Class 0

Powered by: Hanyuu

1945-1968 The origins -> coding -> software development

Code and fix;

1968/69-2001 Software development -> software engineering(NATO,1968/69)

Software Process,Life cycle,系统化工程化

Black-box process

2001-now 敏捷联盟成立->Agile Development[no silver bullet]

Iteration,Increment,Continuous,Intergration

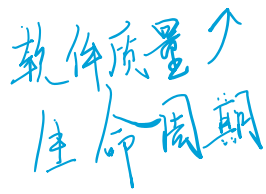
Whole life-cycle. Multiple life-cycles

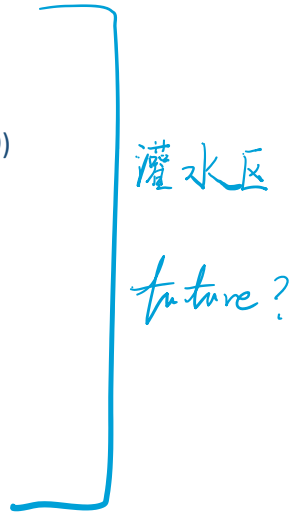
White-box preocess

Way to intelligent SE

Automation ,Intelligence,smart

单/多生命周期





考核:

期末考 80%p

平时 （随堂测试+作业）+课程作业 10%+10%

???

Unit0_0overview


Southeast University
软件工程导论
李必信
bx.li@seu.edu.cn

1968-2018:软件工程发展50年

- **1945 to 1968**: **The origins （二进制）→ Coding（汇编语言）→ software development(1951)（Fortran）**
  - Code and Fix
- **1968/1969 to 2001**: **Software development → software engineering（NATO，1968\1969）**
  - Software Process 、Life-cycle、系统化/工程化
  - Black-box process
- **2001 to present**：敏捷联盟成立→ Agile Development |No silver bullet|…
  - Iteration、Increment、Continuous Integration、
  - Whole life-cycle、Multiple life-cycles
  - White-box process
- **We are on the way to intelligent SE**|人工智能|大数据|…
  - Automation、Intelligence、Smart…

Outline

1. 课程基本情况
2. 课程的目的和地位
3. 教学内容
4. 培养目标

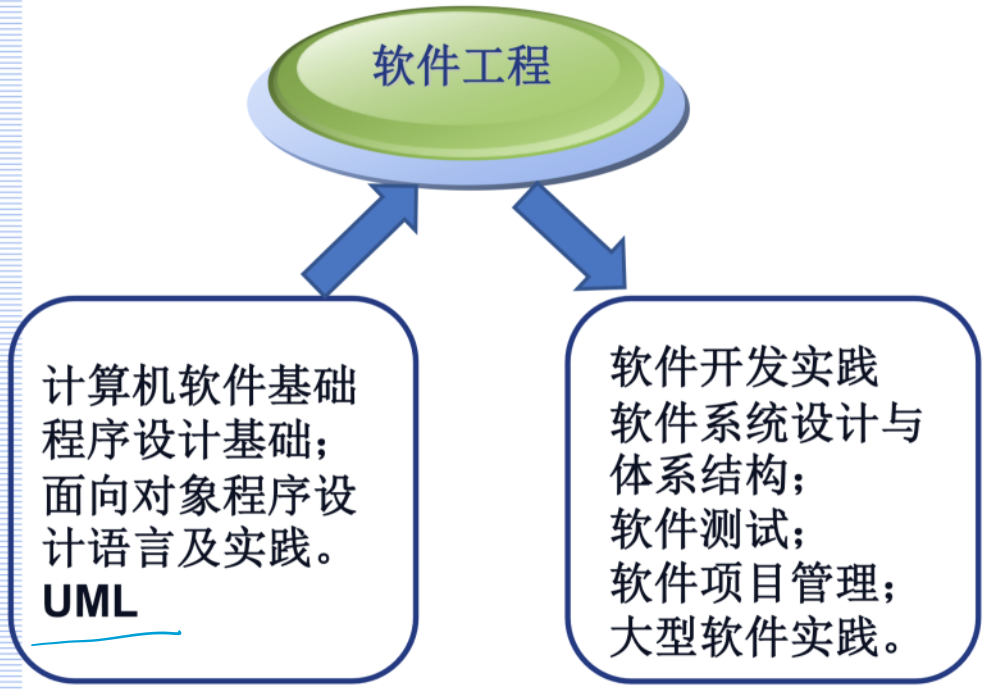
1. 课程基本情况

- 学时数：32
- 学分数：2.5
- 教材： Software Engineering: A Practitioner's Approach (*sixth edition*), SEPA, 6/e. 作者： *R. S. Pressman*
- 考核方式
  - 期末考试 （80%）
  - 平时[(随堂测试+作业)+课程作业][10%+10%]
- 课堂要求：迟到…no!手机…close!

