

Designing Robust APIs

How to Write C++ Code that's Safe, Extensible, Efficient & Easy to Use



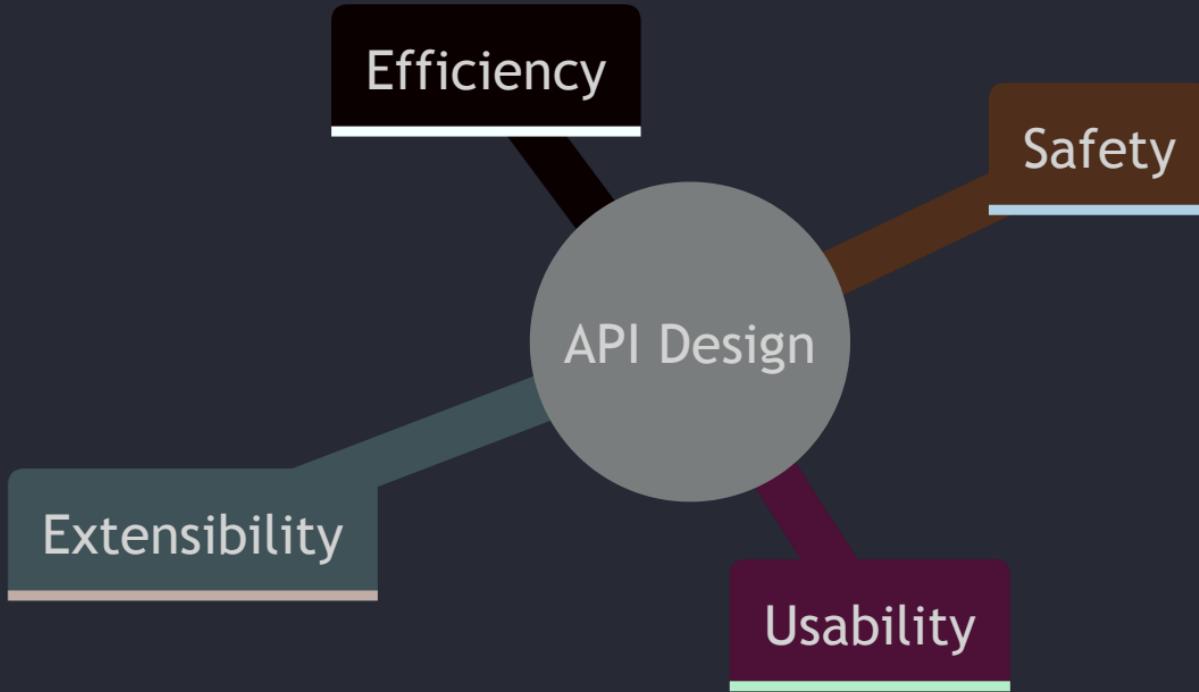
Обо мне

- Пишу на C++ больше 15 лет.
- Основал WG21 Russia в 2016 вместе с [@apolukhin.](#)
- В 2016-2019 представлял предложения от РФ в комитете.
- Руководил разработкой поискового движка в Яндексе.
- Руководил инфраструктурой, поиском и ML в Озоне.

Для кого этот доклад?

- Для тех, кто пишет библиотечный код.
- Для тех, чей код так или иначе будет долго жить или широко использоваться.
- Для тех, кто хочет писать код, которым приятно пользоваться.

```
1  class Buffer {  
2  public:  
3      Buffer(const char *data, size_t size);  
4  
5      Buffer(const Buffer &buffer)  
6          : Data_(nullptr)  
7          , Len_(0)  
8          , Pos_(0)  
9      {  
10         *this = buffer;  
11     }  
12  
13     // ...  
14 }
```



Хороший API находит баланс между всеми этими аспектами.

Почему это важно?

The Wonderfully Terrible World of C and C++ Encoding APIs:

Feature Set 🖐️ vs. Library 👉

	boost.text	utf8cpp	Standard C	Standard C++	Windows API
Handles Legacy Encodings	✗	✗	😊	😊	✓
Handles UTF Encodings	✓	✓	😊	😊	✓
Bounded and Safe Conversion API	✗	✗	😉	✓	✓
Assumed Valid Conversion API	✓	✓	✗	✗	✗
Unbounded Conversion API	✓	✓	✗	✗	✓
Counting API	✗	😉	✗	✗	✓
Validation API	✗	😉	✗	✗	✗
Bulk Conversions	✓	✓	😊	😊	✓
Single Conversions	✓	✓	✓	✓	✗
Custom Error Handling	✗	✓	✓	✓	✗
Updates Input Range (How Much Read™)	✓	✗	✓	✓	✗
Updates Output Range (How Much Written™)	✓	✓	✓	✓	✗

- *Standard C: it's trash.*
- *Standard C++: provides next-to-nothing of its own that is not sourced from C, and when it does it somehow makes it worse. Also trash.*

Почему это важно?

Exploiting aCropalypse: Recovering Truncated PNGs:

SimonTime — 2023-01-02 15:28

so basically the pixel 7 pro, when you crop and save a screenshot, overwrites the image with the new version, but leaves the rest of the original file in its place

Retr0id — 2023-01-02 15:28

ohhhhhh wow

SimonTime — 2023-01-02 15:28

so if you were to take a screenshot of an app which shows your address on screen, then crop it, if you could recover the information somehow that's a big deal

...

