Virtual Function: Addendum (2)

```
class EncryptBase {
protected:
  int key_;
public:
  virtual void encrypt(const string &msg) = 0;
  virtual void decrypt(const string &msg) = 0;
  void erase_key() { ... }
  virtual ~EncryptBase() = default;
};
```

```
class EncryptDES : public EncryptBase {
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
  ~EncryptDES() override;
};
```

```
class EncryptAES : public EncryptBase {
protected:
  vector<int> key_matrix4x4_;
public:
  void encrypt(const string &msg) override;
  void decrypt(const string &cipher) override;
  virtual void calc_key_matrix() { ... }
  ~EncryptAES() override { ... }
};
```

```
class EncryptSecureAES : public EncryptAES {
private:
  vector<int> second_key_matrix4x4_;
public:
  void encrypt(const string &msg) final;
  void decrypt(const string &cipher) final;
  void calc_key_matrix() final { ... }
  ~EncryptSecureAES() final { ... }
};
```

Virtual Functions: Addendum (3.)

```
int main() {
  std::vector<Shape *> shapes;

  shapes.push_back(new Circle(0, 0, 10));
  shapes.push_back(new Circle(5, 5,  6));
  shapes.push_back(new Triangle(0, 0, 0, 1, 2, 1));

  for(auto &shape_ptr : shapes) {
    shape_ptr->draw();
  }
}
```

What is missing ?

Virtual Functions: Addendum (3.)

```
int main() {
  std::vector<Shape *> shapes;

  shapes.push_back(new Circle(0, 0, 10));
  shapes.push_back(new Circle(5, 5,  6));
  shapes.push_back(new Triangle(0, 0, 0, 1, 2, 1));

  for(auto &shape_ptr : shapes) {
    shape_ptr->draw();
  }

  for(auto &shape_ptr : shapes) {
    delete shape_ptr;
  }
}
```

What is missing ?

Yes, but deleting what?

```
shape_ptr->~Shape();
```

Virtual Function: Addendum (1)

- Destructor of the base class must be virtual
  - Otherwise, you may have potential memory leakage!
  - Destructor are called in sequence, from derived class to base class

```
INFO: delete Circle()
INFO: delete Shape()
```

- Don't place virtual keyword on destructors of non-polymorphic classes.

- Good destructor is in most case the default destructor.

Regular Expression (1)

- Validate that string match a specific pattern
- Extract one of more sub-strings in a string
- Rely on meta-characters to express pattern

Characters	Description	Matches
`.`	not newline	any character except line terminators (LF, CR, LS, PS).
`\d`	digit	a decimal digit character.
`\D`	not digit	any character that is not a decimal digit character
`\s`	whitespace	a whitespace character.
`\S`	not whitespace	any character that is not a whitespace character
`\w`	word	an alphanumeric or underscore character
`\W`	not word	any character that is not an alphanumeric or underscore character
`\c`	character	the character as it is, without interpreting its special meaning. Needed for: `^ $ \ . * + ? ( ) [ ] { } |`

Characters	Description	Matches
`*`	0 or more	Match preceding pattern 0 or more times
`+`	1 or more	Match preceding pattern 1 or more times
`?`	0 or 1	Optional pattern
`{n}`	N	Match exactly n times
`(patter)`	Group	Create a group with a backreference

Regular Expression (2)

- Validate that a string matches a correct email

```
elec4.cpp@gmail.com
```

```
R"(.+@.+\.\w+)"
```

- Extract tag content in xml fragment

```
<tag id=123>Hello World</tag>
```

```
R"(<(tag).*>(.+)</\1>)
```

Regular Expression (3)

