# Software and more

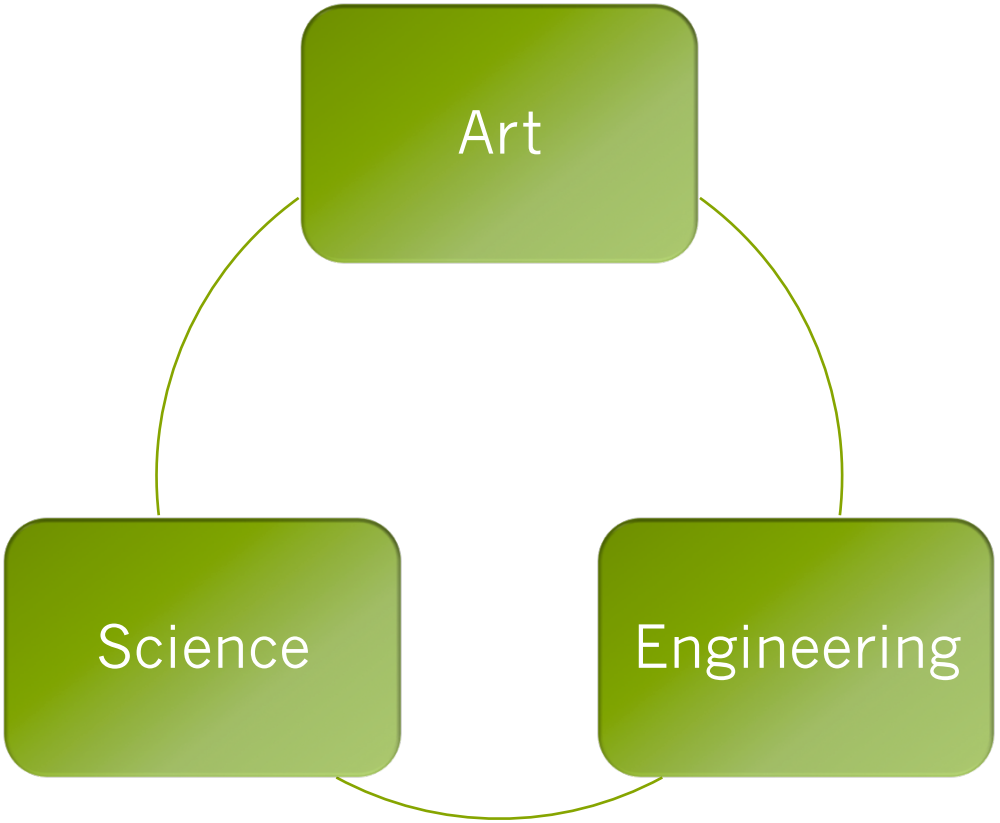
Xiao Jia

Dec. 6th, 2011

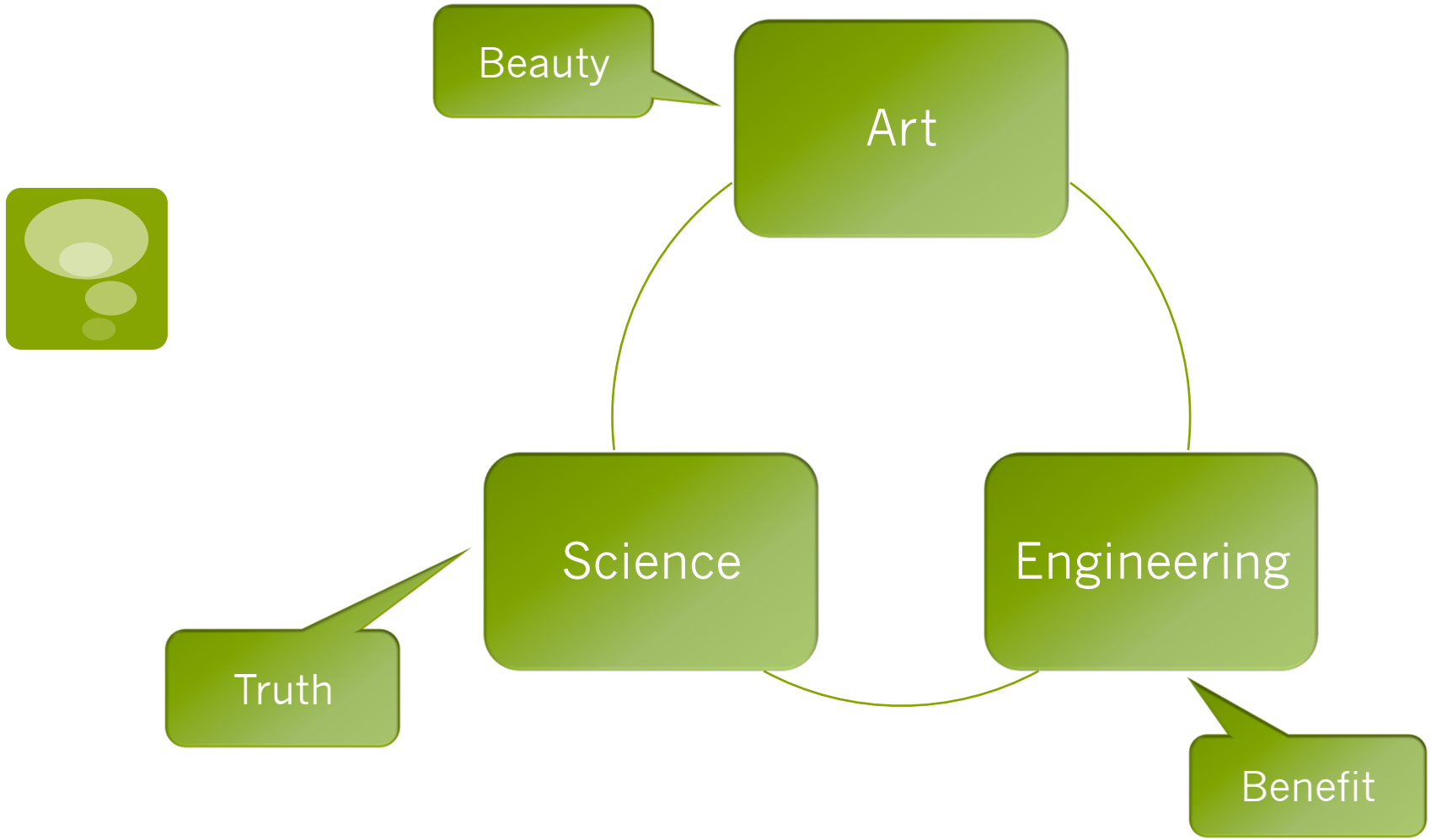
# Outline

- Science & Engineering & Art
- Problem & Solution
- Indirection
- Manpower
- Software Design Principles
- And more ...