IMHO, the takeaway here is that API footguns should be treated as security vulnerabilities.

См. Back to Basics: C++ API Design - CppCon 2022 by Jason Turner .

Rule #1:

Проектируйте API так, чтобы его нельзя было использовать неправильно

- Программа должна или работать корректно, или завершаться с ошибкой.
- Не должно существовать последовательности вызовов, которая приводит вашу программу в некорректное состояние.
- Чем меньше у вашего API способов завершиться с ошибкой — тем лучше. Зачем обрабатывать ошибки, если можно спроектировать API, в котором их нет?

```
1 struct CsvStats {
2     DateTime startTime;
3     DateTime endTime;
4     // ...
5 };
6
7 class CsvDb {
8 public:
9     explicit CsvDb(std::string_view path);
10
11    std::string path(std::string_view tableName);
12
13    CsvStats stats(std::string_view tableName);
14    void setStats(std::string_view tableName, CsvStats stats);
15};
```

```
1 struct CsvStats {
2     DateTime startTime;
3     DateTime endTime;    // Some per-table aggregates: start & end time,
4     // ...                // maybe per-column min, max & mean values.
5 };
6
7 class CsvDb {
8 public:
9     explicit CsvDb(std::string_view path);
10
11    std::string path(std::string_view tableName);
12
13    CsvStats stats(std::string_view tableName);
14    void setStats(std::string_view tableName, CsvStats stats);
15};
```



```
1 struct CsvStats {
2     DateTime startTime;
3     DateTime endTime;
4     // ...
5 };
6
7 class CsvDb {
8 public:
9     explicit CsvDb(std::string_view path);
10
11    std::string path(std::string_view tableName);
12
13    CsvStats stats(std::string_view tableName);           // Getter & setter for stats.
14    void setStats(std::string_view tableName, CsvStats stats); // Stats are stored in db.info,
15};                                         // and flushed on setStats.
```



```
1 struct CsvStats {
2     DateTime startTime;
3     DateTime endTime;
4     // ...
5 };
6
7 class CsvDb {
8 public:
9     explicit CsvDb(std::string_view path);
10
11    std::string path(std::string_view tableName);
12
13    CsvStats stats(std::string_view tableName);
14    void replace(std::string_view tableName, std::string_view newPath,
15                 CsvStats newStats); // Safer!
16};
```

```
1 struct CsvStats {
2     DateTime startTime;
3     DateTime endTime;
4     // ...
5 };
6
7 class CsvDb {
8 public:
9     explicit CsvDb(std::string_view path);
10
11     CsvReader open(std::string_view tableName);
12
13     CsvStats stats(std::string_view tableName);
14     CsvWriter replace(std::string_view tableName); // WAY safer!
15 }
```


VS

```
1 struct CsvStats {
2     DateTime startTime;
3     DateTime endTime;
4     // ...
5 };
6
7 class CsvDb {
8 public:
9     explicit CsvDb(std::string_view path);
10
11    std::string path(std::string_view tableName);
12
13    CsvStats stats(std::string_view tableName);
14    void setStats(std::string_view tableName,
15                  CsvStats stats);
16 }
```

```
1 struct CsvStats {
2     DateTime startTime;
3     DateTime endTime;
4     // ...
5 };
6
7 class CsvDb {
8 public:
9     explicit CsvDb(std::filesystem::path path);
10
11    CsvReader open(std::string_view tableName);
12    CsvWriter replace(std::string_view tableName);
13
14    CsvStats stats(std::string_view tableName);
15
16 }
```

```
1 class Window {
2 public:
3     void setTitle(const std::string &title);
4     std::string title() const;
5
6     void resize(const Size &size);
7     Size size() const;
8
9     void setPosition(const Point &point);
10    Point position();
11
12    // ...
13
14    void initOpenGL();
15    void bindContext();
16    void swapBuffers();
17 }
```

```
1 class Window {
2 public:
3     void setTitle(const std::string &title);      //
4     std::string title() const;                      //
5                                         //
6     void resize(const Size &size);                 // Window properties ...
7     Size size() const;                            //
8                                         //
9     void setPosition(const Point &point);        //
10    Point position();                           //
11
12    // ...
13
14    void initOpenGL();
15    void bindContext();
16    void swapBuffers();
17 }
```



```
1 class Window {
2 public:
3     void setTitle(const std::string &title);
4     std::string title() const;
5
6     void resize(const Size &size);
7     Size size() const;
8
9     void setPosition(const Point &point);
10    Point position();
11
12    // ...
13
14    void initOpenGL();
15    void bindContext(); // OOPS: assert(_glInitialized); inside.
16    void swapBuffers(); // Can only be called after a call to initOpenGL().
17 }
```

```
1 class Window {
2 public:
3     void setTitle(const std::string &title);
4     std::string title() const;
5
6     void resize(const Size &size);
7     Size size() const;
8
9     void setPosition(const Point &point);
10    Point position();
11
12    // ...
13
14    std::unique_ptr<OpenGLContext> createOpenGLContext();
15};
16
17 class OpenGLContext {
18 public:
19     void bind();
20     void swapBuffers();
21
22     // ...
23 }
```

Rule #2:

Divide & Conquer: Дробите!

- Помните про S in SOLID.

