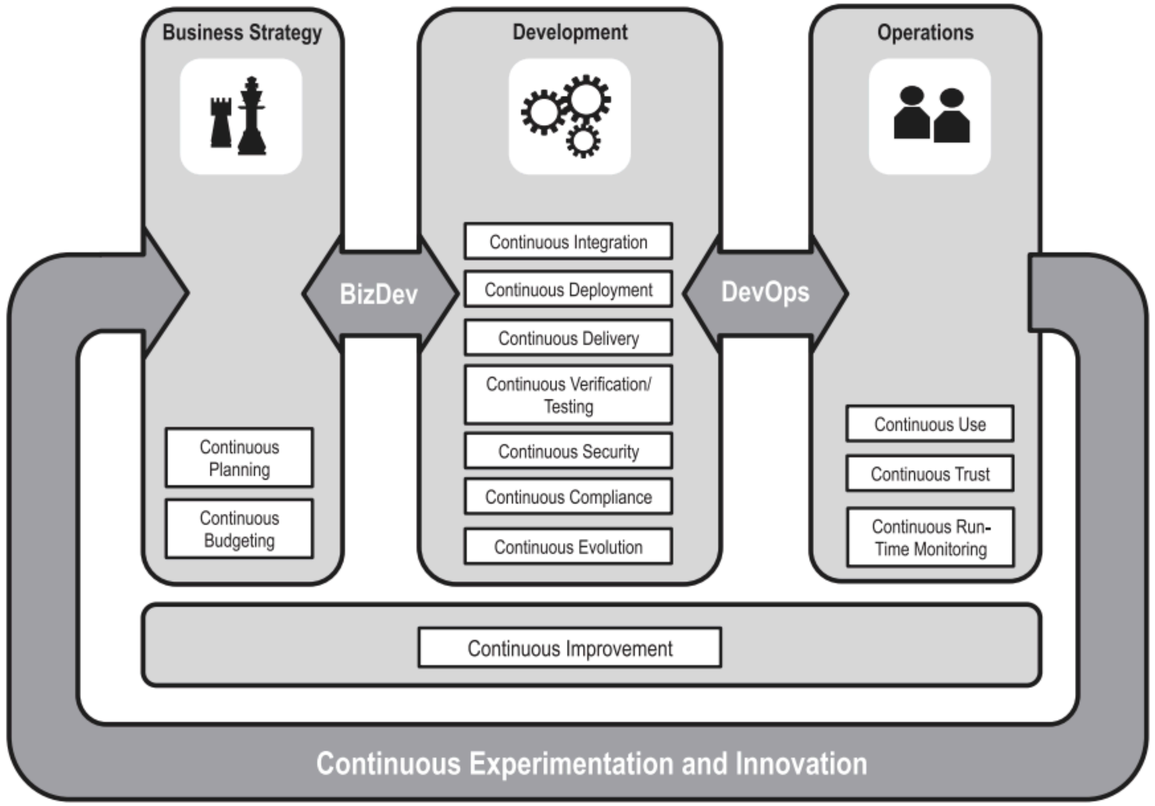
软件工程有可能的改进[Jones2019]

- 软件生产力和质量度量不精确
- 存储认证可重用组件需要成为主流
- 质量控制不稳定
- 变更控制也做的不好
- 项目估算也有待提高
- 软件安全一直在警戒水平以下
- 软件工程需要发放执照和职业资格认证的真正职业
- 非功能性需求的新型度量方法
- 遗留软件的维护



最近热门的软件工程概念


Code Push
Triggers Build
Container Registry (Dev)
Triggers Deployment Pipeline
Reports
Container Registry (Prod)
Download Source
Build
Unit Test
Static Code Analysis
Security Analysis
Dependency-check
Build Docker Image
Push Image To Registry
Build Pipeline
Release Pipeline
Download Image
Deploy to Dev
Dev Stage
Download Image
Deploy to QA
Run Integration Test
QA Stage
Download Image
Deploy to staging
Run Load Test
Stage Stage
Push image
Download Image
Deploy to Prod
Dev
QA
Staging
Production
Application Environments
Manual Step
Automated Process