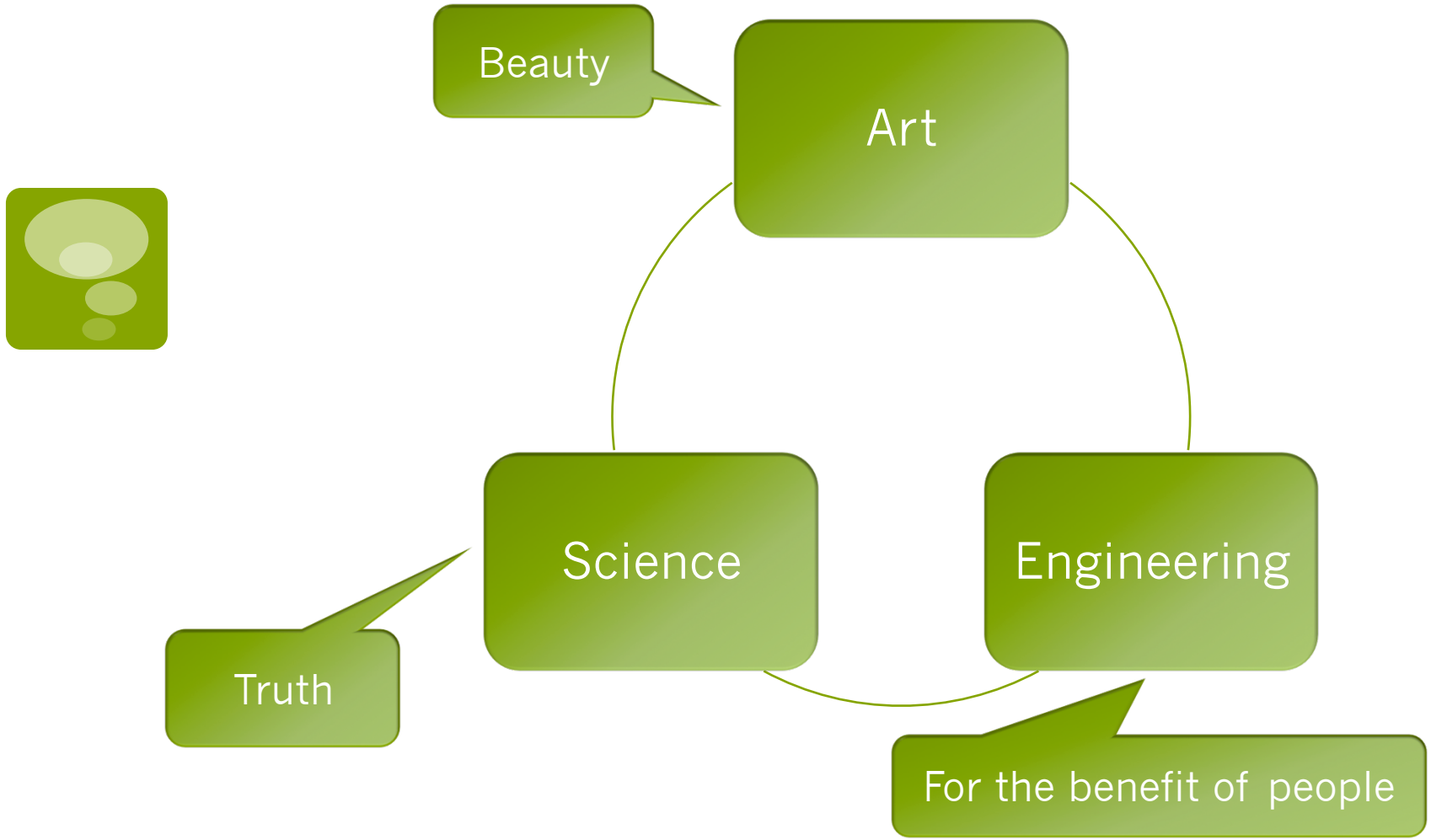
# Science & Engineering & Art



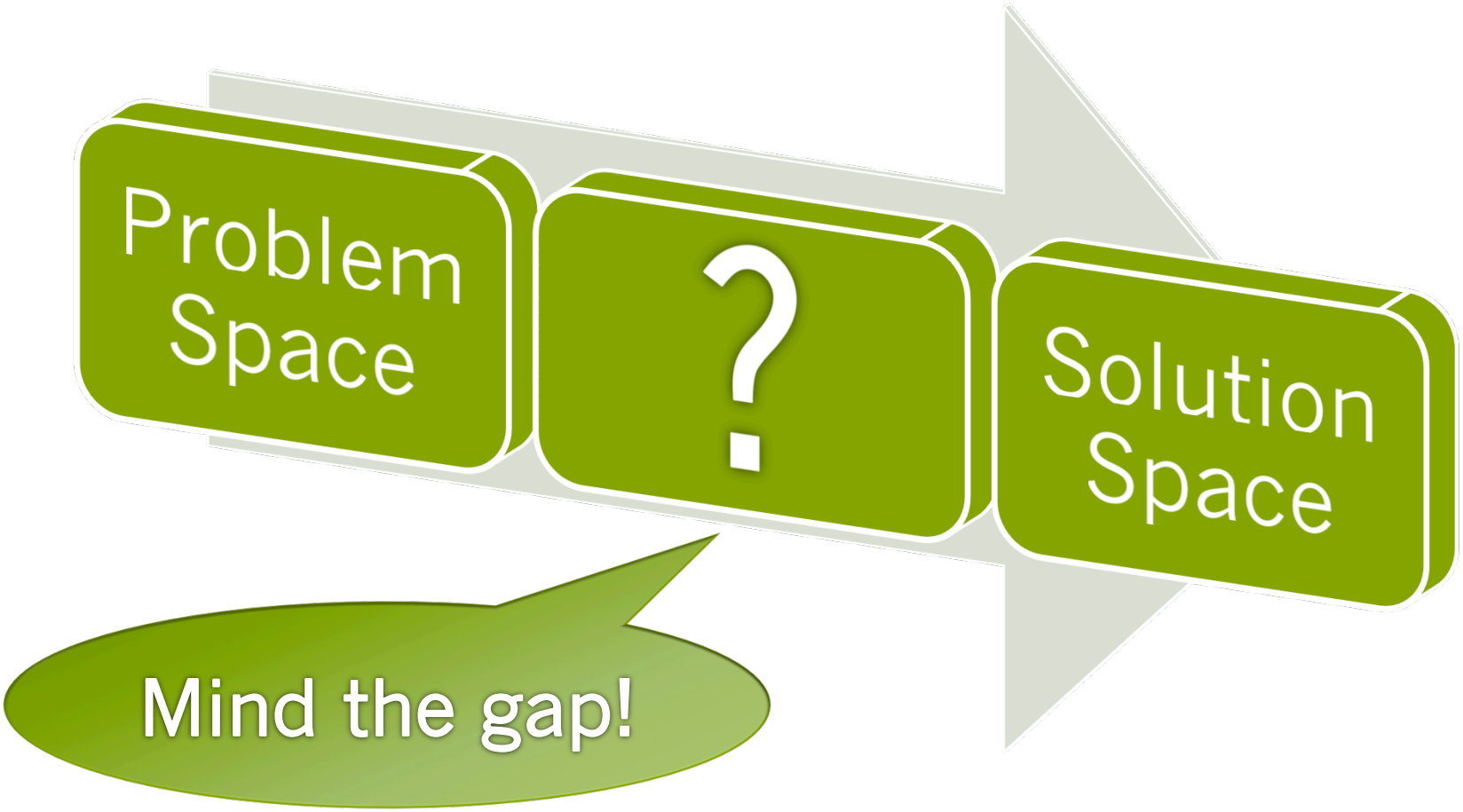
# Science & Engineering & Art



# Science & Engineering & Art



# Problem & Solution



# Fundamental Theorem of Software Engineering

- We can solve any problem by introducing an extra level of **indirection**.

*Andrew Koenig*

*C Traps and Pitfalls*

*Accelerated C++*

*"Koenig lookup"*

# Indirection

- All problems in computer science can be solved by another level of indirection

*Butler Lampson*

- ... except for the problem of too many layers of indirection

*Kevlin Henney*

- Often deliberately mis-quoted with “abstraction” substituted for “indirection”

# What is indirection?

- The ability to reference something using a name, reference, or container instead of the value itself
- Manipulating a value through its memory address
  - accessing a variable through the use of a pointer
- OOP makes use of indirection extensively
  - Dynamic dispatch (polymorphism)

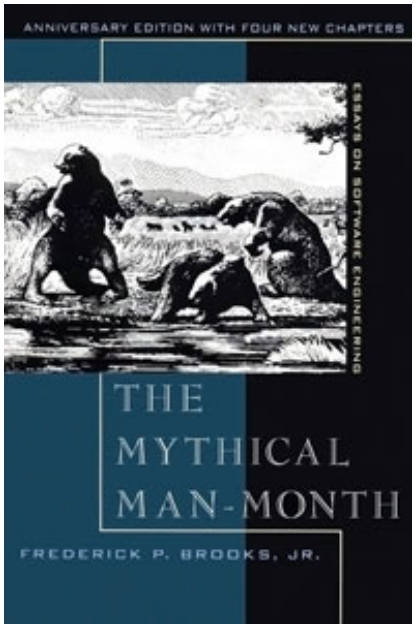
- If one woman can produce a baby in nine months, then nine women should be able to produce a baby in one month.

# Amdahl's Law

- Suppose 70% of the operations are parallelizable

$$\lim_{n \to \infty} \frac{1}{0.3 + \frac{0.7}{n}} = 3.333 \ldots$$

# Brooks' Law




Adding manpower to a late software project makes it later

*Frederick Brooks*

(Intercommunication cost)

# Do It Twice

- No software succeeds at the first time
- Software prototyping
- Second-system effect
  - Tendency of small, elegant, and successful systems to have elephantine, feature-laden monstrosities as their successors

# Zawinski's Law

- Every program attempts to expand until it can read mail.
- Those programs which cannot so expand are replaced by ones which can.

*Jamie Zawinski*

*Mozilla*

*XEmacs*

- Software bloat
  - Apple iTunes
  - Microsoft Windows

# Software Design Principles

- Open Close Principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Single Responsibility Principle
- Liskov's Substitution Principle

# Open Close Principle

- Software entities like classes, modules and functions should be **open for extension** but **closed for modifications**.