- Beware of the RE pitfalls

What is missing ?

```
void
extract_email(const string &line)
{
  std::regex re_email(R"((.+@.+\.\w+))");
  std::smatch matched;
  bool found_it = std::regex_search(line, matched, re_email );
  cout << "Status " << found_it << " for email in string " << line  << '\n';
  cout << "Before email    is " << matched.prefix() << '\n';
  cout << "Extracted email is " << matched[0] << '\n';
  cout << "After email     is " << matched.suffix() << '\n';
}
```

```
INPUT STRING: field1,foo.bar@example.com,1234.56
Status 1 for email in string field1,foo.bar@example.com,1234.56
Before email    is
Extracted email is field1,foo.bar@example.com,1234.56
After email     is
```

Regular Expression (4)

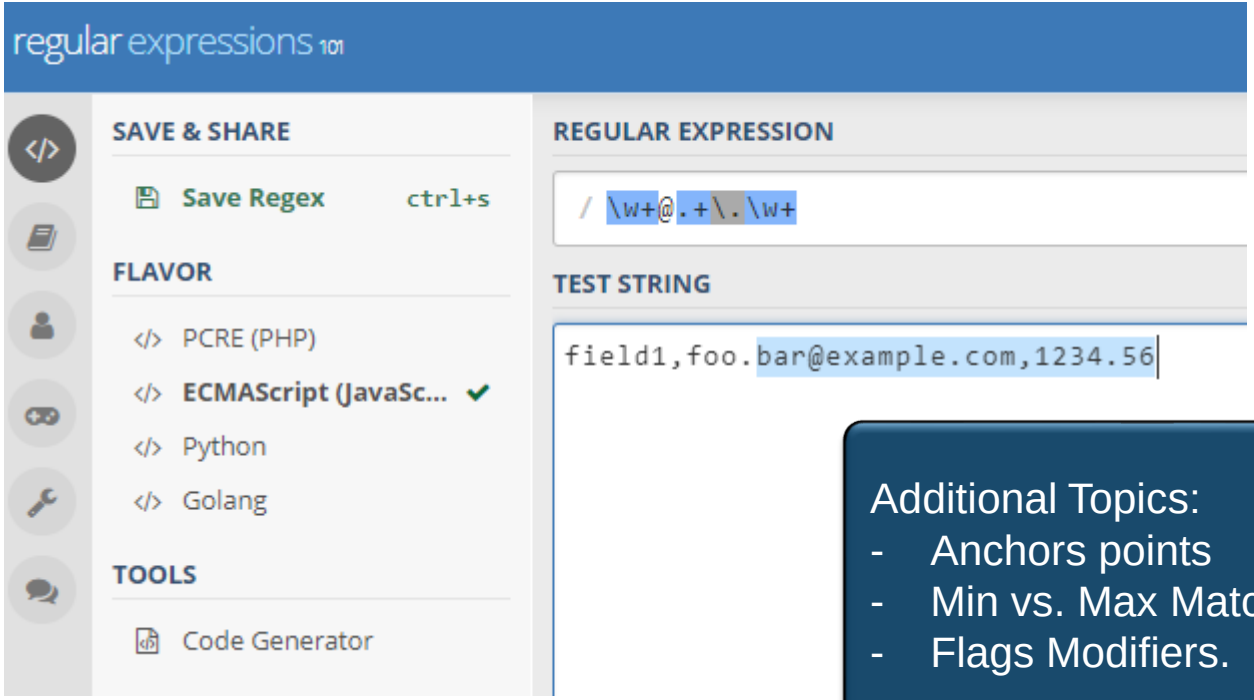
- Beware of the RE pitfalls

Test your regular expression

```
void
extract_email(const string &line)
{
  std::regex re_email(R"((\w[\w\.]*@\w+\.\w+))");
  std::smatch matched;
  bool found_it = std::regex_search(line, matched, re_email );
  cout << "Status " << found_it << " for email in string " << line  << '\n';
  cout << "Before email    is " << matched.prefix() << '\n';
  cout << "Extracted email is " << matched[0] << '\n';
  cout << "After email     is " << matched.suffix() << '\n';
}
```

```
INPUT STRING: field1,foo.bar@example.com,1234.56
Status 1 for email in string field1,foo.bar@example.com,1234.56
Before email    is field1,
Extracted email is foo.bar@example.com
After email     is ,1234.56
```