"The Single-responsibility principle: There should never be more than one reason for a class to change."

- Количество возможных взаимодействий (и багов) внутри класса растет как квадрат от размера класса. Меньше классы — меньше проблем!
- Мелкие классы легче читать и осознавать.
- Аналогично работает и с библиотеками. Классы в "помойке классов" начинают зависеть друг от друга, снова квадрат зависимостей и баги. Дробите на более мелкие библиотеки!
- Эта же логика применима к функциям.

```
1 void parseStrings(const Buffer &buffer, std::vector<std::string> *result) { // Parses a buffer of null-terminated
2     size_t pos = 0; // strings.
3     while (pos < buffer.size()) {
4         const char *nextPos = static_cast<const char *>(memchr(buffer.data() + pos, '\0', buffer.size() - pos));
5         size_t size = (nextPos ? nextPos - buffer.data() : buffer.size()) - pos;
6         std::string str = std::string(buffer.data() + pos, size);
7
8         if (str.size() ≥ 2 && str.front() == '\"' && str.back() == '\"')
9             str = str.substr(1, str.size() - 2);
10
11        result->push_back(std::move(str));
12        pos += size;
13    }
14 }
```

```
1 void parseStrings(const Buffer &buffer, std::vector<std::string> *result) {
2     size_t pos = 0;
3     while (pos < buffer.size()) { // Find next null terminator ↓
4         const char *nextPos = static_cast<const char *>(memchr(buffer.data() + pos, '\0', buffer.size() - pos));
5         size_t size = (nextPos ? nextPos - buffer.data() : buffer.size()) - pos;
6         std::string str = std::string(buffer.data() + pos, size);
7
8         if (str.size() ≥ 2 && str.front() == ''' && str.back() == ''')
9             str = str.substr(1, str.size() - 2);
10
11        result->push_back(std::move(str));
12        pos += size;
13    }
14 }
```

```
1 void parseStrings(const Buffer &buffer, std::vector<std::string> *result) {
2     size_t pos = 0;
3     while (pos < buffer.size()) {
4         const char *nextPos = static_cast<const char *>(memchr(buffer.data() + pos, '\0', buffer.size() - pos));
5         size_t size = (nextPos ? nextPos - buffer.data() : buffer.size() - pos);
6         std::string str = std::string(buffer.data() + pos, size); // Extract next string.
7
8         if (str.size() ≥ 2 && str.front() == '\'' && str.back() == '\'')
9             str = str.substr(1, str.size() - 2);
10
11        result->push_back(std::move(str));
12        pos += size;
13    }
14 }
```



```
1 void parseStrings(const Buffer &buffer, std::vector<std::string> *result) {
2     size_t pos = 0;
3     while (pos < buffer.size()) {
4         const char *nextPos = static_cast<const char *>(memchr(buffer.data() + pos, '\0', buffer.size() - pos));
5         size_t size = (nextPos ? nextPos - buffer.data() : buffer.size()) - pos;
6         std::string str = std::string(buffer.data() + pos, size);
7
8         if (str.size() ≥ 2 && str.front() == ''' && str.back() == ''')
9             str = str.substr(1, str.size() - 2);
10
11        result->push_back(std::move(str)); // Write results &
12        pos += size; // prepare to parse the next string.
13    }
14 }
```



```
1 void parseStrings(const Buffer &buffer, std::vector<std::string> *result) {
2     size_t pos = 0;
3     while (pos < buffer.size()) {
4         const char *nextPos = static_cast<const char *>(memchr(buffer.data() + pos, '\0', buffer.size() - pos));
5         size_t size = (nextPos ? nextPos - buffer.data() : buffer.size()) - pos;
6         std::string str = std::string(buffer.data() + pos, size);
7
8         if (str.size() ≥ 2 && str.front() == '\"' && str.back() == '\"')
9             str = str.substr(1, str.size() - 2);
10
11        result->push_back(std::move(str));
12        pos += size + 1;
13    }
14 }
```

VS

```
1 std::vector<std::string> parseStrings(const Buffer &buffer, std::vector<std::string> *result) {
2     MemoryInput input(buffer);
3
4     std::string line;
5     while (input.readLine(&line, '\0'))
6         result->push_back(unquote(line));
7 }
```

```
1 void parseStrings(const Buffer &buffer, std::vector<std::string> *result) {
2     size_t pos = 0;
3     while (pos < buffer.size()) {
4         const char *nextPos = static_cast<const char *>(memchr(buffer.data() + pos, '\0', buffer.size() - pos));
5         size_t size = (nextPos ? nextPos - buffer.data() : buffer.size()) - pos;
6         std::string str = std::string(buffer.data() + pos, size);
7
8         if (str.size() ≥ 2 && str.front() == '\'' && str.back() == '\'')
9             str = str.substr(1, str.size() - 2);
10
11        result->push_back(std::move(str));
12        pos += size + 1;
13    }
14 }
```

VS

```
1 std::vector<std::string> parseStrings(const Buffer &buffer, std::vector<std::string> *result) {
2     MemoryInput input(buffer);
3
4     std::string line;
5     while (input.readLine(&line, '\0'))
6         result->push_back(unquote(line));
7 }
```


Rule #2:

Divide & Conquer: Дробите!