# Dependency Inversion Principle

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details. Details should depend on abstractions.
- OOP
  - Is-a relation
  - Has-a relation

# Interface Segregation Principle

- Clients should not be forced to depend upon interfaces that they don’t use.
- No fat interfaces

# Single Responsibility Principle

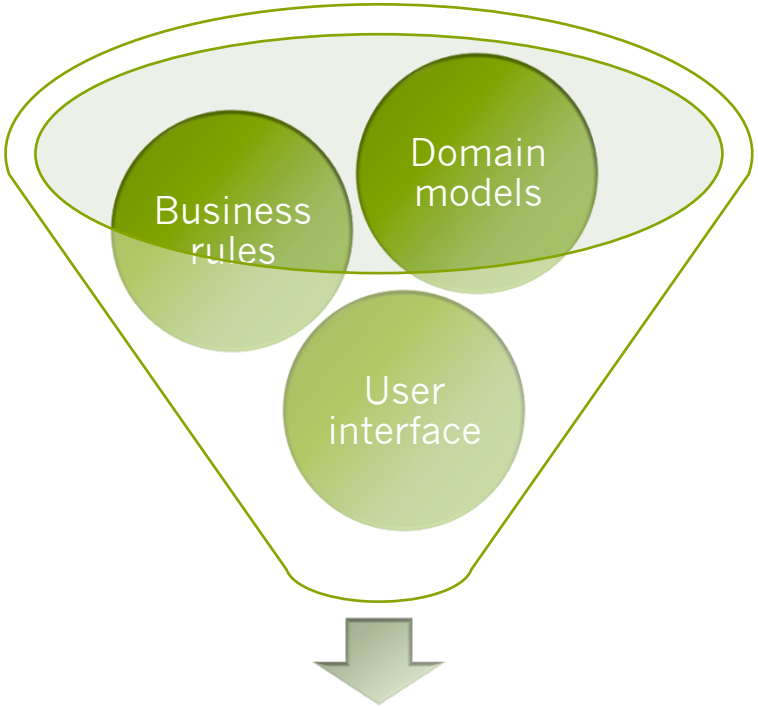
- A class should have only one reason to change.

# Liskov's Substitution Principle

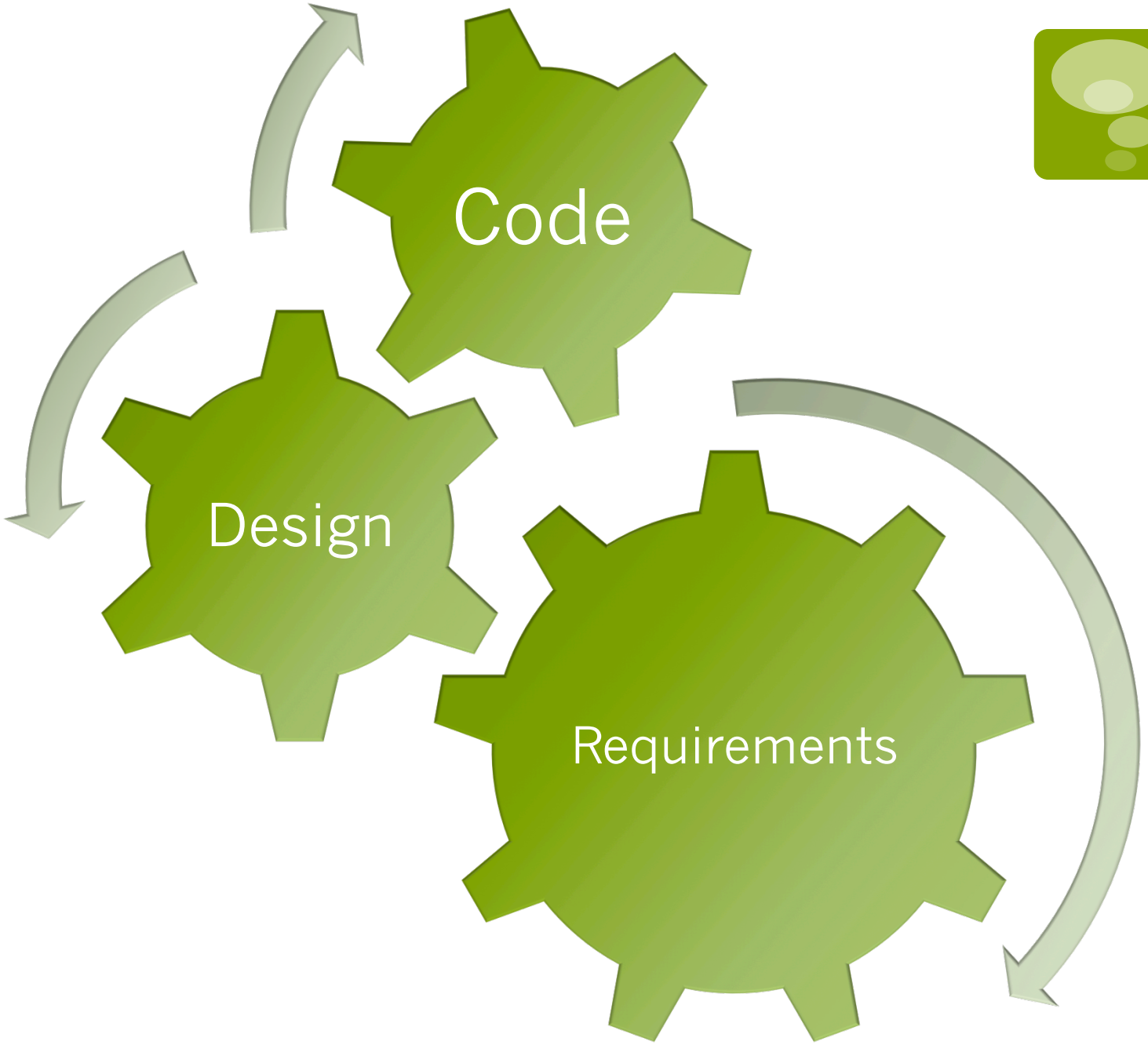
- Derived types must be completely substitutable for their base types.

