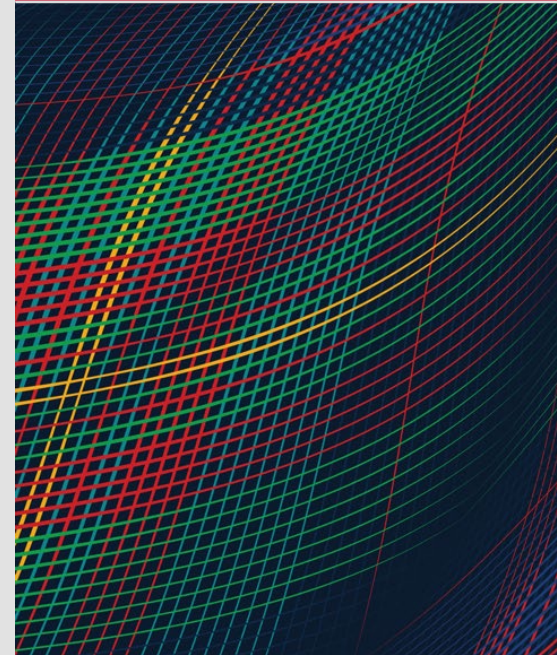


Contract Programming: Formalizing APIs

JULY 9, 2024

Alex Vesey
Software Engineer



Document Markings

Carnegie Mellon University 2024

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

References herein to any specific entity, product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute nor of Carnegie Mellon University - Software Engineering Institute by any such named or represented entity.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Please see Copyright notice for non-US Government use and distribution.

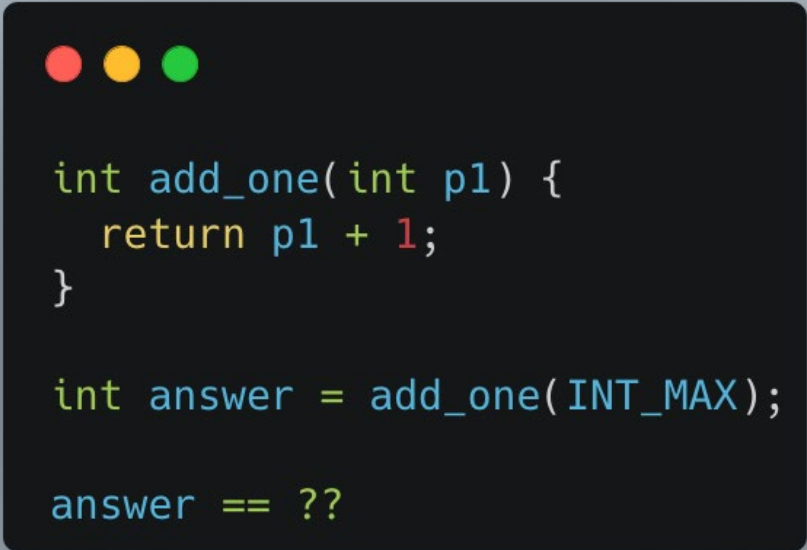
This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Agenda

- Example
- APIs and Contracts
- Contract Programming
- Implications for Security
- Getting Started
- How to make code ~~more boring~~
less surprising