- Как только видите сложный код — думайте, каких абстракций вам не хватает!
- Дробить — это еще и про слои абстракции!
- Если ваш код все еще сложный — дробите дальше!

```
1 template<class T>
2 class QFuture {
3 public:
4     QFuture(const QFuture &other);
5
6     const_iterator begin() const;
7     const_iterator end() const;
8     QList<T> results() const;
9
10    void cancel();
11    T takeResult();
12    T result() const;
13
14    auto then(Function &&function);
15    auto then(QThreadPool *pool, Function &&function);
16    auto then(QObject *context, Function &&function);
17
18    bool isCanceled() const;
19    bool isFinished() const;
20    bool isRunning() const;
21    bool isStarted() const;
22    bool isValid() const;
23
24    // ...
25};
```



```
1 template<class T>
2 class QFuture {
3 public:
4     QFuture(const QFuture &other);
5
6     const_iterator begin() const; //
7     const_iterator end() const; // Async sequence interface.
8     QList<T> results() const; //
9
10    void cancel();
11    T takeResult();
12    T result() const;
13
14    auto then(Function &&function);
15    auto then(QThreadPool *pool, Function &&function);
16    auto then(QObject *context, Function &&function);
17
18    bool isCanceled() const;
19    bool isFinished() const;
20    bool isRunning() const;
21    bool isStarted() const;
22    bool isValid() const;
23
24    // ...
25 }
```



```
1 template<class T>
2 class QFuture {
3 public:
4     QFuture(const QFuture &other);
5
6     const_iterator begin() const;
7     const_iterator end() const;
8     QList<T> results() const;
9
10    void cancel();
11    T takeResult();
12    T result() const;
13
14    auto then(Function &&function); // Continuations.
15    auto then(QThreadPool *pool, Function &&function); // Continuations.
16    auto then(QObject *context, Function &&function); // Continuations.
17
18    bool isCanceled() const;
19    bool isFinished() const;
20    bool isRunning() const;
21    bool isStarted() const;
22    bool isValid() const;
23
24    // ...
25 }
```



```
1 template<class T>
2 class QFuture {
3 public:
4     QFuture(const QFuture &other);
5
6     const_iterator begin() const;
7     const_iterator end() const;
8     QList<T> results() const;
9
10    void cancel();
11    T takeResult();
12    T result() const;
13
14    auto then(Function &&function);
15    auto then(QThreadPool *pool, Function &&function);
16    auto then(QObject *context, Function &&function);
17
18    bool isCanceled() const; //
19    bool isFinished() const; //
20    bool isRunning() const; // State observers (not all of them).
21    bool isStarted() const; //
22    bool isValid() const;   //
23
24    // ...
25 }
```

```
1 template<class T>
2 class QFuture {
3 public:
4     QFuture(const QFuture &other);
5
6     const_iterator begin() const;
7     const_iterator end() const;
8     QList<T> results() const;
9
10    void cancel();
11    T takeResult();
12    T result() const;
13
14    auto then(Function &&function);
15    auto then(QThreadPool *pool, Function &&function);
16    auto then(QObject *context, Function &&function);
17
18    bool isCanceled() const;
19    bool isFinished() const;
20    bool isRunning() const; // How are these two different?
21    bool isStarted() const; // Gotta read the sources to figure out ...
22    bool isValid() const;
23
24    // ...
25 }
```

```
1 template<class T>
2 class QFuture {
3 public:
4     QFuture(const QFuture &other);
5
6     const_iterator begin() const;
7     const_iterator end() const;
8     QList<T> results() const;
9
10    void cancel();
11    T takeResult();
12    T result() const;
13
14    auto then(Function &&function);
15    auto then(QThreadPool *pool, Function &&function);
16    auto then(QObject *context, Function &&function);
17
18    bool isCanceled() const; //
19    bool isFinished() const; // Why do we even have all these state observers?
20    bool isRunning() const; // Why not a single State state() function?
21    bool isStarted() const; //
22    bool isValid() const;   //
23
24    // ...
25 }
```

```
1 template<class T>
2 class QWhatExactlyIsThisMonstrosity /* ^\_(ツ)_/^- */ {
3 public:
4     QWhatExactlyIsThisMonstrosity(const QWhatExactlyIsThisMonstrosity &other); // Shared future?
5
6     const_iterator begin() const; // // Async sequence?
7     const_iterator end() const; // // 
8     QList<T> results() const; // 
9
10    void cancel(); // Not so shared future?
11    T takeResult(); // Shared future?
12    T result() const;
13
14    auto then(Function &&function);
15    auto then(QThreadPool *pool, Function &&function);
16    auto then(QObject *context, Function &&function); // (╯°□°)╯ ≈ ┻━┻
17
18    bool isCanceled() const;
19    bool isFinished() const;
20    bool isRunning() const;
21    bool isStarted() const; // (╯@_@)╯︵ ┻━┻
22    bool isValid() const;
23
24    // P.S.: Don't even try to read the sources. They'll give you nightmares.
25 }
```

Rule #3:

Тратьте время на придумывание хороших имен!