2. 课程的目的和地位(1)

- 软件工程是用工程学的思想和准则（系统的、规范的、经过严格训练的思想）来研究软件开发过程中的问题及解决这些问题的技术和方法，以达到下列目标：
  - **开发出满足客户需求的、让客户满意的软件**
    - **开发出没有缺陷的软件**
    - **软件开发成本在预算范围内**
    - **按时交付软件**

2. 课程的目的和地位(2)


软件工程
计算机软件基础
程序设计基础；
面向对象程序设计语言及实践。
UML
软件开发实践
软件系统设计与体系结构；
软件测试；
软件项目管理；
大型软件实践。

2. 课程的目的和地位(3)

- 本课程学习目标
  - 掌握软件工程涉及的基本概念
  - 了解软件生产过程中可能出现的问题
  - 掌握软件生产过程中解决有关问题的方法与技术
  - 了解软件项目管理
  - 通过实践实际地运用软件工程的方法和技术，基本掌握软件项目[redacted]和团队开发的工作方法

2. 课程的目的和地位(4)

- 软件工程的有关原理和准则、涉及的方法和技术对于计算机应用软件的开发与维护具有直接的指导作用，对于软件方法学的发展有着重要的指导意义。
- 在软件工程学科中处于基础和核心地位，贯穿软件工程科学学位、专业学位学生培养的整个过程。对将来从事软件工程科学研究、软件工程实践的人们来说，绝对是良师益友。
- 另外，本课程在计算机科学与技术学科体系中具有重要的地位。

3. 教学内容

- 软件工程的概念、方法和技术，包括：
  - **软件工程基本概念（软件产品、软件过程、软件开发模型）**
  - **软件工程开发方法和技术**
    - 传统的软件工程方法与技术
    - 面向对象的软件工程方法与技术
  - **软件测试策略和技术**
  - **软件项目管理**
- 授课形式：讲授、自学、讨论、实践
- 课件下载：
- http://cse.seu.edu.cn/PersonalPage/bx.li/index.htm

=

4. 培养目标

- 软件工程意识的培养
  - **程序员→ 软件工程师**
  - **编程→ 软件开发**
  - **个人开发→ 团队合作**
- 软件分析与设计能力的培养
- 软件项目管理能力的培养
- 团队合作精神的培养

Unit0_chapter_01

Chapter 1
Software and Software Engineering

Software Engineering: A Practitioner's Approach, 6th edition
by Roger S. Pressman

Software's Dual Role

- Software is a product 软件即产品（服务）
  - *Transforms* information - produces, manages, acquires(获取), modifies, displays, or transmits information;
  - Delivers computing potential of hardware and networks
- Software is a vehicle for delivering a product
  - Controls other programs (operating system);
  - Effects(实现) communications (networking software);
  - Helps build other software (software tools & environments)

Eg.CAD

Software Applications

- system software
- application software
- engineering/scientific software[ROSE, CAD etc.]
- embedded software
- product-line software
- web applications
- AI software

Hardware vs. Software

Hardware	Software
■ Manufactured	■ Developed/engineered
■ Wears out[磨损]	■ Deteriorates[恶化]
■ Built using components	■ Custom built[根据需求定制的]
■ Relatively simple	■ Complex

构建

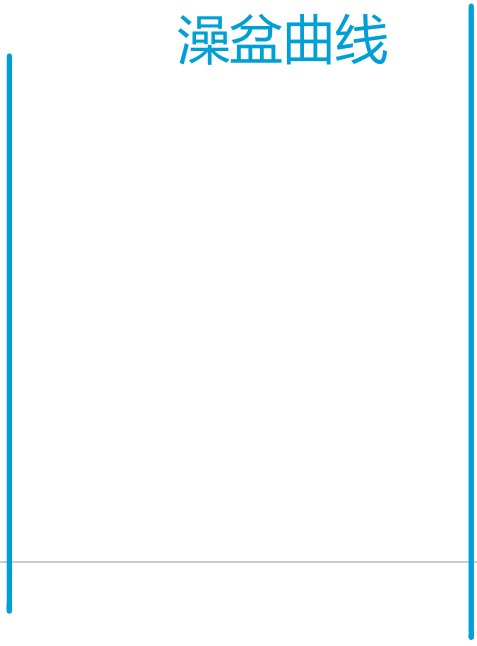
Construct

Manufacturing vs. Development

Manufacturing vs. Development

- Once a hardware product has been manufactured, it is difficult or impossible to modify. In contrast, software products are routinely modified and upgraded.
- In hardware, hiring more people allows you to accomplish more work, but the same does not necessarily hold true in software engineering.
- Unlike hardware, software costs are concentrated in design rather than production.