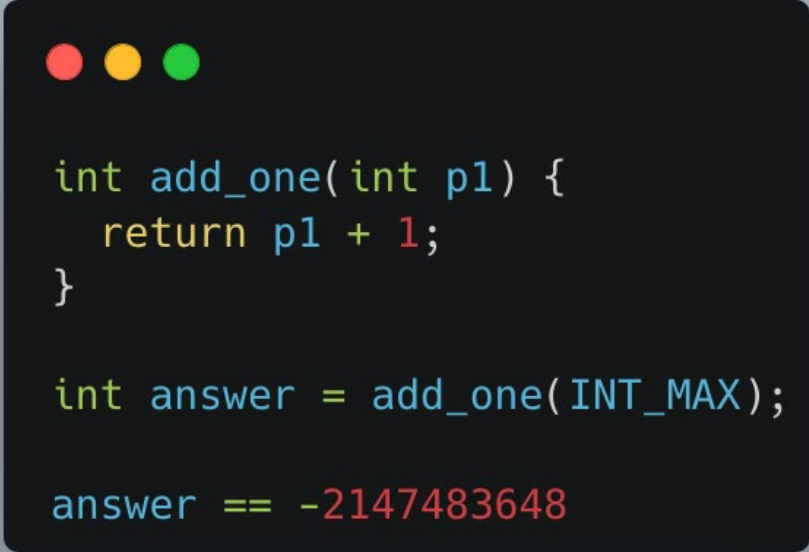


Examples of functions with ill-specified contracts



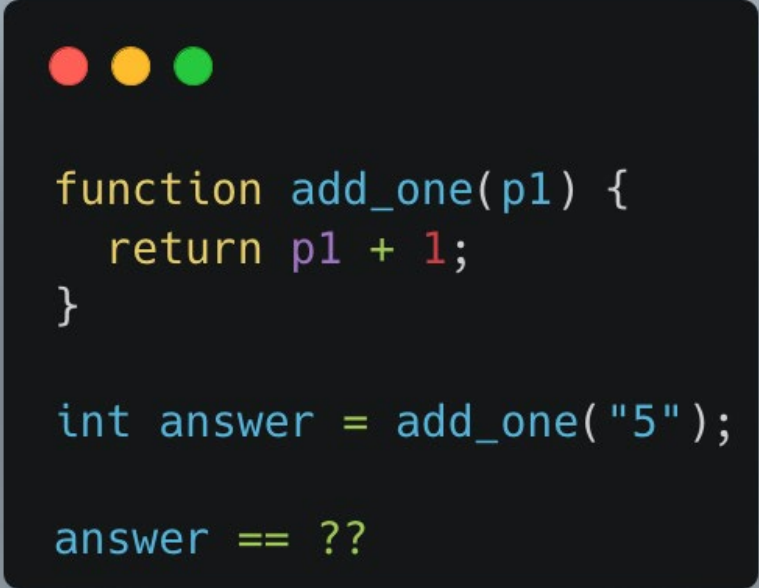
```
int add_one(int p1) {  
    return p1 + 1;  
}  
  
int answer = add_one(INT_MAX);  
  
answer == ??
```

Examples of functions with ill-specified contracts



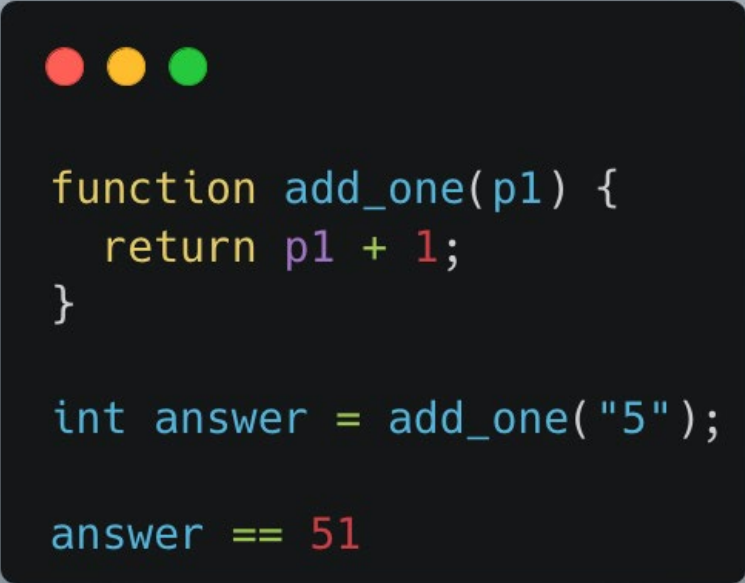
```
int add_one(int p1) {  
    return p1 + 1;  
}  
  
int answer = add_one(INT_MAX);  
  
answer == -2147483648
```

Examples of functions with ill-specified contracts



```
function add_one(p1) {  
  return p1 + 1;  
}  
  
int answer = add_one("5");  
  
answer == ??
```

Examples of functions with ill-specified contracts



```
function add_one(p1) {  
  return p1 + 1;  
}  
  
int answer = add_one("5");  
  
answer == 51
```

Application Programming Interface (API)