"There are only two hard things in Computer Science: cache invalidation and naming things." — Phil Karlton

"Clean code reads like well-written prose." — Robert C. Martin

- Если что-то не получается назвать нормально — значит вы хотите странного, переделывайте свой дизайн.
- Не поддавайтесь соблазну назвать класс HandlerHelper. Все, что заканчивается на Helper — это признание поражения.
- Думайте над тем, в чем вообще концептуальная суть ваших абстракций.
"Что такое future?"
- Пользуйтесь thesaurus.com и ChatGPT.

```
1 template<class T>
2 class QFuturePart1 {
3 public:
4     void cancel();
5     const_iterator begin() const;
6     const_iterator end() const;
7     QList<T> results() const;
8     // ...
9 }
10
11 template<class T>
12 class QFuturePart2 {
13 public:
14     QFuture(const QFuture &other);
15     void cancel();
16     T result() const;
17     // ...
18 }
19
20 template<class T>
21 class QFuturePart3 {
22 public:
23     QFuture(QFuture &&other);
24     void cancel();
25     T takeResult();
26     // ...
27 }
```

```
1 template<class T>
2 class QGenerator {
3 public:
4     void cancel();
5     const_iterator begin() const;
6     const_iterator end() const;
7     QList<T> results() const;
8     // ...
9 }
10
11 template<class T>
12 class QSharedFuture {
13 public:
14     QFuture(const QFuture &other);
15     void cancel();
16     T result() const;
17     // ...
18 }
19
20 template<class T>
21 class QUniqueFuture {
22 public:
23     QFuture(QFuture &&other);
24     void cancel();
25     T takeResult();
26     // ...
27 }
```

Rule #2 + Rule #3:

Дробление и нейминг — это про абстракции, а не про классы!

- Ваш код — это перевод абстракций в вашей голове на C++.
- Если в голове нет порядка, то в коде его тоже не будет.

Что такое future?

- Это абстракция над вычислением.
- Но это и абстракция над контекстом, в котором это вычисление производится!

```
1  template<class T>
2  class QSharedFuture {
3  public:
4      auto then(Function &&function); // Where will it run?
5                                // Can I pass in a function that does a lot of work?
6      // ...                         // Can we do better?
7  };
```

```
1 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {
2     return network
3         .request(makeNetworkRequest(opts))
4         .then([](std::string_view jsonData) {
5             TwitterPosts result;
6             deserialize(Json::parse(jsonData), &result);
7             return result;
8         });
9 }
10
11 void myAwesomeFunction(Network &network) {
12     TwitterPosts posts = fetchTwitterPosts(network, TwitterFetchOptions("@stroustrup", 20))
13         .run(globalThreadPool())
14         .join();
15     std::print("{}\n", posts);
16 }
17 }
```

```
1 // Returns a Task that will fetch twitter posts with the provided options when run.
2 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {
3     return network
4         .request(makeNetworkRequest(opts))
5         .then([](std::string_view jsonData) {
6             TwitterPosts result;
7             deserialize(Json::parse(jsonData), &result);
8             return result;
9         });
10 }
11
12 void myAwesomeFunction(Network &network) {
13     TwitterPosts posts = fetchTwitterPosts(network, TwitterFetchOptions("@stroustrup", 20))
14         .run(globalThreadPool())
15         .join();
16     std::print("{}\n", posts);
17 }
```

```
1 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {
2     return network
3         .request(makeNetworkRequest(opts)) // Network::request() also returns a Task!
4         .then([](std::string_view jsonData) {
5             TwitterPosts result;
6             deserialize(Json::parse(jsonData), &result);
7             return result;
8         });
9 }
10
11 void myAwesomeFunction(Network &network) {
12     TwitterPosts posts = fetchTwitterPosts(network, TwitterFetchOptions("@stroustrup", 20))
13         .run(globalThreadPool())
14         .join();
15     std::print("{}\n", posts);
16 }
17 }
```

```
1 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {
2     return network
3         .request(makeNetworkRequest(opts))
4         .then([](std::string_view jsonData) { // Tasks are composable!
5             TwitterPosts result;
6             deserialize(Json::parse(jsonData), &result);
7             return result;
8         });
9 }
10
11 void myAwesomeFunction(Network &network) {
12     TwitterPosts posts = fetchTwitterPosts(network, TwitterFetchOptions("@stroustrup", 20))
13         .run(globalThreadPool())
14         .join();
15     std::print("{}\n", posts);
16 }
17 }
```

```
1 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {
2     return network
3         .request(makeNetworkRequest(opts))
4         .then([](std::string_view jsonData) {
5             TwitterPosts result;
6             deserialize(Json::parse(jsonData), &result);
7             return result;
8         });
9 }
10 }
11
12 void myAwesomeFunction(Network &network) { // Fetch Bjarne's 20 latest posts ↓
13     TwitterPosts posts = fetchTwitterPosts(network, TwitterFetchOptions("@stroustrup", 20))
14         .run(globalThreadPool())
15         .join();
16     std::print("{}\n", posts);
17 }
```



```
1 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {
2     return network
3         .request(makeNetworkRequest(opts))
4         .then([](std::string_view jsonData) {
5             TwitterPosts result;
6             deserialize(Json::parse(jsonData), &result);
7             return result;
8         });
9 }
10
11 void myAwesomeFunction(Network &network) {
12     TwitterPosts posts = fetchTwitterPosts(network, TwitterFetchOptions("astroustrup", 20))
13         .run(globalThreadPool())
14         .join();
15     std::print("{}\n", posts);
16 }
17 }
```

- Task и контекст выполнения ортогональны!