Regular Expression (5)



Additional Topics:
- Anchors points
- Min vs. Max Matching
- Flags Modifiers.
-

POLYTECH
NICE-SOPHIA

Polymorphism

Polymorphism [1]

- Run Time Polymorphism:
  - A pointer of a derived class is type-compatible with a pointer of its base class.
  - Polymorphic pointer: pointer of the derived class *disguised* as a pointer of the base class.
  - With run time dispatch to the method of the derived class.

Polymorphism [2.]

```cpp
#include <iostream>
using namespace std;

class Car {
 public:
  virtual ~Car() = default;
  void get_info() const {
    cout << "It's a Car" << '\n';
  }
  virtual void get_price() const = 0;
};

class Dacia : public Car {
 public:
  void get_info() const {
    cout << "It's a Dacia" << '\n';
  }
  void get_price() const override {
    cout << "It's not too expensive" << '\n';
  }
};
```

```cpp
void test1() {

  Car *c = new Dacia()
  c->get_info();

  c->get_price();

  delete c;
}
```

What is the output ?

```
It's a Car
It's not too expensive
```

Polymorphism [2.]

```
1 #include <iostream>
2 using namespace std;
3
4 class Car {
5  public:
6   virtual ~Car() = default;
7   void get_info() const {
8     cout << "It's a Car" << '\n';
9   }
10  virtual void get_price() const = 0;
11 };
12
13 class Dacia : public Car {
14  public:
15   void get_info() const {
16     cout << "It's a Dacia" << '\n';
17   }
18   void get_price() const override {
19     cout << "It's not too expensive" << '\n';
20   }
21 };
```

```
36 void test1() {
37
38   Car *c = new Dacia()
39   c->get_info();
40
41   c->get_price();

42   delete c;
43 }
```

Polymorphic pointer

Compile time resolution

Run time resolution

```
It's a Car
It's not too expensive
```

Polymorphism [3]

```
1 #include <iostream>
2 using namespace std;
3
4 class Car {
5  public:
6   virtual ~Car() = default;
7   void get_info() const {
8     cout << "It's a Car" << '\n';
9   }
10  virtual void get_price() const = 0;
11 };
12
13 class Dacia : public Car {
14  public:
15   void get_info() const {
16     cout << "It's a Dacia" << '\n';
17   }
18   void get_price() const override {
19     cout << "It's not too expensive" << '\n';
20   }
21 };
```

```
23 class Audi : public Car {
24  public:
25   void get_info() const {
26     cout << "It's an Audi" << '\n';
27   }
28   void get_price() const override {
29     cout << "It's expensive" << '\n';
30   }
31 };

48 void test2(const bool selectAudi) {
49
50   Car *c;
51   if (selectAudi) {
52     c = new Audi();
53   } else {
54     c = new Dacia();
55   }
56
57   c->get_info();
58   c->get_price();
59
60   delete c;
61 }
```

Run time selection

Run time dispatch

Polymorphism Improved [1]

```
52 void test3(const bool selectAudi) {
53
54   Car *c = selectAudi ?  new Audi() : new Dacia();
55
56
57   c->get_info();
58   c->get_price();
59
60   delete c;
61 }
```

It is ok?
Compile time error?
Run time error?

```
test.cpp:54: error: conditional expression between
distinct pointer types ‘Audi*’ and ‘Dacia*’ lacks a
cast
   Car *c = selectAudi ? new Audi() : new Dacia();
                                                ^
```

Polymorphism Improved [2]

```
91 int test4(const bool selectAudi) {
92
93   Audi a;
94   Dacia d;
95
96   Car &ra = a;
97   Car &rd = d;
98
99   Car &rc = selectAudi ? ra : rd;
100
101  rc.get_info();
102  rc.get_price();
103
104  return 0;
105
106 }
```

