COMS W4156 Advanced Software Engineering (ASE)

September 22, 2022

Agenda

1. Software Reuse
2. APIs

Software Reuse

Don't Reinvent The Wheel !

Our courseworks home page ends with

“Students are strongly encouraged to reuse publicly available open-source or free-for-education software artifacts, in whole or in part, including but not limited to source code, test cases, APIs, libraries, frameworks, services and tools, but all reuse must be clearly documented in submissions for the relevant assignments (e.g., include the download or documentation url). It is not acceptable to resubmit or modify a project developed for another course, reuse proprietary or private software artifacts, e.g., owned by a current or former employer or institution, or to purchase software artifacts to be included in assignments (or the work of someone else writing artifacts on the student's behalf). Otherwise, the course follows the department's standard academic honesty policy. “

Modern software cannot be constructed without reuse, no one constructs non-trivial software from scratch (and even trivial software typically uses standard libraries that ‘come with’ the programming language or platform)

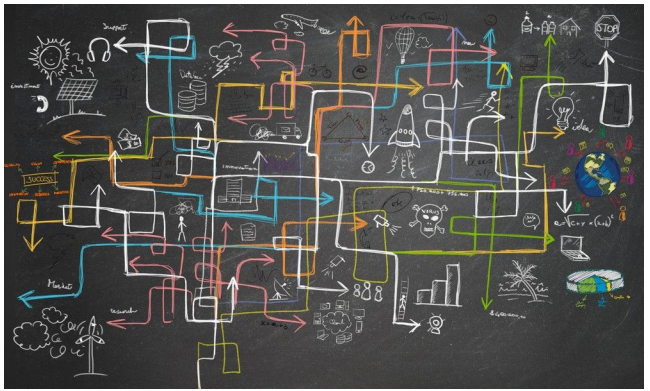
Reusing third-party code (when available) is almost always better, faster, cheaper than developing your application from scratch

Reuse from Your Own Codebase

To reuse previously written code from your own team (same codebase), you can either copy from wherever it is in the codebase and paste into whatever you are working on now, and then modify it to fit (e.g., rename identifiers)

Or you can call it. In most cases it's best to call it (usually the same way you call any other code in your own codebase).

Code from your own organization but not your codebase known as "software assets" and treated much like third-party code



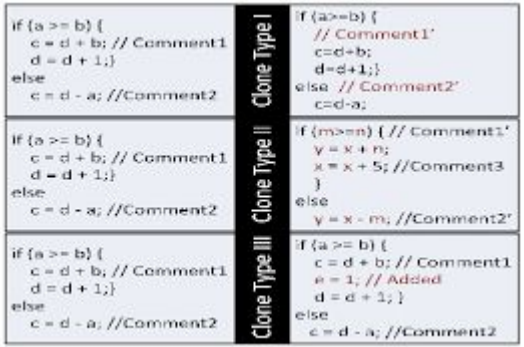
Code Clones

Duplicate or near-duplicate code, known as *code clones*, is considered a “code smell” or “anti-pattern”

A codebase containing code clones has more code to maintain and is thus more expensive to maintain than if there was only one

It is likely that a bug will be fixed in some copies but not in all copies, and the code may diverge over time making it harder to find the remaining copies of the bug

Some static analysis tools (e.g., SonarQube) automatically detect code clones and other code smells (code clones and other code smells discussed later in semester)



Refactoring

Some IDEs (e.g., VS Code) and standalone refactoring tools (e.g., OpenRewrite) automatically detect a group of code clones and help developers replace each instance of the clone with a call to a single code unit