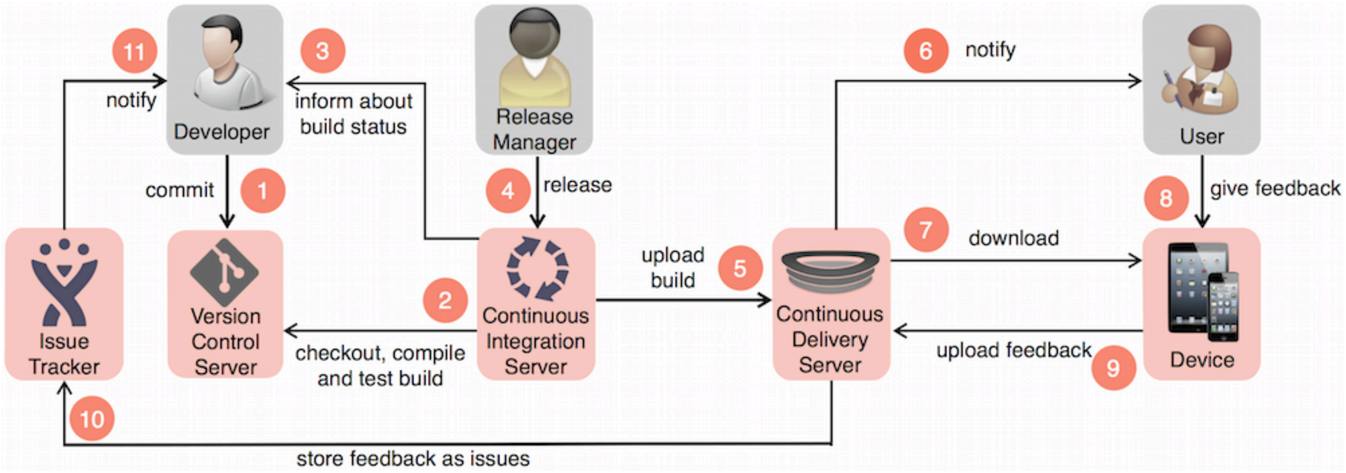
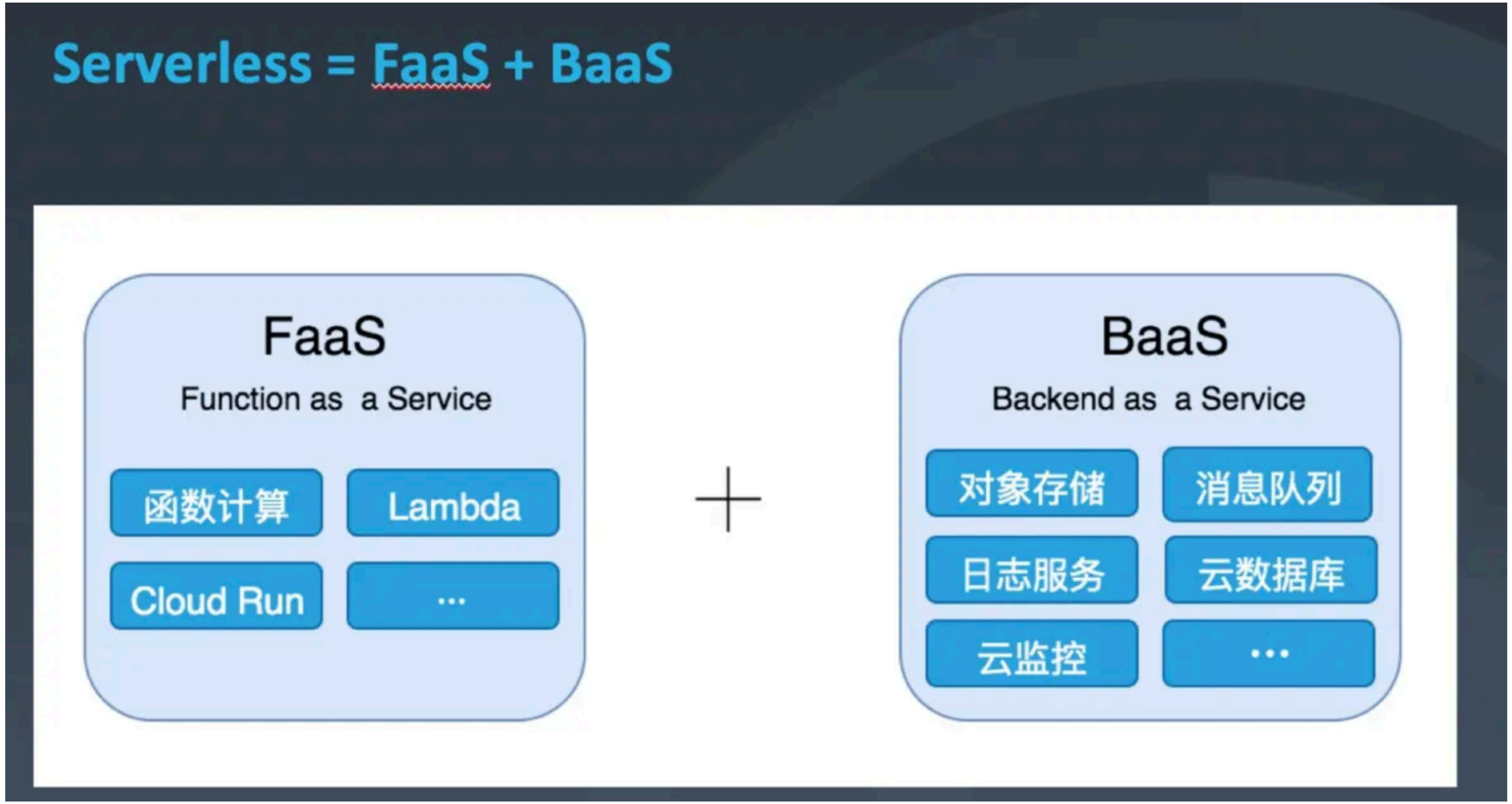