Using Reference

Why can we write ?

```
Car &ra = Audi();
```

```
test.cpp:96: error: invalid initialization of non-const reference of type ‘Car&’ from an rvalue of type ‘Car’
   Car &ra = Audi();
```

Polymorphism Improved [3]

```
//
// polymorphic objects!!!
//
int test6() {

  Car c = Audi();

  c.get_info();
  c.get_price();

  return 0;

}
```

Using Object

Compile time error?
Run time error?

```
Error: cannot allocate an object of abstract type ‘Car’
   Car c = Audi();
because the following virtual functions are pure within
‘Car’: get_price()
```

Polymorphism Improved [3]

```
//
// polymorphic objects!!!
//
int test6() {

  Car c = Audi();

  c.get_info();
  c.get_price();

  return 0;

}
```

Using Object

Assume that Car is a simple base class (not abstract).

Compile OK?
Run time OK? Ouput?

Slicing

```
--------- Test #6
It's a Car
Default price
```

POLYTECH
NICE-SOPHIA

CRTP: Curiously Recurring Template Pattern

CRTP: Static Polymorphism

- Alternative to dynamic dispatch
  - No need for virtual functions, Base class is not abstract.

```
template <class T>
class Base {
 void implementation() {
   ...
   static_cast<T*>(this)->implementation();
 }
};

class Derived : public Base<Derived> {
    void implementation() {
      // code below
      ...
    }
};
```

CRTP: Example (1/2)

```cpp
#include <string>
#include <fstream>
#include <iostream>
#include <ctime>
#include <locale>



class TextoutA {
 public:
  void print(const std::string &str) {
    process(str);
  }

  void print(const std::string &str, const std::string &tag) {
    std::string res = "<" + tag + ">" + str + "</" +  tag +  ">";
    process(res);
  }
 protected:
  virtual void process(const std::string &str) = 0;
};


class Debugout : public TextoutA {
 protected:
  void process(const std::string &str) override {
    std::cout << "DEBUG: string length = " << str.size()
              << " string value = |" << str << "|" << std::endl;
  }
};
```

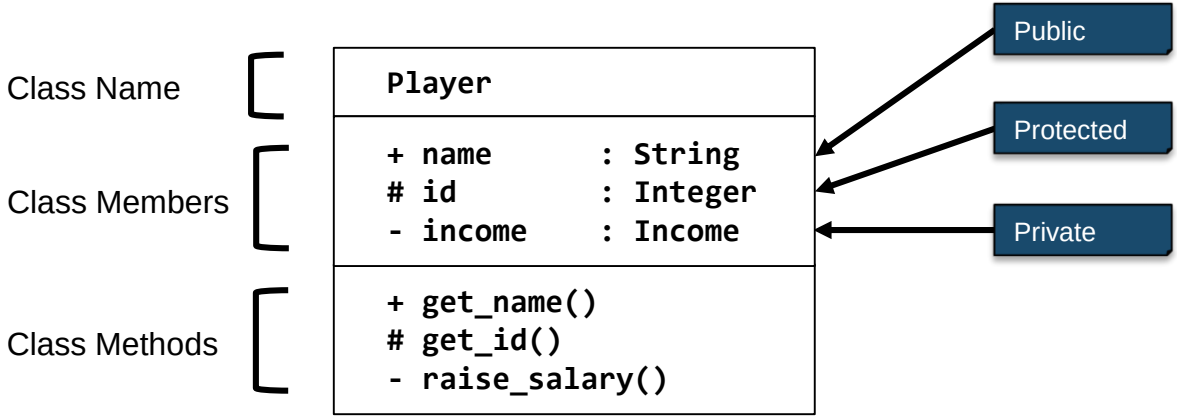
```cpp
#include <string>
#include <fstream>
#include <iostream>
#include <ctime>
#include <locale>


template <typename Derived>
class TextoutA {
 public:
  void print(const std::string &str) {
    static_cast<Derived *>(this)->process(str);
  }

  void print(const std::string &str, const std::string &tag) {
    std::string res = "<" + tag + ">" + str + "</" +  tag +  ">";
    static_cast<Derived *>(this)->process(res);
  }


};


class Debugout : public TextoutA<Debugout> {
 public:
  void process(const std::string &str) {
    std::cout << "DEBUG: string length = " << str.size()
              << " string value = |" << str << "|" << std::endl;
  }
};
```