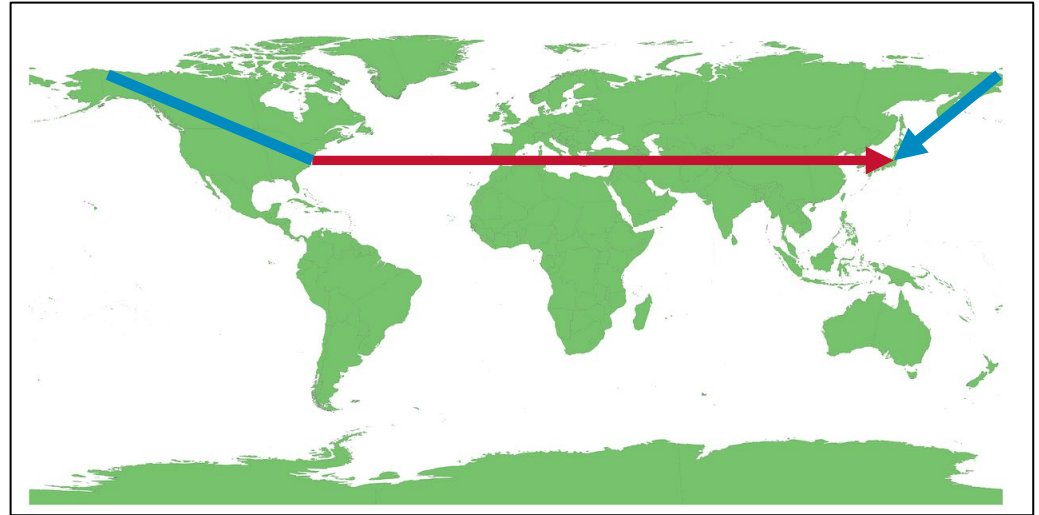
“A set of functionalities independent of their implementation, allowing the implementation to vary without compromising the users of the component.” — Joshua Bloch

A way to decouple what code can do for a user from how the code does it.

Mental Models

With abstraction comes a “mental” model and ambiguity

“All models are wrong” – George Box



Contracts in Computer Science

“A contract is a formal interface specification for a software component such as a function or a class” – C++ SG21
(Contracts) Document P2900R3

When a piece of code executes it will change the state of the computer system.

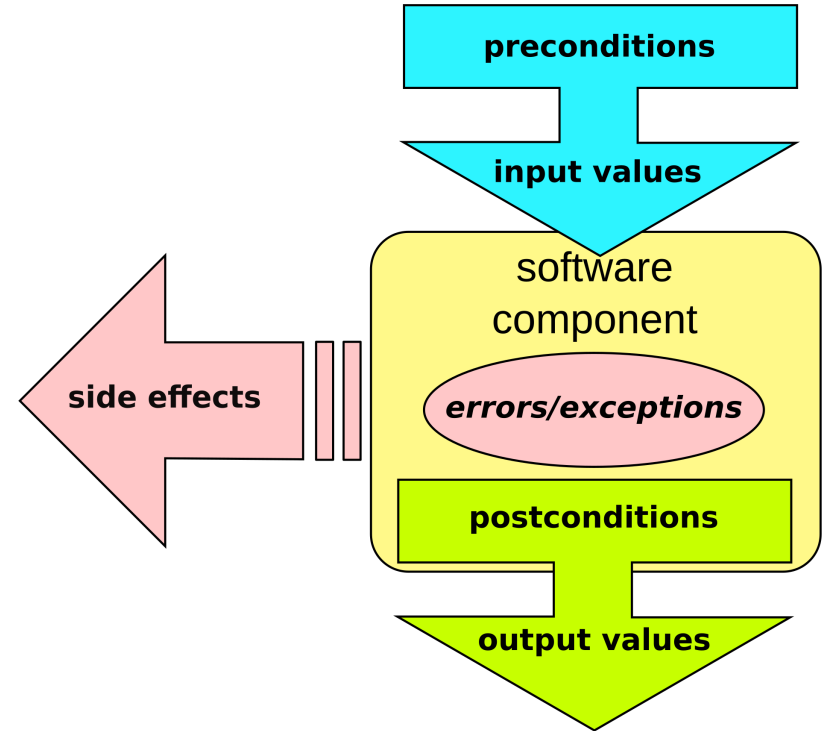
For a precondition P , postcondition Q and command C it can be said that if P is true and C is executed then Q will be true (if C terminates).