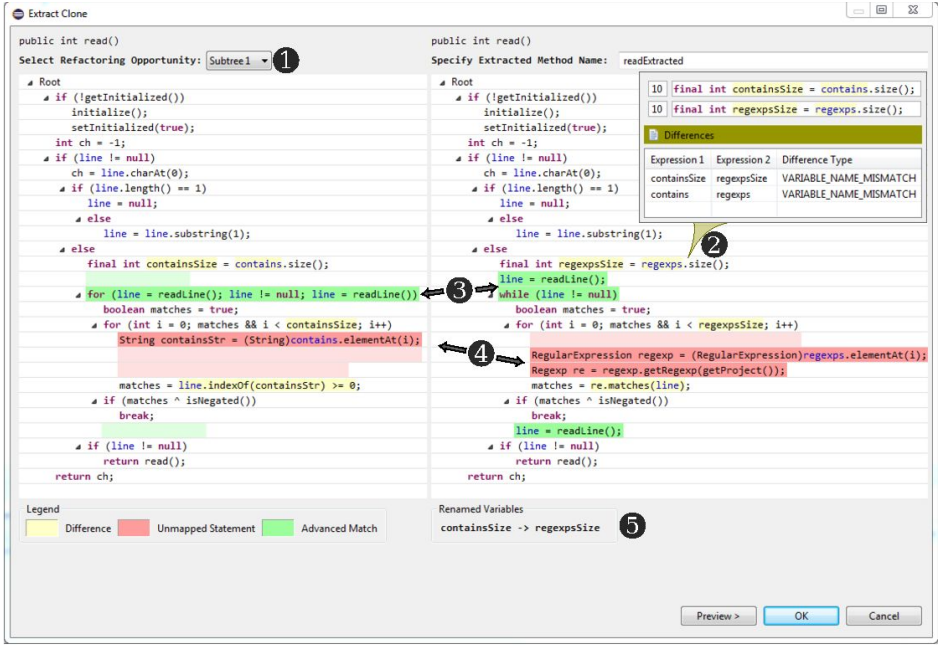


Figure 2: Clone pair visualization and refactorability analysis.

Reuse from Third-Party Codebase

To reuse previously written code from some third party (different codebase), and the source code is open-source, you can copy and paste, and then modify it to fit (e.g., rename identifiers)

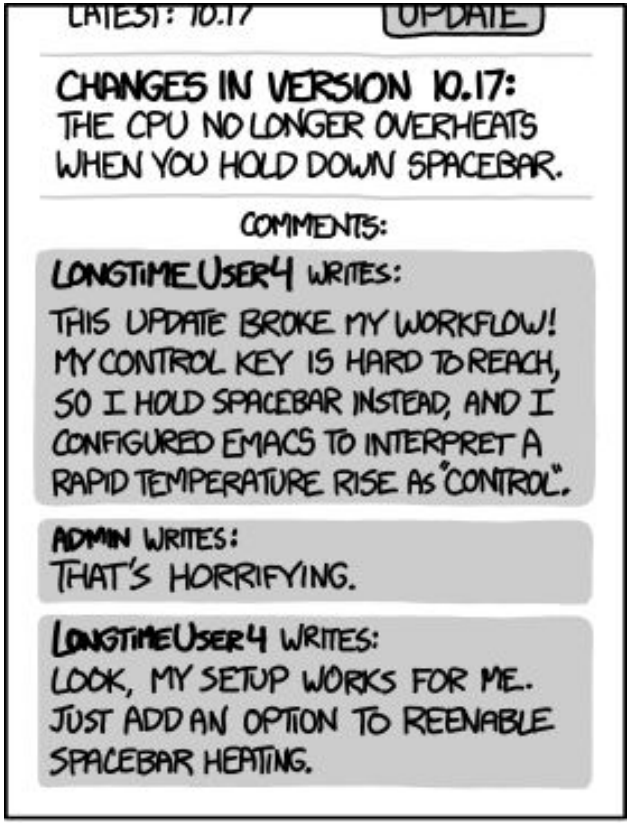
If there's an API available, you can call it

If there's both source code and API available, usually best to call it (using the API). Why?

What's Wrong with Copy/Paste?

Copy/paste from proprietary code, or from open-source code into proprietary code, raises licensing / intellectual property issues (plus you don't get the bug fixes)

But copy/paste may be the only option since not all open-source code is provided in a form intended for independent reuse, e.g., the code you want to reuse may be part of someone else's unrelated application but happens to 'do the right thing' for your application



Agenda

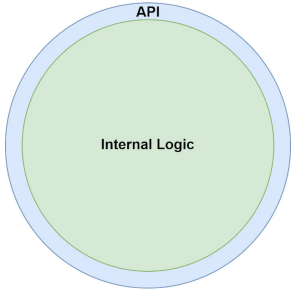
1. Software Reuse
2. APIs

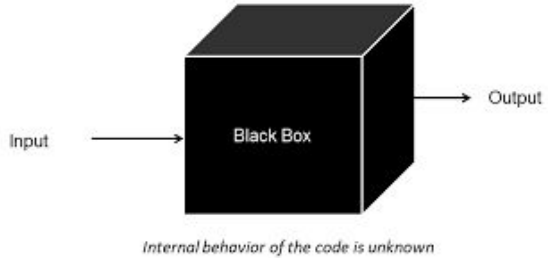
What's an API?

