Software Development Process

- “A structure imposed on the development of a software product.”
- Importance of the use of software development process:
  - Project **Schedule**
  - Development **Cost** $$$
  - Software **Quality**

General Approach

- **Requirement**
  - A desired feature, attribute or behaviour of a software product
- **Specification**
  - Precise description of the functionality of the software product
- **Design**
  - How do they work?
- **Implementation**
  - Mapping a design to code.
  - Not the reverse!
- **Test**
  - Does the software product meet the Specification?

Cost of fixing a fault at each phase
Requirement analysis
- Most customers have little knowledge of software development;
- Software development is complex and expensive, thus better identify potential issues in advance;
- To recognize unreasonable, incomplete, ambiguous or contradictory requirements;
- Goal: To set Deadline and Cost which both you and the client are happy 😊
- Other constraints, e.g. security or reliability

Specification analysis
- Precisely describes what the product is supposed to do
- Constitutes a contract to the client
- Basic elements:
  - Inputs
  - Required outputs
  - Constraints

Design analysis
- To design the internal structure of the software product;
- To determine behaviours of modules and their interactions;
- Things to be considered:
  - Algorithms;
  - Data structures;
  - Data and Control flows;
  - Modules and Interfaces;
  - and so on.

Testing techniques
- Black-box testing
  - To test a module without looking its internal design
- White-box testing
  - To test the internal structure of a module, e.g. logic errors
An example

- Design a program that will calculate the average, smallest and largest number in a set of user-inputted numbers.
- This is an incomplete and vague statement and needed to be improved.

Requirement

- Let’s re-phase our question:

Design a program that will calculate the average in a set of user-inputted positive integer, and further display the largest and smallest integers. The program will terminate when a non-positive integer is entered.

Specification

- Inputs:
  - Positive integers
  - Any non-positive integer to quit
- Outputs:
  - Average (double value!)
  - Largest integer
  - Smallest integer
- Constraints
  - Sum of all inputs must not exceed the max integer value in C

Design analysis

- Initialise sum, largest and smallest
- read an integer
- while (input is a positive integer)
  - increment sum by the value of input
  - if (largest < input) swap input and largest
  - if (smallest > input) swap input and smallest
- read a integer
- Display sum, largest and smallest
Common errors (1)

- Students normally initialise *largest* and *smallest* to be zero.
- Which variable, *largest* or *smallest* is incorrectly initialised?
- Solution:
  Set the value of *smallest* as big as possible

Common errors (2)

- What if…
  - if (*largest* < *input*) swap *input* and *largest*
  - else swap *input* and *smallest*

Common errors (3)

- if (*largest* < *input*) swap *input* and *largest*
  - else if (*smallest* > *input*) swap *input* and *smallest*

- Assume *largest*=0 and *smallest*=99, initially.
- User inputs: <5, 0>
- Outputs:
  - Average=5; largest=5; smallest=99 (wrong)

Basic program structure

```c
/* Header Comments
 *----------------------------------------------------------*/
#include files
#include global declarations

int main()
{
   declarations and executable statements
   return 0;
}
//end block of main
```
C program

Compilation

- Compiler
  Converts source program to object program

- Linker
  Converts object program to executable program

Types of errors

- Syntax errors
  - reported by the compiler

- Linker errors
  - reported by the linker

- Run-time errors
  - reported by the operating system

- Logic and Design errors
  - not reported;

Testing

- To discover logic and design errors in the product;
- Some syntax errors are “invisible” to compilers
- E.g.
  ```c
  int i=0, out=0;
  for(i=0;i<10;i++)
    out+=i; // wrong syntax
  printf("Output: %d",out); // Output: 9
  ```
Sample test cases

<table>
<thead>
<tr>
<th>Input set</th>
<th>Desired output</th>
<th>Actual output</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;8,2,5,0&gt;</td>
<td>5.00</td>
<td>5.00</td>
</tr>
<tr>
<td>&lt;12,11,24,97,33,0&gt;</td>
<td>35.40</td>
<td>35.40</td>
</tr>
<tr>
<td>&lt;1,1,1,1,1&gt;</td>
<td>1.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Can it be better?

- Comment the code;
- Verify user inputs;
- More user-friendly output