Object-Oriented Programming

API

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

API

Application Programming Interface

- Modern design is the design of APIs and how they interact with each other
- Developers build large programs using existing APIs as much as possible
- For missing functionality, create new APIs
- Final application is integrated into existing APIs
- Large-scale mechanism for *separation of concerns*

Role in Programming Languages [Koenig, Moo]

Language design is library design

Library design is language design

Existing APIs

- Standard library of programming language and platform

  E.g., `std::*`
- External libraries for a programming language

  E.g., `boost::*`
- Custom APIs for specific purposes:

  libxml2 - XML parser and toolkit

  libarchive - Multi-format archive and compression library

  wxWidgets - Toolkit and tools library for creating cross-platform GUIs

  cpp-netlib - C++ network programming library

Good APIs

- Low complexity
- High degree of safety
- Flexible *enough*
- Efficient *enough*

Complexity

- Low *external complexity* often means high *internal complexity*

- Flexibility adds *external complexity*

- Partial solution: Good defaults

- Partial solution: Staged features

Staged Feature: Reading a file into a std::vector

```
// read the file content into a vector
std::vector<char> chars;

std::ifstream input(argv[1]);
input.unsetf(std::ios::skipws);
char c;
while (input.get(c)) {
    chars.push_back(c);
}
```

```
// read the file content into a vector
std::vector<char> chars;
chars.reserve(std::filesystem::file_size(argv[1]));
std::ifstream input(argv[1]);
input.unsetf(std::ios::skipws);
char c;
while (input.get(c)) {
    chars.push_back(c);
}
```

How to Decide

- Carefully name any parts of the API
- Methods are *actions*, and naming should reflect that
- Look at existing code and see if anything is not in the API that should be
- Make sure that the API does not contain concerns for a specific client, e.g., `loc` is an srcML concern
- Imagine separate developers working on the different parts (even if you are doing both parts)
- *Why should they know this concern?*
- *Where would they have learned this concern*?

API Taxonomy

- *Platform*
- *Framework*
- *Toolkit*
- *Library*

Platform

Framework

Toolkit

Library

- Functions/classes typically for a single purpose

- E.g., libxml2, libarchive

- Used with many other libraries and client code

- Control: client (small-scale inversion of control)

Platform

Framework

Toolkit

Library

- Functions/classes for a set of related purposes

- Used with other toolkits and client code

- Control: client (small-scale inversion of control)

Platform

Framework

Toolkit

Library

- Large set of classes for a broad set of purposes

- E.g., MFC, Qt

- Client code is integrated into classes

- Control: Heavy use of *inversion of control*

Platform

Framework

Toolkit

Library

- Entire programming environment, often with a custom language and IDE

- E.g., iOS, .NET, Android, UNIX/Linux

- Client code is integrated into multiple frameworks and supporting tools

- Control: Typically complete *inversion of control*

Design Using APIs

- Use existing APIs as much as possible
- Create new APIs for any functionality that is missing
- Start with *libraries*
- Then move into *toolkits*
- Finally, if significant enough, *framework*
- Minimal chance to create a *platform*

Order of Magnitude

Powers of 10	Exponent Form	Order of Magnitude
1	10^0	0
10	10^1	1
100	10^2	2
1,000	10^3	3
10,000	10^4	4

- Large programs are more complex than small programs by an order of magnitude
- Essential skill of development is to (temporarily) "forget" and concentrate on the current task
- If you do not develop this essential skill, you are limited in what you can develop and the types of jobs you can handle. **This will impact your career**
- The composition and frequency of commits indicate how tasks are isolated