Third-party code intended for reuse has an API = Application Programming Interface

An API is an interface that software developers use to programmatically interact with software components or resources outside of their own code, defines a known range of allowable inputs and associated outputs

When one program is used by another program, we call the first program the provider and the second one the client. The part of the provider accessible to clients is the API





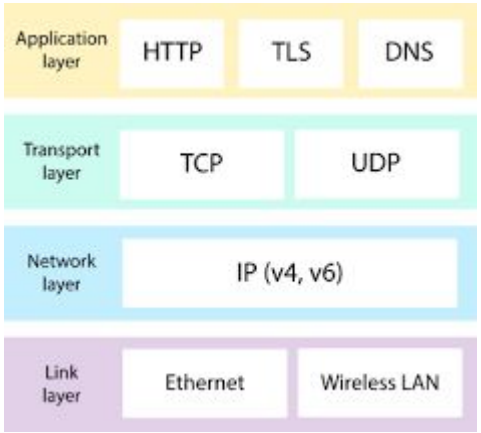
API vs. UI

An interface’s job is to simplify and concentrate internal capabilities into a form that is useful for the client, what is different about APIs and UIs is that they interface with different types of clients

UI is an interface provided to human users

API is an interface provided to code

API vs. Protocol



Protocol - the *rules* for initiating, continuing and completing communications among components and how the communication messages are formatted in terms of bytes (e.g., SSH, HTTP)

API - the set of *function calls* that a given component understands (e.g., Java SE and JDK, Facebook APIs)

We are concerned in this class primarily with APIs, but you will need to use some underlying protocol to send API calls across processes and machines

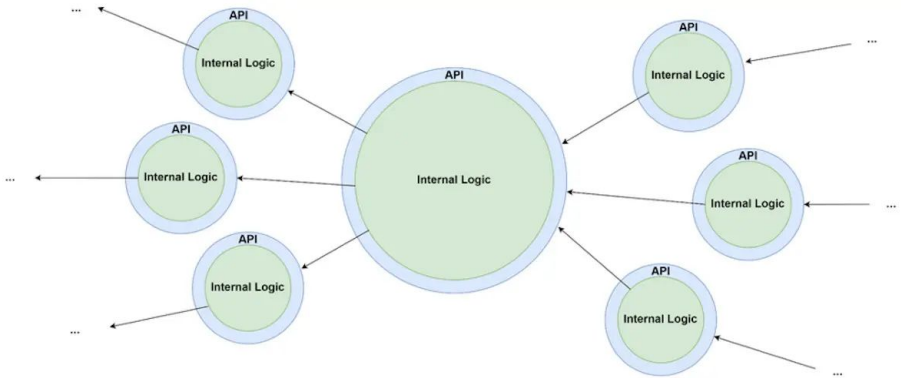
Some APIs are designed to access data, not code, but here we focus on code

Blackbox APIs

Many ‘public’ APIs are blackbox = You can use the interface provided by the box but you cannot peek inside the box

Might be local = runs in same OS process as your executable

Or external = runs somewhere else (possibly same host but often across network) and your code, or an intermediary, needs to know where it is



How Do They Keep Us From Peeking at Local Code?

