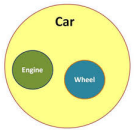
4. Abstraction

- Identify the important aspects of a phenomenon and ignore its details
  - Special case of separation of concerns



Abstraction Examples

Java Class

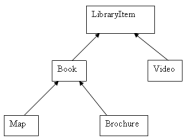
```
class Employee
{
   public String name;
   public int age;
   public long salary;
   public long companyID;
   Address address;
   public byte[] photo;
}
```

```
#include<stdio.h>
#include<conio.h>

void main()

{
 clrscr();
 printf("I am an engineer");
 getch();

}
```

API (printf)

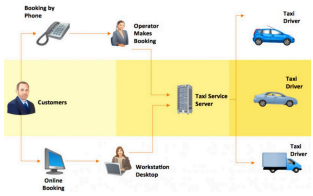
Project Requirement:
Build a website for book selling.

Abstraction Examples



Multiple-level abstraction

Abstraction Examples



Workflow

4. Abstraction

Software qualities benefit from abstraction:

- Functionality
  - Correctness
  - Reliability
  - Robustness
- Maintainability
  - Repairability
  - Evolvability
- Usability
- Performance
- Verifiability
- Reusability
- Portability
- Understandablity
- Interoperability
- Productivity
- Timeless
- Visibility

5. Anticipation of change

- Ability to support software evolution requires anticipating potential future changes

Algorithm ← Anticipation of Algorithm Change

Insertion sort
Merge sort
Heapsort
Quicksort
Counting sort
Radix sort
Bucket sort

```
sort(a[])
{
   mergesort(a[]);
}
```

- *Exercise: Which nature of software results in this principle ?*

5. Anticipation of change

- Ability to support software evolution requires anticipating potential future changes
  - Needs tools to manage changes
  - highly recommended for your project

GitHub

Atlassian Bitbucket

git

SUBVERSION

6. Generality

```
{
    "empid": "SJ011MS",
    "personal": {
        "name": "Smith Jones",
        "gender": "Male",
        "age": 28,
        "address":
        {
            "streetaddress": "7 24th Street",
            "city": "New York",
            "state": "NY",
            "postalcode": "10038"
        }
    },
    "profile": {
        "designation": "Deputy General",
        "department": "Finance"
    }
}
```

Task: Parse the data

6. Generality

```
{
    "empid": "SJ011MS",
    "personal": {
        "name": "Smith Jones",
        "gender": "Male",
        "age": 28,
        "address":
        {
            "streetaddress": "7 24th Street",
            "city": "New York",
            "state": "NY",
            "postalcode": "10038"
        }
    },
    "profile": {
        "designation": "Deputy General",
        "department": "Finance"
    }
}
```

V.S.

ANTLR

Parser Generator

6. Generality

- While solving a problem, try to discover if it is an instance of a more general problem whose solution can be *reused* in other cases

An electronic sign displaying the year incorrectly as 1900 on 3 January 2000 in France

95 ***v.s.*** 1995

6. Generality

Task: Sort the data

165 cm	165	***V.S.***	165.4 cm
178 cm	178		178.23 cm
173 cm	173		173.2 cm
186 cm	186		186.145 cm
160 cm	160		160.24 cm
183 cm	183		183.29 cm
…	…		…
Product size	Radix Sort		Quick/Merge Sort…

Generality vs Performance / Cost

Generality vs Performance/Cost

```
Sorting () {
  int a[];
  …
  if (a[j]<a[j-1])
    swap(a[j], a[j-1]);
  …
}
```

```
Sorting () {
  double a[];
  …
  if (a[j]<a[j-1])
    swap(a[j], a[j-1]);
  …
}
```

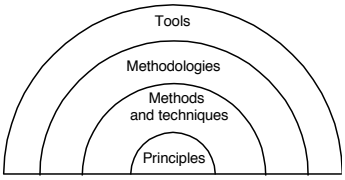
7. Incrementality

- Process proceeds in a stepwise fashion (*increments*)
- Examples (process)
  - deliver subsets of a system early ➔ get early feedback from expected users, then add new features incrementally
  - first functionality, then performance

Key principles

- Rigor and formality
- Separation of concerns
- Modularity
- Abstraction
- Anticipation of change
- Generality
- Incrementality