Rule #4:

Создавайте ортогональные и взаимозаменяемые абстракции

- Дробите абстракции на взаимозаменяемые кусочки.
⇒ Реализуйте $N+M$ вариантов вашего кода вместо $N \times M$.
- Вдохновляйтесь STL. Ортогональность данных и алгоритмов можно успешно распространить на очень многие предметные области!
- Если у вас есть два схожих концепта — думайте, должны ли они быть взаимозаменяемы. Выбирайте подходящий под вашу задачу тип полиморфизма:
 - Виртуальные функции для динамического полиморфизма.
 - Overload sets для статического полиморфизма.

```
1 // Static polymorphism.  
2  
3 struct TwitterPost {  
4     std::string author;  
5     std::string text;  
6 };  
7  
8 struct TwitterUser {  
9     // ...  
10};  
11  
12 void deserialize(const Json &json, TwitterPost *value) {  
13     value->author = json["author"];  
14     value->text = json["text"];  
15 }  
16  
17 void deserialize(const Json &json, TwitterUser *value) {  
18     // ...  
19 }
```

VS

```
1 // Static polymorphism.  
2  
3 struct TwitterPost {  
4     std::string author;  
5     std::string text;  
6 };  
7  
8 struct TwitterUser {  
9     // ...  
10};  
11  
12 void deserialize(const Json &json, TwitterPost *value) {  
13     value->author = json["author"];  
14     value->text = json["text"];  
15 }  
16  
17 void deserialize(const Json &json, TwitterUser *value) {  
18     // ...  
19 }
```

```
1 // Dynamic polymorphism.  
2  
3 class Deserializable {  
4 public:  
5     virtual void deserialize(const Json &json) = 0;  
6  
7 protected:  
8     ~Deserializable() = default;  
9 };  
10  
11 struct TwitterPost : Deserializable {  
12     std::string author;  
13     std::string text;  
14  
15     virtual void deserialize(const Json &json) override {  
16         author = json["author"];  
17         text = json["text"];  
18     }  
19 };  
20  
21 struct TwitterUser : Deserializable {  
22     // ...  
23  
24     virtual void deserialize(const Json &json) override {  
25         // ...  
26     }  
27 };
```

[123456789] Move - Examine [+-] monster or [*_] spot - [Z] Abort

Ig skel St:39 Le:26 Wi:20 Dx:25 To:41 Ch:29 Ap:14 Ma:19 Pe:20 N=
DV/PV:105/44 H:584(587) P:84(136) Exp:34/7636445 MT Sp:126
Blessed Satiated Burdened Coward


```
1 class Equipment {
2 public:
3     virtual void onUse() = 0;
4     // ...
5 };
6
7 class Weapon : public Equipment {
8 public:
9     virtual void onAttack(Monster &monster) = 0;
10    // ...
11};
```

```
1 class Equipment {
2     public:
3         virtual void onUse() = 0;
4         // ...
5     };
6
7 class Weapon : public Equipment {
8     public:
9         virtual void onAttack(Monster &monster) = 0;
10        // ...
11    };
12
13 class VampiricSword : public Weapon {
14     public:
15         virtual void onAttack(Monster &monster) override {
16             Damage damage(DMG_PHYSICAL, rollDice(3, 5) + 5);
17             monster.takeDamage(damage);
18             Damage healing(DMG_DARKMAGIC, damage.amount / 2);
19             owner().heal(healing);
20         }
21         // ...
22     };

```

```
1 class Equipment {
2 public:
3     virtual void onUse() = 0;
4     // ...
5 };
6
7 class Weapon : public Equipment {
8 public:
9     virtual void onAttack(Monster &monster) = 0;
10    // ...
11 };
12
13 class VampiricSword : public Weapon {
14 public:
15     virtual void onAttack(Monster &monster) override {
16         Damage damage(DMG_PHYSICAL, rollDice(3, 5) + 5); // Create a damage instance for 3d5+5.
17         monster.takeDamage(damage);
18         Damage healing(DMG_DARKMAGIC, damage.amount / 2);
19         owner().heal(healing);
20     }
21     // ...
22 };
```

```
1 class Equipment {
2     public:
3         virtual void onUse() = 0;
4         // ...
5     };
6
7 class Weapon : public Equipment {
8     public:
9         virtual void onAttack(Monster &monster) = 0;
10        // ...
11    };
12
13 class VampiricSword : public Weapon {
14     public:
15         virtual void onAttack(Monster &monster) override {
16             Damage damage(DMG_PHYSICAL, rollDice(3, 5) + 5);
17             monster.takeDamage(damage);           // Send it to monster, this call updates damage.
18             Damage healing(DMG_DARKMAGIC, damage.amount / 2);
19             owner().heal(healing);
20         }
21         // ...
22    };

```

```
1 class Equipment {
2     public:
3         virtual void onUse() = 0;
4         // ...
5     };
6
7 class Weapon : public Equipment {
8     public:
9         virtual void onAttack(Monster &monster) = 0;
10        // ...
11    };
12
13 class VampiricSword : public Weapon {
14     public:
15         virtual void onAttack(Monster &monster) override {
16             Damage damage(DMG_PHYSICAL, rollDice(3, 5) + 5);
17             monster.takeDamage(damage);
18             Damage healing(DMG_DARKMAGIC, damage.amount / 2); // Heal the player for half the damage dealt.
19             owner().heal(healing); // Also, it's DARKMAGIC!
20         }
21         // ...
22    };