Distributed only in obfuscated binary or bytecode format

What does this code do?

```
#include <stdio.h>

#define N(a)       "%"#a"$hhn"
#define O(a,b)     "%10$"#a"d"N(b)
#define U          "%10$.*37$d"
#define G(a)       "%"#a"$s"
#define H(a,b)     G(a)G(b)
#define T(a)       a a
#define s(a)       T(a)T(a)
#define A(a)       s(a)T(a)a
#define n(a)       A(a)a
#define D(a)       n(a)A(a)
#define C(a)       D(a)a
#define R          C(C(N(12)G(12)))
#define o(a,b,c)   C(H(a,a))D(G(a))C(H(b,b)G(b))n(G(b))O(32,c)R
#define SS         O(78,55)R "\n\033[2J\n%26$s";
#define E(a,b,c,d) H(a,b)G(c)O(253,11)R G(11)O(255,11)R H(11,d)N(d)O(253,35)R
#define S(a,b)     O(254,11)H(a,b)N(68)R G(68)O(255,68)N(12)H(12,68)G(67)N(67)

char* fmt = O(10,39)N(40)N(41)N(42)N(43)N(66)N(69)N(24)O(22,65)O(5,70)O(8,44)N(
            45)N(46)N    (47)N(48)N(    49)N( 50)N(     51)N(52)N(53    )O( 28,
            54)O(5,        55) O(2,    56)O(3,57)O(      4,58 )O(13,    73)O(4,
            71 )N(   72)O   (20,59    )N(60)N(61)N(       62)N (63)N    (64)R R
            E(1,2,   3,13   )E(4,    5,6,13)E(7,8,9        ,13)E(1,4    ,7,13)E
            (2,5,8,        13)E(    3,6,9,13)E(1,5,         9,13)E(3    ,5,7,13
            )E(14,15,    16,23)    E(17,18,19,23)E(          20, 21,    22,23)E
            (14,17,20,23)E(15,    18,21,23)E(16,19,    22     ,23)E(    14, 18,
            22,23)E(16,18,20,    23)R U O(255 ,38)R    G (     38)O(    255,36)
            R H(13,23)O(255,    11)R H(11,36) O(254    ,36)     R G(    36 ) O(
            255,36)R S(1,14    )S(2,15)S(3, 16)S(4,    17 )S     (5,    18)S(6,
            19)S(7,20)S(8,    21)S(9    ,22)H(13,23    )H(36,    67    )N(11)R
            G(11)""O(255,    25 )R        s(C(G(11)    ))n (G(          11) )G(
            11)N(54)R C(    "aa")   s(A(   G(25)))T    (G(25))N         (69)R o
            (14,1,26)o(    15, 2,   27)o   (16,3,28    )o( 17,4,        29)o(18
            ,5,30)o(19    ,6,31)o(        20,7,32)o    (21,8,33)o       (22 ,9,
            34)n(C(U)    )N( 68)R H(    36,13)G(23)    N(11)R C(D(      G(11)))
            D(G(11))G(68)N(68)R G(68)O(49,35)R H(13,23)G(67)N(11)R C(H(11,11)G(
            11))A(G(11))C(H(36,36)G(36))s(G(36))O(32,58)R C(D(G(36)))A(G(36))SS

#define arg d+6,d+8,d+10,d+12,d+14,d+16,d+18,d+20,d+22,0,d+46,d+52,d+48,d+24,d\
            +26,d+28,d+30,d+32,d+34,d+36,d+38,d+40,d+50,(scanf(d+126,d+4),d+(6\
            -2)+18*(1-d[2]%2)+d[4]*2),d,d+66,d+68,d+70, d+78,d+80,d+82,d+90,d+\
            92,d+94,d+97,d+54,d[2],d+2,d+71,d+77,d+83,d+89,d+95,d+72,d+73,d+74\
            ,d+75,d+76,d+84,d+85,d+86,d+87,d+88,d+100,d+101,d+96,d+102,d+99,d+\
            67,d+69,d+79,d+81,d+91,d+93,d+98,d+103,d+58,d+60,d+98,d+126,d+127,\
            d+128,d+129

char d[538] = {1,0,10,0,10};

int main() {
    while(*d) printf(fmt, arg);
}
```

API Communication (Meta-)Patterns

Library = shared code that you compile or bundle into your application, usually runs in same process so might or might not be considered a separate architectural component

Service = a shared capability that you access from your application, usually runs in another process, often on another machine

Frameworks - shared code, and sometimes shared capabilities, for building applications. Frameworks often include libraries and may use services.

Libraries

Most programming languages have “standard libraries” that come with the language or the platform (e.g., JVM, Linux)

- Typically includes I/O, string processing, math, “system calls”
- In most cases, just call the function like any other call - and remember to check status codes (when applicable)
- If you’ve written a program that does pretty much anything, you’ve already used some standard libraries

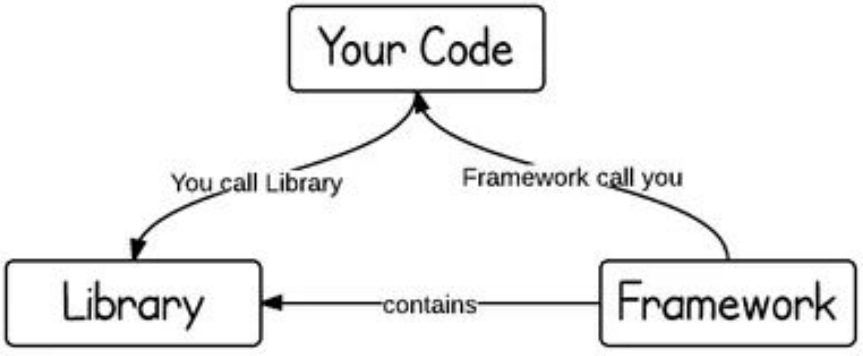
Many other libraries also implement reusable functionality