A visual representation



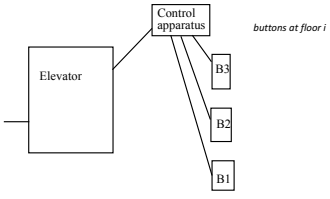
Elevator System : Rigor & formality

- Quite relevant: it is a *safety critical system*
  - Define requirements
    - must be able to carry up to 1000 lbs. (safety alarm and no operation if overloaded)
    - emergency brakes must be able to stop elevator within 3 ft. and 2 sec. in case of cable failures
  - Later, verify their fulfillment

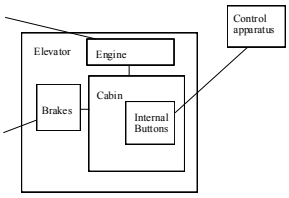
Elevator System : Separation of concerns

- Try to separate
  - safety
  - performance
  - usability (e.g, button illumination)
  - cost
- although some are strongly related
  - cost reduction by using cheap material can make solution unsafe

Elevator System : A modular structure

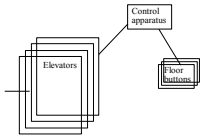


Elevator System : Abstraction



Elevator System : Anticipation of change, generality, incrementality

- Make the project parametric w.r.t. the number of elevators (and floor buttons)



Key principles

1. Rigor and formality
2. Separation of concerns
3. Modularity
4. Abstraction
5. Anticipation of change
6. Generality
7. Incrementality

Illustrate these on a compiler

The software production process

(Chapter 7)

Outline

- What is *a software process*
- Why we need *software process models*
- Process model examples:
  - Waterfall, Evolutionary, Rapid Development/Extreme Programming

Earliest approach: Code&Fix

- Write code
- Fix it to eliminate any errors that have been detected, to enhance existing functionality, or to add new features

What are the problems with this approach?

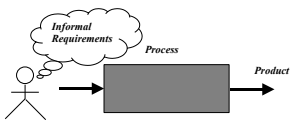
The needs for software process models

- Symptoms of inadequacy: the software crisis
  - Code&Fix doesn't work
  - scheduled time and cost exceeded
  - user expectations not met (user ≠ developer)
  - poor quality
- The size and economic value of software applications requires *process models*
  - Remember: SW industry revenue > **7x** civil engineering industry revenue

Software process model

- Organize the software life cycle by
  - defining activities involved in software production
  - order of activities and their relationships

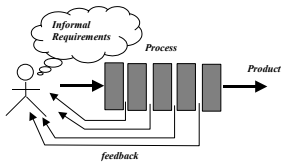
Process as a "black box"



Problems:

- Assumes requirements can be fully understood prior to development

Process as a "white box"

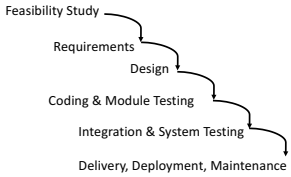


Advantages:

- improved visibility
- feedback from the customer

Traditional Software Process Models

Waterfall models



Waterfall: Feasibility Study

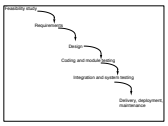
- Support the decision of a new development
  - Perform preliminary requirements analysis
  - Produces a Feasibility Study Document
    1. Definition of the problem
    2. Alternative solutions and their expected benefits
    3. Required resources, costs, and delivery dates in each proposed alternative solution

Waterfall: Requirements Engineering

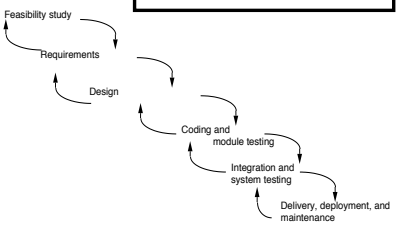
- Understand interface between the application and the external world
- Understand the application domain
- Identify the main stakeholders
- Focus *what* qualities, NOT on *how*

Critical evaluation of the waterfall model

+ SW process subject to discipline, planning, and management
+ Postpone implementation to after understanding objectives
– Linear, rigid, monolithic, document driven
  - no feedback
  - no parallelism
  - a single delivery date
  - no change anticipation