CRTP: Example (2/2)

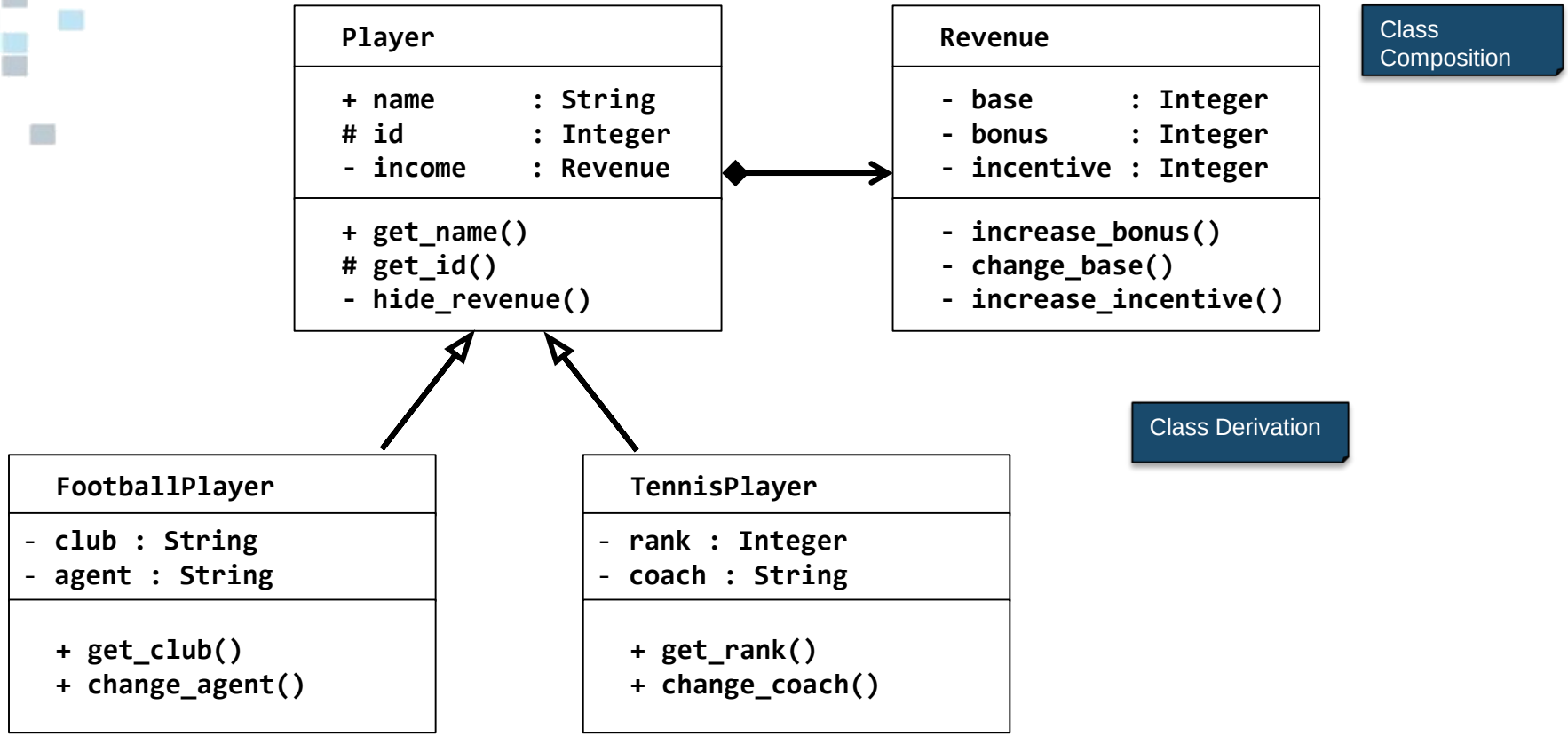
```cpp
class Logout : public TextoutA {
 protected:
  void process(const std::string &str) override {
    std::time_t current_time;
    std::time(&current_time);

    char tbuffer[100];
    std::strftime(tbuffer, 100, "%Y-%m-%d %H:%M:%S", std::localtime(&current_time));
    std::cout << "LOG[" << tbuffer << "] " << str << std::endl;
  }
};



int main() {
  Debugout dout;
  Logout lout;

  dout.print("string printed in debug mode");
  lout.print("string printed in log mode");
  lout.print("Main Title", "h1");

  return 0;
}
```

```cpp
class Logout : public TextoutA<Logout> {
 public:
  void process(const std::string &str) {
    std::time_t current_time;
    std::time(&current_time);

    char tbuffer[100];
    std::strftime(tbuffer, 100, "%Y-%m-%d %H:%M:%S", std::localtime(&current_time));
    std::cout << "LOG[" << tbuffer << "] " << str << std::endl;
  }
};



int main() {
  Debugout dout;
  Logout lout;

  dout.print("string printed in debug mode");
  lout.print("string printed in log mode");
  lout.print("Main Title", "h1");

  return 0;
}
```

