04834580 Software Engineering (Honor Track) 2024-25

Design Principles

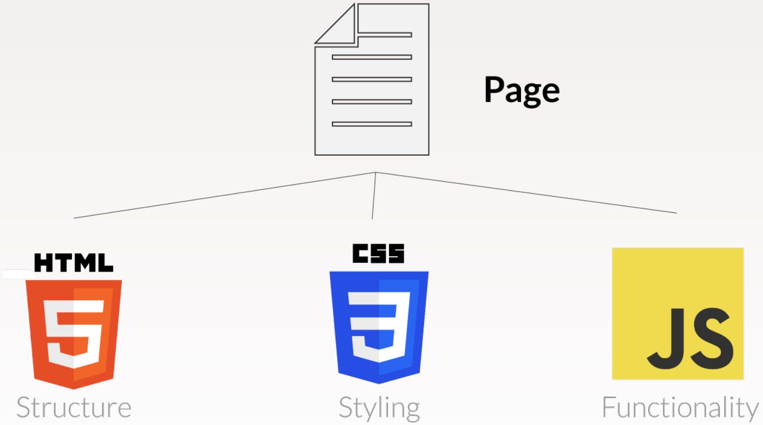
Sergey Mechtaev
mechtaev@pku.edu.cn
School of Computer Science, Peking University

Separation of Concerns

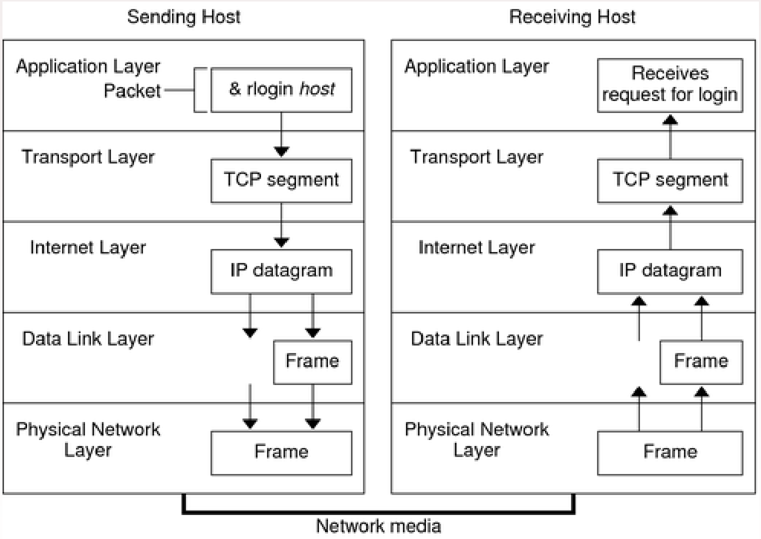
Edsger W. Dijkstra in “On the role of scientific thought” [1]:

> *We know that a program must be correct and we can study it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. But nothing is gained — on the contrary! — by tackling these various aspects simultaneously. It is what I sometimes have called “the separation of concerns”.*

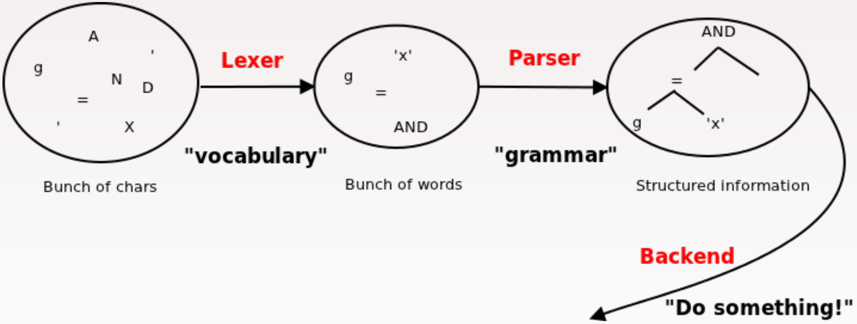
Separation of Concerns: HTML, CSS, JavaScript



Separation of Concerns: TCP/IP Protocol Stack



Separation of Concerns: Parsers



Single Responsibility Principle

Robert C. Martin, the originator of the term [2, 3], expresses the principle as

> *A class should have only one reason to change.*

Illustrating Example:

> *As an example, consider a module that compiles and prints a report. Imagine such a module can be changed for two reasons. First, the content of the report could change. Second, the format of the report could change. These two things change for different causes. The single responsibility principle says that these two aspects of the problem are really two separate responsibilities, and should, therefore, be in separate classes or modules. It would be a bad design to couple two things that change for different reasons at different times.*

Open/Closed Principle

Formulated by Bertrand Meyer [4]:

> *Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.*

Open/Closed Principle: Negative Example

```java
class Drawing {
    public void drawShape(String shape) {
        if ("circle".equals(shape)) {
            System.out.println("Drawing a circle");
        } else if ("rectangle".equals(shape)) {
            System.out.println("Drawing a rectangle");
        }
        // Adding new shapes requires modifying this method:
        // else if ("triangle".equals(shape)) {
        //     System.out.println("Drawing a triangle");
        // }
    }
}
```