Waterfall with feedback



Evolutionary models

- Product development evolves through increments
  - "do it twice" (F. Brooks, 1995)
  - evolutionary prototype
- Evolutionary process model (B. Boehm, 1988)
  "model whose stages consist of expanding increments of an operational software product, with the direction of evolution being determined by operational experience"
- Many variants available

Modern Software Process Models

Agile Software Development

“ Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan ”

- Manifesto for Agile Software Development

Agile Methods

- Adaptive software development (ASD)
- Agile modeling
- Agile Unified Process (AUP)
- Crystal Clear methods
- Disciplined agile delivery
- Dynamic systems development method (DSDM)
- Extreme programming (XP)
- Feature-driven development (FDD)
- Lean software development
- Kanban
- Scrum
- Scrumban

Extreme Programming (XP)

- One of best-known Agile methods
- Push good practice, e.g., *iterative development*, to ‘extreme’ levels.
  - New versions may be built several times per day
  - Increments delivered to customers every 2 weeks

XP: Planning

Requirements document

User Stories

- For time estimate purpose
- Written by customers
- Similar to usage scenarios
- Much less details

Story card for document downloading

Downloading and printing an article

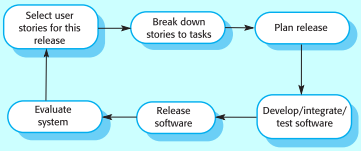
First, you select the article that you want from a displayed list. You then have to tell the system how you will pay for it - this can either be through a subscription, through a company account or by credit card.

After this, you get a copyright form from the system to fill in and, when you have submitted this, the article you want is downloaded onto your computer.

You then choose a printer and a copy of the article is printed. You tell the system if printing has been successful.

If the article is a print-only article, you can't keep the PDF version so it is automatically deleted from your computer.

XP: Iterative Development



Task cards for document downloading

Task 1: Implement principal workflow

Task 2: Implement article catalog and selection

Task 3: Implement payment collection

Payment may be made in 3 different ways. The user selects which way they wish to pay. If the user has a library subscription, then they can input the subscriber key which should be checked by the system. Alternatively, they can input an organisational account number. If this is valid, a debit of the cost of the article is posted to this account. Finally, they may input a 16 digit credit card number and expiry date. This should be checked for validity and, if valid a debit is posted to that credit card account.

XP: Testing

- Test-first development
  - Writing tests before code clarifies requirements
  - All previous and new tests are automatically run when new functionality is added
- Incremental test development from scenarios
- User involvement in test development and validation

Test case

Test 4: Test credit card validity

Input:
A string representing the credit card number and two integers representing the month and year when the card expires

Tests:
Check that all bytes in the string are digits
Check that the month lies between 1 and 12 and the year is greater than or equal to the current year.
Using the first 4 digits of the credit card number, check that the card issuer is valid by looking up the card issuer table. Check credit card validity by submitting the card number and expire date information to the card issuer

Output:
OK or error message indicating that the card is invalid

XP: Pair Programming

- Helps develop *common ownership* of code
- Serves as informal review
- Support refactoring
- Development productivity
  ≈ two people working independently
  [Williams, et al., 2000]

XP: Summary (I)

Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'.
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.

XP: Summary (II)

Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP: Advantages

- Accelerated delivery of customer services
  – Each increment delivers the highest priority functionality to the customer
- User engagement with the system
  – Users involved in the development
    - System more likely to meet requirements
    - More user commitment

XP: Problems

- Management problems
  – "80% ready"
  – documentation brushed aside
  – Barrier to entry
- Contractual problems
  – No carved-in-stone specification
- Validation problems
  – Without a specification
- Maintenance problems
  – Continual change tends to corrupt software

Project tips

- Use version control, e.g., Subversion, Git
  – Have an *always working* branch
  – Don't break compilation when checking in
- Set up an online communication channel for the whole group
- Consider designating a team member as manager/release engineer
- Popular processes
  – Synchronize-and-stabilize
  – XP: pair programming, peer validation
    - "Stayed focused on task, kept each other honest, answered each other's questions"
- Open source components -> avoid re-inventing the wheel
  – Jquery, Joomla, Rails, WAMP/LAMP
- Write user manual in parallel with tests
  – Automate the test suite, e.g., JUnit, Hudson, cURL
- Modularity is key -> separation of concerns ->parallelism