```



```
1 class Event { // Every interaction in the game is an event, or several events.  
2 public:  
3     EventType type;  
4     // ...  
5 }
```



```
1 class Event {
2 public:
3     EventType type;
4     // ...
5 }
6
7 class Behaviour {
8 public:
9     virtual void process(Event *event) = 0;
10    // ...
11 }
12
13 class Equipment { // No subclassing,
14 public:           // all logic is in behaviours ↓
15     std::vector<std::unique_ptr<Behaviour>> behaviours;
16     // ...
17 };
```

```
1 // First event: sent to player's items to populate the damage rolls.
2 class AttackOutEvent : public Event {
3 public:
4     std::vector<Damage> damageRolls;
5     // ...
6 };
7
8 // Second event: sent to monster's items to apply armor & protection.
9 class AttackInEvent : public Event {
10 public:
11     std::vector<Damage> damageRolls;
12     // ...
13 };
14
15 // Third event: sent back to player's items to notify of success / failure.
16 class DamageEvent : public Event {
17 public:
18     std::vector<Damage> damageRolls;
19     // ...
20 };
```

```
1 class WeaponBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != ATTACK_OUT_EVENT)
4             return;
5         AttackOutEvent *e = static_cast<AttackOutEvent *>(event);
6
7         e->damageRolls.push_back(Damage(
8             owner(),
9             DMG_PHYSICAL,
10            owner().rollAttack()
11        ));
12    }
13 }
14
15 class ArmorBehavior : public Behaviour {
16     virtual void process(Event *event) override {
17         if (event->type != ATTACK_IN_EVENT)
18             return;
19         AttackInEvent *e = static_cast<AttackInEvent *>(event);
20
21         for (Damage &damage : e->damageRolls)
22             if (damage.type == DMG_PHYSICAL)
23                 damage.amount = std::max(0, damage.amount - owner().armorClass());
24     }
25 };
```

```
1 class WeaponBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != ATTACK_OUT_EVENT) // Process only outgoing attack events.
4             return;
5         AttackOutEvent *e = static_cast<AttackOutEvent *>(event);
6
7         e->damageRolls.push_back(Damage(
8             owner(),
9             DMG_PHYSICAL,
10            owner().rollAttack()
11        ));
12    }
13 }
14
15 class ArmorBehavior : public Behaviour {
16     virtual void process(Event *event) override {
17         if (event->type != ATTACK_IN_EVENT)
18             return;
19         AttackInEvent *e = static_cast<AttackInEvent *>(event);
20
21         for (Damage &damage : e->damageRolls)
22             if (damage.type == DMG_PHYSICAL)
23                 damage.amount = std::max(0, damage.amount - owner().armorClass());
24     }
25 };
```

```
1 class WeaponBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != ATTACK_OUT_EVENT)
4             return;
5         AttackOutEvent *e = static_cast<AttackOutEvent *>(event);
6
7         e->damageRolls.push_back(Damage( // Record a damage roll
8             owner(),
9             DMG_PHYSICAL,
10            owner().rollDamage()
11        ));
12    }
13 }
14
15 class ArmorBehavior : public Behaviour {
16     virtual void process(Event *event) override {
17         if (event->type != ATTACK_IN_EVENT)
18             return;
19         AttackInEvent *e = static_cast<AttackInEvent *>(event);
20
21         for (Damage &damage : e->damageRolls)
22             if (damage.type == DMG_PHYSICAL)
23                 damage.amount = std::max(0, damage.amount - owner().armorClass());
24     }
25 };
```

```
1 class WeaponBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != ATTACK_OUT_EVENT)
4             return;
5         AttackOutEvent *e = static_cast<AttackOutEvent *>(event);
6
7         e->damageRolls.push_back(Damage(
8             owner(),
9             DMG_PHYSICAL,
10            owner().rollDamage()
11        ));
12    }
13 }
14
15 class ArmorBehavior : public Behaviour {
16     virtual void process(Event *event) override {
17         if (event->type != ATTACK_IN_EVENT) // Process only incoming attack events.
18             return;
19         AttackInEvent *e = static_cast<AttackInEvent *>(event);
20
21         for (Damage &damage : e->damageRolls)
22             if (damage.type == DMG_PHYSICAL)
23                 damage.amount = std::max(0, damage.amount - owner().armorClass());
24     }
25 };
```

```
1 class WeaponBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != ATTACK_OUT_EVENT)
4             return;
5         AttackOutEvent *e = static_cast<AttackOutEvent *>(event);
6
7         e->damageRolls.push_back(Damage(
8             owner(),
9             DMG_PHYSICAL,
10            owner().rollDamage()
11        ));
12    }
13 }
14
15 class ArmorBehavior : public Behaviour {
16     virtual void process(Event *event) override {
17         if (event->type != ATTACK_IN_EVENT)
18             return;
19         AttackInEvent *e = static_cast<AttackInEvent *>(event);
20
21         for (Damage &damage : e->damageRolls) // Apply protection against physical damage.
22             if (damage.type == DMG_PHYSICAL)
23                 damage.amount = std::max(0, damage.amount - owner().armorClass());
24     }
25 };
```

```
1 class VampiricBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != DAMAGE_EVENT)
4             return;
5         DamageEvent *e = static_cast<DamageEvent *>(event);
6
7         int totalDamage = 0;
8         for (const Damage &damage : e->damageRolls) {
9             if (damage.source == owner() && damage.type == DMG_PHYSICAL)
10                 totalDamage += damage.amount;
11
12         if (totalDamage <= 1)
13             return;
14
15         sendEvent(
16             owner().owner(),
17             SpellEvent(
18                 owner(),
19                 SPELL_VAMPIRIC_HEALING,
20                 totalDamage / 2
21             )
22         );
23     }
24 }
```

```
1 class VampiricBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != DAMAGE_EVENT) // Handle only damage notification events.
4             return;
5         DamageEvent *e = static_cast<DamageEvent *>(event);
6
7         int totalDamage = 0;
8         for (const Damage &damage : e->damageRolls) {
9             if (damage.source == owner() && damage.type == DMG_PHYSICAL)
10                 totalDamage += damage.amount;
11
12         if (totalDamage <= 1)
13             return;
14
15         sendEvent(
16             owner().owner(),
17             SpellEvent(
18                 owner(),
19                 SPELL_VAMPIRIC_HEALING,
20                 totalDamage / 2
21                     )
22                 );
23     }
24 }
```

```
1 class VampiricBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != DAMAGE_EVENT)
4             return;
5         DamageEvent *e = static_cast<DamageEvent *>(event);
6
7         int totalDamage = 0; // Calculate total physical damage dealt by this weapon.
8         for (const Damage &damage : e->damageRolls) {
9             if (damage.source == owner() && damage.type == DMG_PHYSICAL)
10                 totalDamage += damage.amount;
11
12         if (totalDamage <= 1)
13             return;
14
15         sendEvent(
16             owner().owner(),
17             SpellEvent(
18                 owner(),
19                 SPELL_VAMPIRIC_HEALING,
20                 totalDamage / 2
21             )
22         );
23     }
24 }
```

```
1 class VampiricBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != DAMAGE_EVENT)
4             return;
5         DamageEvent *e = static_cast<DamageEvent *>(event);
6
7         int totalDamage = 0;
8         for (const Damage &damage : e->damageRolls) {
9             if (damage.source == owner() && damage.type == DMG_PHYSICAL)
10                 totalDamage += damage.amount;
11
12         if (totalDamage < 2) // Vampirism kicks in at totalDamage ≥ 2.
13             return;
14
15         sendEvent(
16             owner().owner(),
17             SpellEvent(
18                 owner(),
19                 SPELL_VAMPIRIC_HEALING,
20                 totalDamage / 2
21                     )
22                 );
23     }
24 }
```

```
1 class VampiricBehavior : public Behaviour {
2     virtual void process(Event *event) override {
3         if (event->type != DAMAGE_EVENT)
4             return;
5         DamageEvent *e = static_cast<DamageEvent *>(event);
6
7         int totalDamage = 0;
8         for (const Damage &damage : e->damageRolls) {
9             if (damage.source == owner() && damage.type == DMG_PHYSICAL)
10                 totalDamage += damage.amount;
11
12         if (totalDamage <= 1)
13             return;
14
15         sendEvent(                      // Send vampiric healing event to the player
16             owner().owner(),          // for half the damage dealt.
17             SpellEvent(
18                 owner(),
19                 SPELL_VAMPIRIC_HEALING,
20                 totalDamage / 2
21             )
22         );
23     }
24 }
```