UML Overview (1)

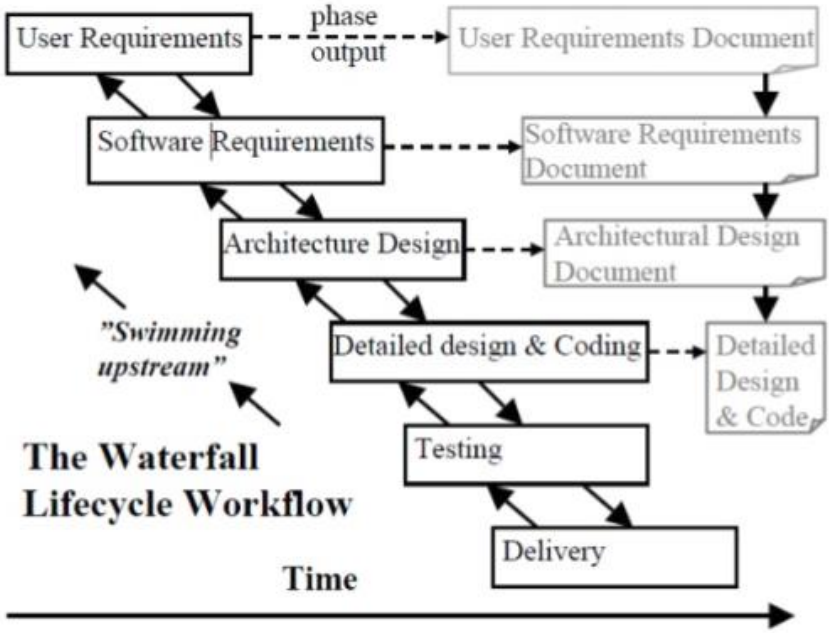


UML Overview (2)



SDLC: Software Dev. Life Cycle [1]