Fig. 3. Continuous*: a holistic view on activities from business, development, operations and innovation.

Continuous Software Engineering

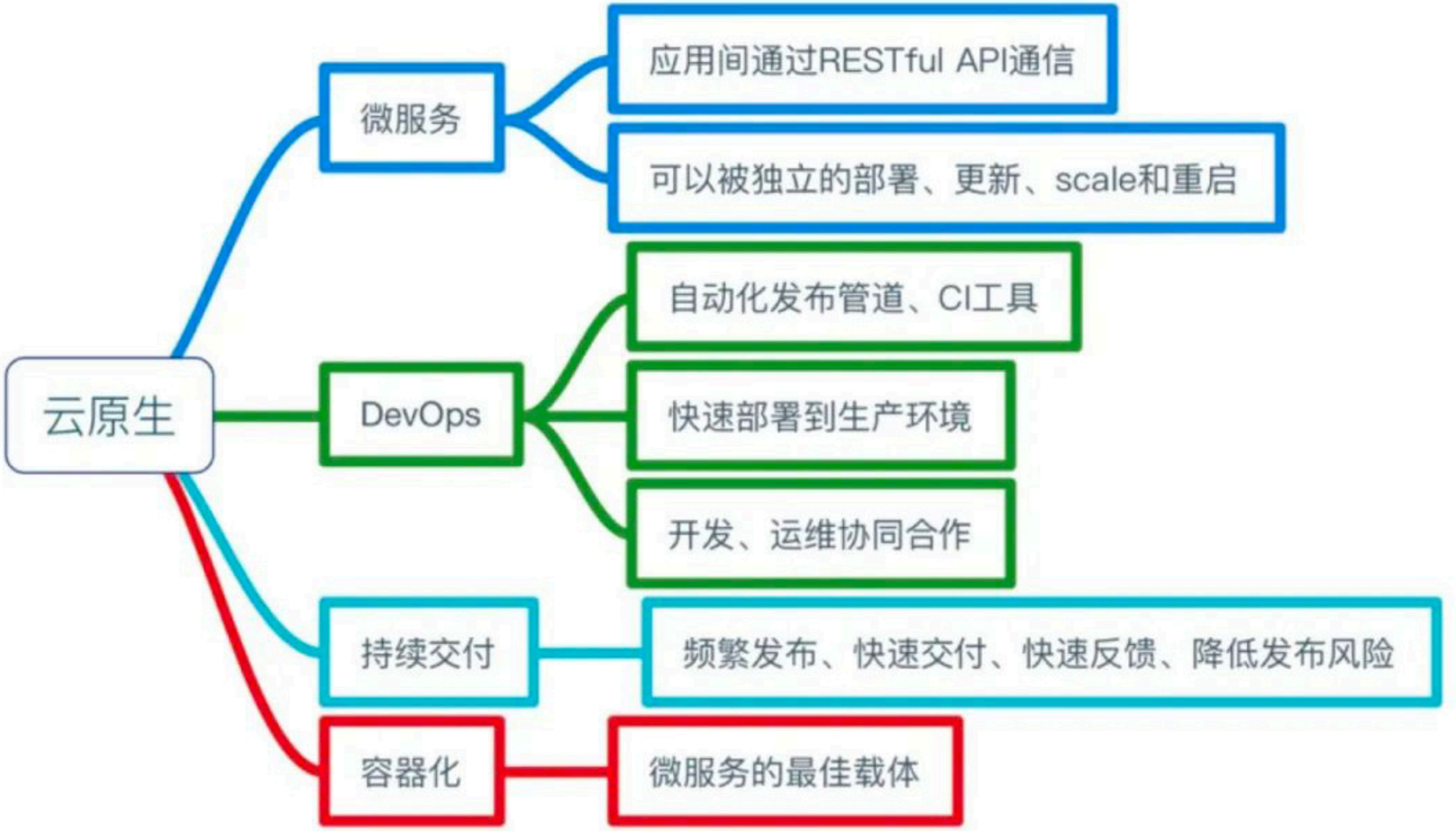

Serverless = FaaS + BaaS
FaaS
Function as a Service
函数计算
Lambda
Cloud Run
...
BaaS
Backend as a Service
对象存储
消息队列
日志服务
云数据库
云监控
...