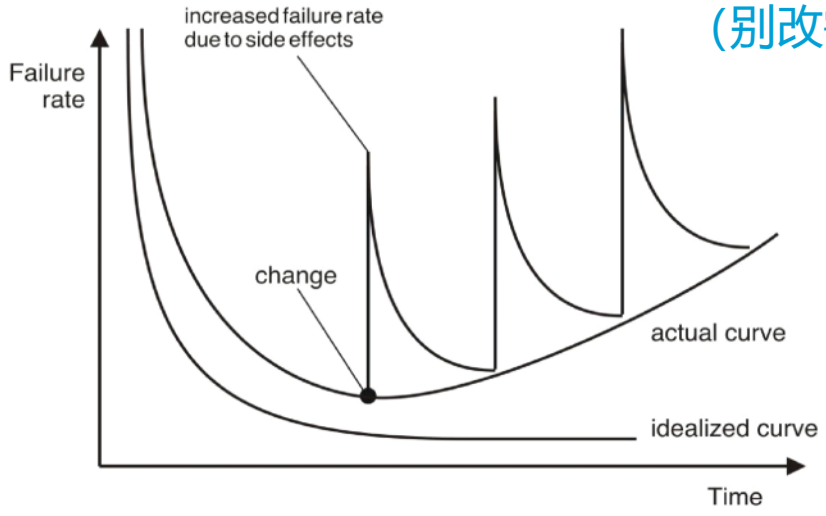
design ⟷ product



Wear out vs. Deterioration

Wear out vs. Deterioration

Software deteriorates over time



(别改需求了...,

Component Based vs. Custom Built

- Hardware products typically employ many standardized design components.
- Most software continues to be custom built.
- The software industry does seem to be moving (slowly) toward component-based construction.

Software *Complexity*

I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor(劳动) of representing it and testing the fidelity(逼真) of the representation.

If this is true, building software will always be hard. There is inherently no silver bullet.

- Fred Brooks, "No Silver Bullet"

http://www.computer.org/computer/homepage/misc/Brooks/

Furude Hanyuu 于 2018/11/13 11:23 修改

No Silver Bullet： 没有仙丹 指没有一下子就能解决问题的方法 没有银弹

Software *Changeablility*

Why must it change?

- It must be fixed to eliminate errors.
- It must be enhanced to implement new *functional* and *non-functional* requirements
- Software must be adapted to meet the needs of new computing environments or technology.
- Software must be enhanced to implement new business requirements.
- Software must be extended to make it interoperable with other more modern systems or databases.
- Software must be re-architected to make it viable(切实可行的) within a network environment.

E-Type Systems

- **E-Type Systems:**
  - Software that has been implemented in a real-world computing context and will therefore evolve over time.

Software Evolution[Lehman定律]

- The Law of Continuing Change [持续变化规律] (1974): E-type systems must be continually adapted else they become progressively less satisfactory.
- The Law of Increasing Complexity [复杂性增长规律] (1974): As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.
- The Law of Self Regulation [自我调控规律] (1974): The E-type system evolution process is self-regulating with distribution of product and process measures close to normal.

Software Evolution…

- The Law of Conservation of Organizational Stability [组织稳定性守恒规律] (1980): The average effective global activity rate in an evolving E-type system is invariant over product lifetime.
- The Law of Conservation of Familiarity [保证通晓性规律] (1980): As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of(熟悉) its content and behavior to achieve satisfactory evolution.
- The Law of Continuing Growth [持续增长规律] (1980): The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.

Software Evolution…

- The Law of Declining Quality [质量衰减规律] (1996): The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
- The Feedback System Law [反馈系统规律] (1996): E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

Source: Lehman, M., et al, "Metrics and Laws of Software Evolution—The Nineties View," Proceedings of the 4th International Software Metrics Symposium (METRICS '97), IEEE, 1997, can be downloaded from http://www.ece.utexas.edu/~perry/work/papers/feast1.pdf

Software Myths[谬论]

- Software Myths affect managers, customers (and other non-technical stakeholders) and practitioners
- Software Myths are believable because they often have elements of truth,

  but …

  - Invariably lead to bad decisions,

  therefore …

  - Insist on reality as you navigate your way through software engineering

Software Myths

- If we get behind schedule, we can add more programmers and catch up.
- A general statement about objectives is sufficient to begin building programs.
- Change in project requirements can be easily accommodated because software is flexible.

Software Myths

- Once we write a working program, we're done.
- Until I get the program running, I have no way of assessing its quality.
- The only deliverable work product for a successful project is the working program.
- Software engineering will make us create too much documentation and will slow us down.

Management Myths

- "We already have a book of standards and procedures for building software. It does provide my people with everything they need to know …"
- "If my project is behind the schedule, I always can add more programmers to it and catch up …"
- "If I decide to outsource the software project to a third party, I can just relax: Let them build it, and I will just pocket my profits …"

Customer Myths

- "A general statement of objectives is sufficient to begin writing programs - we can fill in the details later …"
- "Project requirements continually change but this change can easily be accommodated because software is flexible …"

Practitioner's Myths

- "Let's start coding ASAP, because once we write the program and get it to work, our job is done …"
- "Until I get the program running, I have no way of assessing its quality …"
- "The only deliverable work product for a successful project is the working program …"
- "Software engineering is baloney[胡扯]. It makes us create tons of paperwork, only to slow us down …"