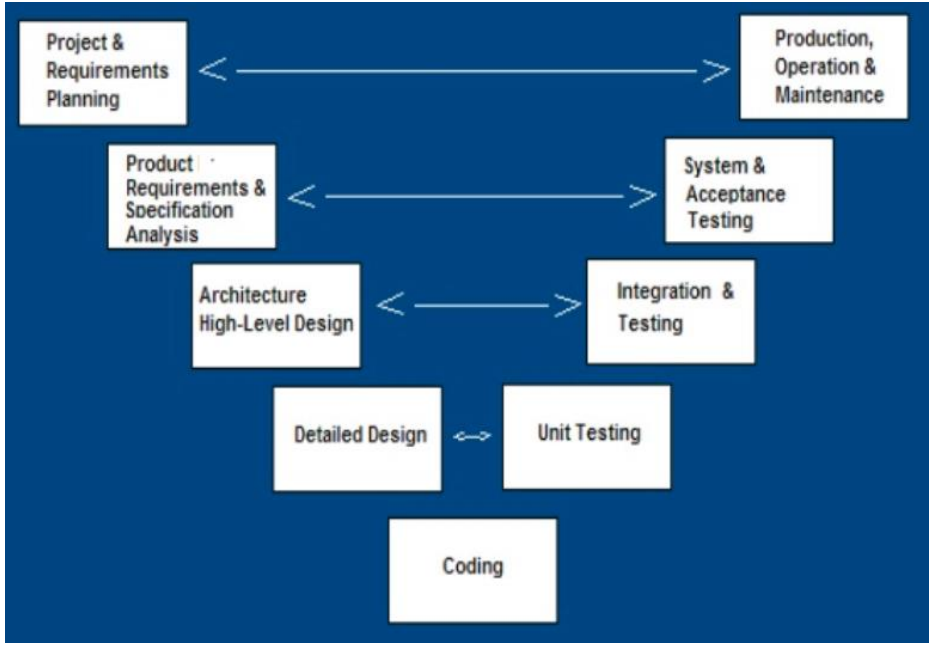
- Traditional Waterfall



https://https://www.slideshare.net/Compare2011/software-development-life-cycle-sdlc

SDLC: Software Dev. Life Cycle [2]

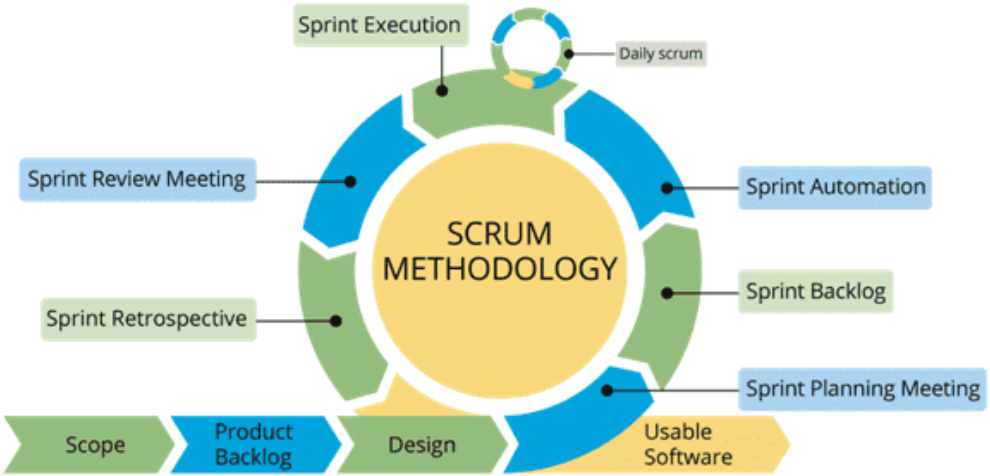
- V-Cycle



https://https://www.slideshare.net/Compare2011/software-development-life-cycle-sdlc

