1-Introduction

Designing Software [for large systems]

1. – Principles
2. – Techniques
3. – Tools
4. – Practices

Libraries: very well tested code solving real problems
Library developers: help many heroes at once!
Dressed with the code developed by everyone else, Hero can win!
How to be a library developer
• Different mindset: no main!
code is parametric
no clear behavior
sort(List<T> list, Comparator<? super T> c)

How to use libraries
• Adapting code without modifying it, Objects as operations Reading documentation

2 – Refactoring and Version Control

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Its heart is a series of small behavior preserving transformations
In computer programming, code smell is any symptom 症状in the source code of a program that possibly indicates a deeper problem
• Big Methods

1. – code too long
2. – code doing too much

• Big Classes

1. – too many variables
2. – "God" classes

- Duplicate Code
  1. – How many times should code do anything?
  2. – "Once and only once"

```
public int method1() {
    int x = doThis();
    int y;
    if(..){y=x*2;}
    else {y=x+1;}
    return y+x;}
```

extract me

```
public int method1() {
  int x = doThis();
  int y = computeY(x);
  return x+y;}
private int computeY(int x) {
  if(..){return x*2;}
  else {return x+1;}}
```

Avoid very nested code

```
public float averageMark(Student s) {
  if(s!=null){
    if(s.isEnrolled("ECS")){
      if(s.hasPassed("SWEN489"){
        float tot=0;
        for(float mark:s.marks()){
          tot+=mark;
          }
        return tot/s.marks().size();
        }
      }
    }
  return 0;
}
```

No automatic solution here

```
public double averageMark(Student s) {
  assert s!=null;
  if(!s.isEnrolled("ECS")){throw new Error("..");}
  if(!s.hasPassed("SWEN489"){return 0d;}
  double tot=s.marks().stream()
    .mapToDouble(e->e).sum();
  return tot/s.marks().size();
}
```

Better code

1. – be negative: start by checking the absence of errors (both assertions/preconditions and regular errors/exceptions)
2. – try to run away: consider simple cases FIRST
3. – be lazy: use libraries when possible (streams here)

Well defined roles

Difference between printing a string and returning a string?

Difference between exception throw and exception print/log?

Few methods (may be only main+tests) should print things. All the other methods should just throw exceptions and/or returning values or messages.

– leak an exception / assertion / error

– DO NOT PRINT AN ERROR MESSAGE

If a method can not complete its objective

– DO NOT PRINT A SUCCESS MESSAGE

2-Version Control

Git is a Version Control System

– It helps coordinate projects developed in teams

– It provides many useful pieces of functionality:

- Project files stored in distributed repositories – Repositories can be accessed remotely
- Complete history of all changes made
- Each change has a log entry associated with it
- Team members can "synchronize同步 "their code changes easily

– It replaces old and somewhat antiquated过时的 systems, like SVN and CVS
– It is freely available as an open source project

1. Must record changes made to enable integration of each others changes
2. must keep log of each changes they make
3. so if they need to undo one, they know why they made it in the first place
4. Also, good to keep original code before and after every change. So, can put back in working version

3-API Design

Simplicity
Over Engineering

Overengineering (or over-engineering) is the designing of a product to be more robust or complicated than is necessary for its application, either (charitably) to ensure sufficient factor of safety, sufficient functionality, or because of design errors. Overengineering can be desirable when safety or performance on a particular criterion is critical, or when extremely broad

functionality is required, but it is generally criticized from the point of view of value engineering as wasteful. As a design philosophy, such overcomplexity is the opposite of the less is more school of thought

Premature Optimisation *（早期优化）*