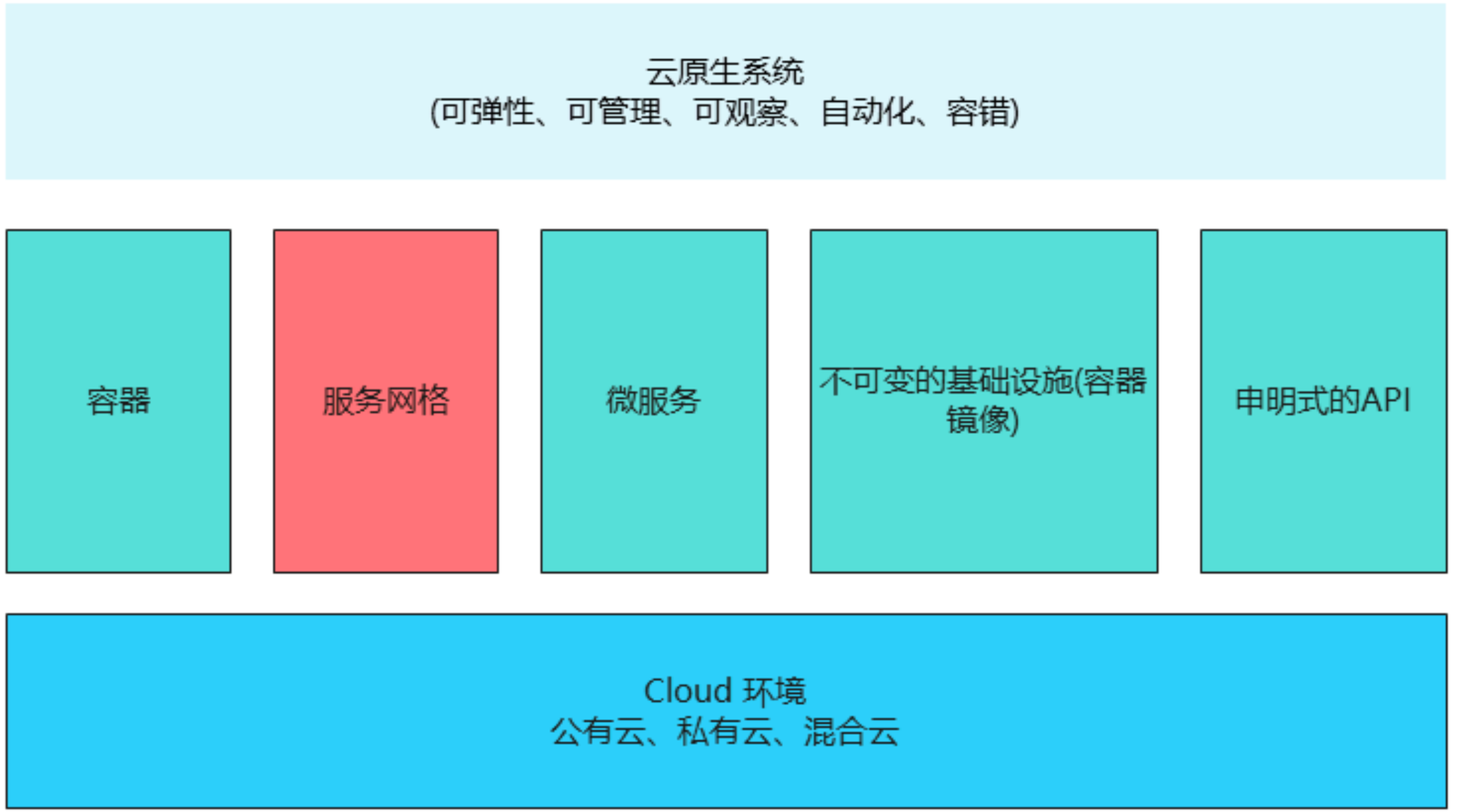
Serverless

云原生

历史

- 2013年，Pivotal公司的Matt Stine于首次提出原生云(CloudNative)的概念，用于区分为云而设计的应用和云上部属传统应用。
- 2015年，Matt Stine在《迁移到云原生架构》一书中定义了符合原生云架构的几个特征：12因素、微服务、自敏捷架构、基于API协作、抗脆弱性;
- 2015年云原生计算基金会(CNCF)成立,CNCF作为一个厂商中立的基金会，致力于云原生应用推广和普及。
- 2017年，Matt Stine将原生云架构归纳为模块化、可观察、可部署、可测试、可替换、可处理6特质;而Pivotal最新官网对云原生概括为4个要点：DevOps+持续交付+微服务+容器。


云原生
微服务
应用间通过RESTful API通信
可以被独立的部署、更新、scale和重启
DevOps
自动化发布管道、CI工具
快速部署到生产环境
开发、运维协同合作
持续交付
频繁发布、快速交付、快速反馈、降低发布风险
容器化
微服务的最佳载体


云原生系统
(可弹性、可管理、可观察、自动化、容错)
容器
服务网格
微服务
不可变的基础设施(容器镜像)
申明式的API
Cloud 环境
公有云、私有云、混合云

云原生技术

定义

- Cloud native technologies empower organizations to build and run scalable applications in modern, dynamic environments such as public, private, and hybrid clouds. Containers, service meshes, microservices, immutable infrastructure, and declarative APIs exemplify this approach.

- 云原生技术有利于各组织在公有云、私有云和混合云等新型动态环境中，构建和运行可弹性扩展的应用。云原生的代表技术包括容器、服务网格、微服务、不可变基础设施和声明式 API。

- These techniques enable loosely coupled systems that are resilient, manageable, and observable. Combined with robust automation, they allow engineers to make high-impact changes frequently and predictably with minimal toil.

- 这些技术能够构建容错性好、易于管理和便于观察的松耦合系统。结合可靠的自动化手段，云原生技术使工程师能够轻松地对系统作出频繁和可预测的重大变更。

- The Cloud Native Computing Foundation seeks to drive adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects. We democratize state-of-the-art patterns to make these innovations accessible for everyone.

- 云原生计算基金会（CNCF）致力于培育和维护一个厂商中立的开源生态系统，来推广云原生技术。我们通过将最前沿的模式民主化，让这些创新为大众所用。