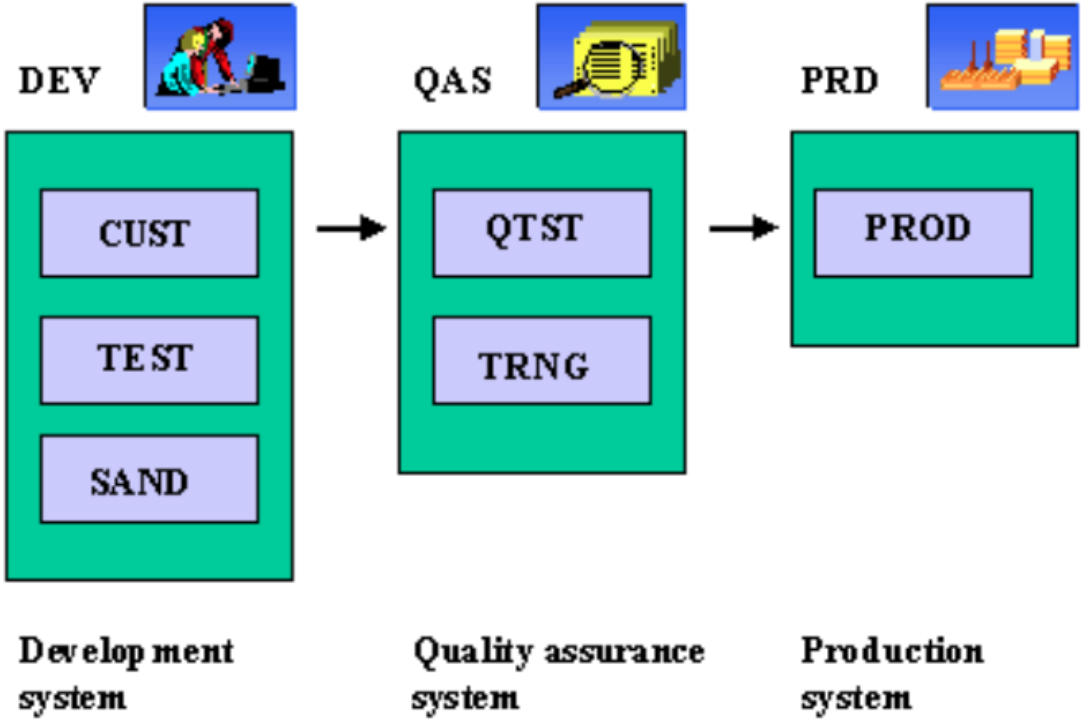
SDLC: Software Dev. Life Cycle [3]

- Agile



https://dzone.com/articles/battle-of-pm-methodologies-waterfall-vs-agile-vs-s

Technical Architecture Landscape



https://help.sap.com/SAPHelp_46C/helpdata/ES/63/a30a4ac00811d2851c0000e8a57770/frameset.htm

Misc. Topics: PostFix or Prefix

- ++*i* or *i*++

```
// postfix increment
int operator ++ (int& n) {
  int tmp = n;
  n = n + 1;
  return tmp;
}
```

```
// prefix increment
int &operator ++ (int& n) {
   n = n + 1;
   return n;
}
```

Misc. Topics: `Using` keyword

- Basic usage: indicate the compiler where to search for the object type

```
using std::string;
...
  string my_string{"Hello"};
```

- Provides alias to complex type

```
using EdgeList = std::vector<int>;

using AdjacencyMatrix =  std::unordered_map<
          GraphNode,
          EdgeList,
          GraphNode::Hash,
          RushState::Compare>;
```

Misc. Topics: West vs. East const

- Pure coding style

```
int const a = 42;  // East const
const int a = 42;  // West const
```

- East const could be more readable

```
int * p;             // p is a mutable pointer to a mutable int
int const * p;       // p is a mutable pointer to a constant int
int * const p;       // p is a constant pointer to a mutable int
int const * const p; // p is a constant pointer to a constant int
```

```
const int * p;       // p is a mutable pointer to a constant int
const int * const p; // p is a constant pointer to a constant int
```