$$\{ x + 1 = 42 \}$$
$$y = x + 1$$
$$\{ y = 42 \}$$

Contracts: Conditions and Invariants

Contracts typically consist of:

- A set of preconditions that must be true
- A set of post conditions the function guarantees will be true
- A set of invariants that the function will not change



Narrow and Wide Contracts

“Wide” contracts have no pre-conditions

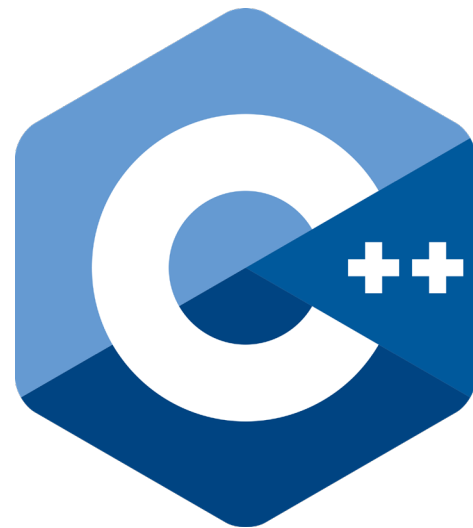
“Narrow” contracts have (sometimes many) preconditions

Example: **`std::vector::size()`**

- Wide
- No-throw guarantee

Example: **`std::vector::at()`**

- Narrow
- May throw and exception



Contract Programming and Security

Research¹²³ has shown that bad APIs are bad for security

- Incorrect usage of security functions
- Assumed or implied preconditions or post conditions
- Invariants that are not

Don't trust that developers will read your documentation

- Your documentation may not serve their learning style⁴
- They might read it, but are likely to go to stack overflow too⁵

1 <https://dl.acm.org/doi/10.1145/2508859.2516655>

2 <https://www.usenix.org/system/files/sec20-tang.pdf>

3 <https://dl.acm.org/doi/10.1145/2896587>

4 <https://journals.sagepub.com/doi/10.1177/0047281617721853>

5 <https://survey.stackoverflow.co/2023/#learning-to-code-learn-code>

APIs and CWEs

There are many Common Weakness Enumerations pertaining to APIs

Implementing:

- CWE 1059 – “The product does not contain sufficient technical or engineering documentation”
- CWE 628 – “The product calls a function, procedure, or routine with arguments that are not correctly specified”

Consuming:

- CWE 648 – “The product does not conform to the API requirements for a function call that requires extra privileges”
- CWE 475 – “The behavior of this function is undefined unless its control parameter is set to a specific value”

Security Example: Heartbleed

```
int dtls1_process_heartbeat(SSL* s) {
    unsigned char *p = &s->s3->rrec.data[0], *pl;
    unsigned int payload; unsigned int padding = 16;
    hbtype = *p++;
    n2s(p, payload);
    pl = p;

    ...

    if (hbtype == TLS1_HB_REQUEST) {
        unsigned char *buffer, *bp; int r;
        buffer = OPENSSL_malloc(1 + 2 + payload + padding);
        bp = buffer;
        s2n(payload, bp);
        memcpy(bp, pl, payload);
        bp += payload;
        RAND_pseudo_bytes(bp, padding);
        OPENSSL_free(buffer);
        r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding); }
}
```

```
struct {
    HeartbeatMessageType type; // 1 byte: request or the response
    uint16 payload_length; // 2 byte: the length of the payload
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;
```

Assumptions: (1) pl is valid and points to the beginning of the payload, (2) the size of the payload should be the same as the value in the payload length field

Security Example: Heartbleed

Would a contract have solved the issue? Maybe



```
// HeartbeatMessage is valid if payload_length and size of
// payload are the same
struct {
...
}
// Precondition: &s->s3->rrec.data[0] is valid
int dtls1_process_heartbeat(SSL* s) {
// TODO: check if payload + padding is the same as s->s3->rrec.length
// to satisfy the precondition
}
```


Getting Started

Existing codebases

- Start with comments about assumptions, preconditions and post conditions
- Enforcement through gradual typing if the language allows