Summary - Timeless principles(+) and aging practices(-)

- From the 1950’s
  - + Don’t neglect the sciences
  - + Look before you leap (avoid premature commitments)
  - - Avoid inflexible sequential processes
- From the 1960’s
  - + Think outside the box
  - + Respect software’s differences
  - - Avoid cowboy programming

- From the 1970’s
  - + Eliminate errors early
  - + Determine the system’s purpose
  - - Avoid top-down development and reductionism

- From the 1980’s
  - + These are many roads to increased productivity
  - + What’s good for products is good for processes
  - - Be skeptical about silver bullets

- From the 1990’s
  - + Time is money and value to people
  - + Make software useful to people
  - - Be quick, but don’t hurry
- From the 2000’s
  - + If change is rapid, adaptability trumps repeatability
  - + Consider and satisfy all of your stakeholders’ value propositions
  - - Avoid falling in love with your slogans(YAGNI)

- For the 2010’s
  - + Keep your reach within your grasp
  - + Have an exist strategy
  - - Don’t believe everything you read
  - - “It’s true because I read it on the Internet”

Future Challenges for SW Engineering Education - Student careers go through 2050's

- Keeping courseware continually up-to-date
- Anticipating future trends and preparing students for them
- Separating timeless principles from aging practices
- Making small student projects relevant to large industry practices
- Participating in research; incorporating results in courses
- Helping students learn how to learn
- Offering lifelong learning to practitioners

Q&A