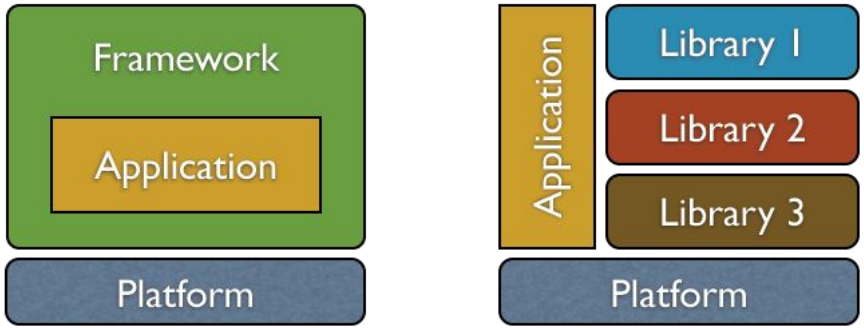
Download manually or via a build tool / package manager ecosystem (e.g., Maven Central) to bundle with your program

	Library	Service
Characteristics	Executes locally inside of your program	Executes remotely outside of your program
	Distributed/deployed with your program	Exists independently of your program
	Internal compilation dependency for your program	External consumption dependency for your program
	Cannot change unless you recompile/redeploy your program	Versioned independently of your program
	Instance used in your program is used only by your program	Can be shared with many remote programs
	Access to library routines cannot be mediated	Access can be mediated for security, QoS, or other reasons
	Access to library routines cannot be audited	Access can be logged and audited by a mediator

	Library	Service
Strengths	Program cannot be inadvertently broken by a dependency change	Many programs can benefit from service reuse and improvement
	Programmers can easily understand all project dependencies	Service mediation can provide many benefits
	Compiled program is entirely self-contained	Service-based design is more flexible to unanticipated changes
Weaknesses	Library improvements can only be used at program recompilation	Service oversubscription creates an availability bottleneck
	Multiple versions of the same library can be used in many programs	Service mediation is a black art to most programmers
	Patterns of library consumption are not clearly visible at runtime	Services may not be modeled with the right level of granularity for reuse

Library vs. Framework





"*Inversion of control*"
(Hollywood Principle = don't call us, we'll call you)

- When your code calls a library (or a service), your code is in control.
- In contrast, the framework calls your code, so the framework is in control. The framework plays the role of the main program in coordinating and sequencing application activity and you fill in the blanks.

Frameworks

Set of generic components (some of which may be libraries or services) that work together to accomplish common development tasks and functionalities for a popular target domain, such as web or mobile apps

Most frameworks define an application skeleton with places to put your code for the framework to call

The context of those places (e.g., callbacks or event handlers) comprises the framework's API

Most web apps are built on some MVC-oriented framework (e.g., spring)

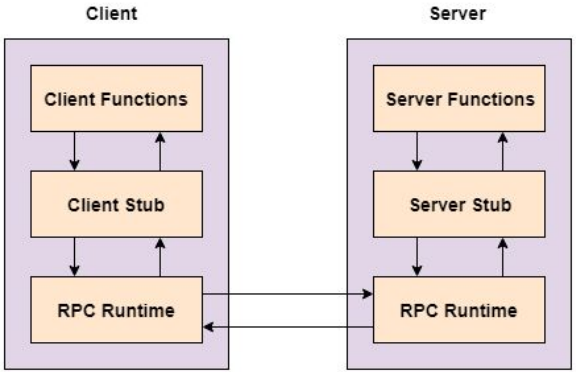
Services

Some services are very specific to a particular task or domain (e.g., https://ipinfo.io/ maps an IP address to its geographic location, carrier, whether using VPN, etc.)

Some services interface to a particular application or ecosystem (e.g., Twitter API)

Lots of public services listed here and here

Services might be called via what appear to be regular function calls (*RPC* = remote procedure call) or invoked via publish/subscribe (events), or [now very popular] REST = Representational State Transfer



Next Class

RESTful APIs

RESTful demo

Upcoming Assignments

Preliminary Project Proposal: due September 26, next Monday - Tentative mentors will be assigned after we read your preliminary proposals

Revised Project Proposal: due October 3 - meet with your Mentor to discuss your preliminary proposal *before* submitting revision

Ask Me Anything