Open/Closed Principle: Improved Example

北京大学

```java
interface Shape {
    void draw();
}

// Implement different shapes
class Circle implements Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }
}

class Rectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a rectangle");
    }
}

// Add a Drawing class that works with the Shape abstraction
class Drawing {
    public void drawShape(Shape shape) {
        shape.draw();
    }
}
```

Two Example of Inheritance

北京大学

```java
class Rectangle {
  private double width;
  private double height;

  public Rectangle(double width, double height) {
    this.width = width;
    this.height = height;
  }

  ...

  public double calculateArea() {
    return width * height;
  }
}

class Square extends Rectangle {
  public Square(double side) {
    super(side, side);
  }
}
```

```java
class Square {
  private double side;

  public Square(double side) {
    this.side = side;
  }

  public double calculateArea() {
    return side * side;
  }
}

class Rectangle extends Square {
  private double height;

  public Rectangle(double width, double height) {
    super(width);
    this.height = height;
  }

  @Override
  public double calculateArea() {
    return getWidth() * height;
  }
}
```

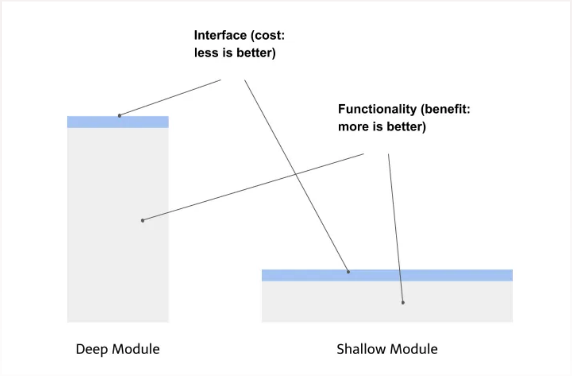
Liskov Substitution Principle

Barbara Liskov and Jeannette Wing formulated this principle in “A behavioral notion of subtyping” [5]:

> *Subtype Requirement: Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.*

Shallow And Deep Interfaces

Deep modules provide a lot of functionality behind a simple interface, while shallow modules have a relatively complicated interface.


Interface (cost: less is better)
Functionality (benefit: more is better)
Deep Module
Shallow Module

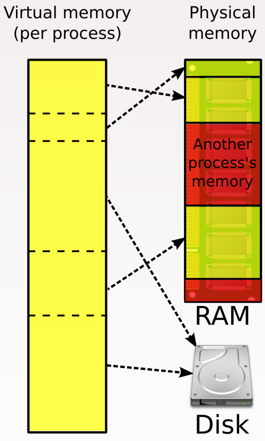
Shallow And Deep Interfaces: UNIX I/O

```
int open(const char* path, int flags, mode_t permissions)
int close(int fd)
ssize_t read(int fd, void* buffer, size_t const)
ssize_t write(int fd, const void* buffer, size_t count)
off_t lseek(int fd, off_t offset, int referencePosition)
```

Abstracts:

- On-disk representation, disk block allocation
- Directory management, path lookup
- Permission management
- Disk scheduling
- Block caching

Shallow And Deep Interfaces: Virtual Memory



Linus Torvalds on Good Taste

Linus Torvalds in TED interview [6]:

> *sometimes you can see a problem in a different way and rewrite it so that a special case goes away and becomes the normal case, and that's good code*

```c
void remove(list *l, list_item *target)
{
  list_item *cur = l->head, *prev = NULL;
  while (cur != target) {
    prev = cur;
    cur = cur->next;
  }
  if (prev)
    prev->next = cur->next;
  else
    l->head = cur->next;
}
```

```c
void remove(list *l, list_item *target)
{
  list_item **p = &l->head;
  while (*p != target)
    p = &(*p)->next;
    *p = target->next;
}
```

YAGNI (You Aren't Gonna Need It)

Ron Jeffries, a co-founder of extreme programming (XP):

> *Always implement things when you actually need them, never when you just foresee that you will need them.*

Other Principles

- DRY (Don’t Repeat Yourself)
- KISS (Keep it simple, stupid!)

References I

[1] Edsger W Dijkstra and Edsger W Dijkstra.
On the role of scientific thought.
Selected writings on computing: a personal perspective, pages 60–66, 1982.

[2] Micah Martin and Robert C Martin.
Agile principles, patterns, and practices in C.
Pearson Education, 2006.

[3] Robert C. Martin.
The single responsibility principle.
`https://blog.cleancoder.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html`, 2014.

[4] Bertrand Meyer.
Object-oriented software construction, volume 2.
Prentice hall Englewood Cliffs, 1997.

References II

[5] Barbara H Liskov and Jeannette M Wing.
A behavioral notion of subtyping.
ACM Transactions on Programming Languages and Systems (TOPLAS), 16(6):1811–1841, 1994.

[6] Linus Torvalds.
The mind behind linux.
https://www.ted.com/talks/linus_torvalds_the_mind_behind_linux, 2016.