*Barbara Jane Liskov*

- Object-oriented design
- Functional implementation




Software artifact


Code
Design
Requirements

# And more ...

从这里开始是中文

# 承上启下

- 需求分析和业务建模都是要描述客观事实
- 用面向对象的方法去分析与设计
- 用最合适的方式去实现
- 语言的表达能力存在巨大差异
- 要多学几门语言

# 如何提高(务虚)

- 专注
- 体会
- 感悟

# 如何提高(务实)

- 努力融入到一个有规模的项目中
  - 一定要做大的东西
  - 瓶颈往往在最初
  - 比如编译环境、运行环境的搭建
- 学会搜索和独立思考

# 做些什么？

- 真正有价值的是把几年的工作缩减到几个月，而不是把几天的事情缩短到十五分钟
- 有意思的、有技术难度的实际工程中，不要指望把已经学到的东西恰好应用上去
  - 学习能力
  - 应变能力
  - 清楚原理
  - 融会贯通
  - 学会取舍
  - 与人交流

# 我做过什么?

- 期货
  - 数据分析
  - 验证码
  - 电子表格
  - 面向特定群体的SNS
- “十年献礼工程”

# 我在做什么？

- 程序语言(理论与实现)
- 数据密集型软件系统的自动生成
- 电子商务系统中的服装自动推荐
- “十年献礼工程”

# 自动化

- 科技发展使生产过程自动化程度提高，使劳动者的智能迅速提高，大大地改变了体力劳动与脑力劳动的比例，使劳动力结构向着智能化趋势发展。
- 智能机器代替了人的部分脑力劳动，使人们的劳动方式正在经历着由机械自动化走向智能自动化，由局部自动化走向大系统管理和控制自动化的根本性变革。
- 科技革命推动了生产规模的扩大，进而推动生产的分工和协作的广泛发展，并使生产社会化的程度进一步提高，最终必然会导致生产关系的变革。

# 最后……

- ACM班是一个很好的平台
  - 外：树立品牌
  - 内：资源充足，技术氛围良好
- 爱因斯坦：
  - 关心人的本身，应当始终成为一切技术上奋斗的主要目标。
  - 关心怎样组织人的劳动和产品分配这样一些尚未解决的重大问题，用以保证我们科学思想的成果会造福于人类，而不致成为祸害。

# Thank you for listening

Always Challenge Miracles