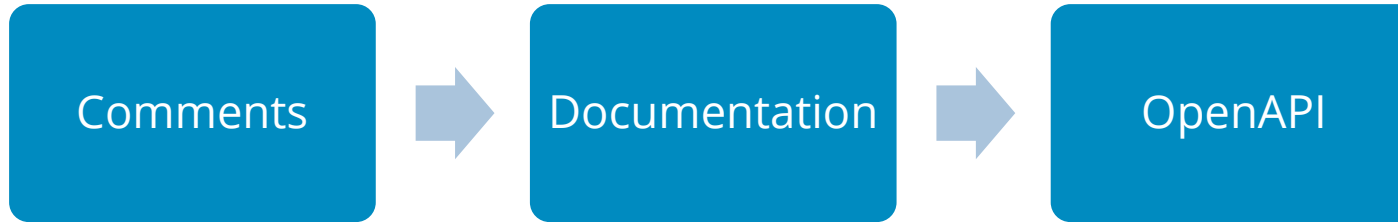
Green field projects

- OpenAPI with definitions of parameters, responses and schemas
- Libraries like Guava (Java) and Preconditions in the Kotlin stdlib

Coming soon:

- Language support for contracts is expected in C++26 (maybe)

Levels of Contract Formality



```

// Please only use num < INT_MAX
int add_one(int num) {
    return num + 1;
}
  
```

get_sync_config(pythonify=False)

[\[source\]](#)

Get the sync server config. WARNING: This method only works if the user calling it is a sync user

Parameters: **pythonify** (bool) – Returns a PyMISP Object instead of the plain json output

Return type: dict[str, Any] | MISPServer

Add event

AUTHORIZATIONS: [ApiKeyAuth](#)

REQUEST BODY SCHEMA: application/json

org_id: string (OrganisationId) <= 10 characters ^d+\$

distribution: string (DistributionLevelId)
Enum: ["0", "1", "2", "3", "4", "5"]
Who will be able to see this event once it becomes published and eventually when it becomes pulled:

- 0 - Your organization only
- 1 - This community only
- 2 - Connected communities
- 3 - All communities
- 4 - Sharing group
- 5 - Inherit Event

info: string (EventInfo) <= 65535 characters

orgc_id: string (OrganisationId) <= 10 characters ^d+\$

uuid: string <uuid> (UUID) <= 36 characters

date: string

published: boolean (PublishedFlag)
Default: false

analysis: string (AnalysisLevelId)
Enum: ["0", "1", "2"]
Represents the analysis maturity level.

- 0 - Initial
- 1 - Ongoing
- 2 - Complete

Request samples

POST /events/add

Payload

Content type: application/json

```

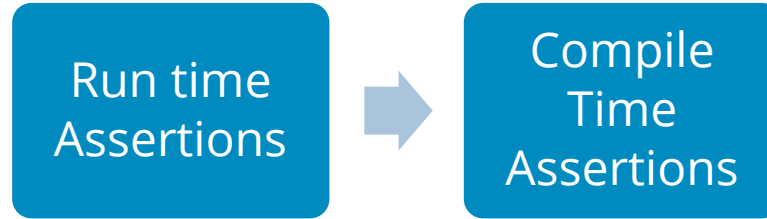
{
  "org_id": "12345",
  "distribution": "0",
  "info": "logged source ip",
  "orgc_id": "12345",
  "uuid": "c99506a6-1255-4b71-afa5-7b8b048c3b1b",
  "date": "1991-01-15",
  "published": false,
  "analysis": "0",
  "attribute_count": "321",
  "timestamp": "1617875568",
  "sharing_group_id": "1",
  "proposal_email_lock": true,
  "locked": true,
  "threat_level_id": "1",
  "publish_timestamp": "1617875568",
  "sighting_timestamp": "1617875568",
  "disable_correlation": false,
  "extends_uuid": "c99506a6-1255-4b71-afa5-7b8b048c3b1b",
  "event_creator_email": "user@example.com"
}
  
```

Response samples

200 403 default

Content type

Levels of Contract Formality



```
int add_one(int p1) {  
    assert(p1 < INT_MAX)  
    return p1 + 1;  
}  
// Error!  
int answer = add_one(INT_MAX);
```

```
void odd_func(){  
    static_assert(sizeof(long) == 8, "This code relies on 'long' being exactly 8 bytes");  
}
```

Closing Thoughts

- APIs are useful and pervasive.
- Contracts help build better APIs
- Poorly defined APIs are a security risk
- The formality of an API Contract is flexible