Rule #4:

Создавайте ортогональные и взаимозаменяемые абстракции

На выходе имеем невероятную гибкость:

```
1 // With a little bit of DSL help:  
2 auto VampiricSword      = WeaponBehavior() & VampiricBehavior();  
3 auto SpikedShield        = WeaponBehavior() & ArmorBehavior();  
4 auto CrownOfChaos       = ArmorBehavior() & CorruptingBehavior();  
5 auto SwordOfFireballs   = WeaponBehavior() & RandomCastBehavior(SPELL_FIREBALL, 0.01);  
6  
7 auto RingOfIce          = ResistanceBehavior(DMG_WATER, 0.0) &  
8                           VulnerabilityBehavior(DMG_FIRE, 2.0) &  
9                           FreezingBehavior();  
10 // ...
```

И теперь ваш гейм-дизайнер здоровается с вами за руку!

А еще мы на самом деле придумали часть Entity Component System (ECS). Рекомендую доклад Brian Bucklew про Caves of Qud.

```
1 // Fetch twitter posts asynchronously.
2 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {
3     return network
4         .request(makeNetworkRequest(opts))
5         .then([](std::string_view jsonData) {
6             TwitterPosts result;
7             deserialize(Json::parse(jsonData), &result);
8             return result;
9         });
10 }
```

```
1 // Fetch twitter posts asynchronously.  
2 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     return network  
        .request(makeNetworkRequest(opts))  
        .then([](std::string_view jsonData) {  
            TwitterPosts result;  
            deserialize(Json::parse(jsonData), &result);  
            return result;  
        });  
10 }
```

VS

```
1 // Fetch twitter posts ... asynchronously?  
2 TwitterPosts fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     auto jsonData = network.request(makeNetworkRequest(opts));  
4     TwitterPosts result;  
5     deserialize(Json::parse(jsonData), &result);  
6     return result;  
7 }
```

```
1 // Fetch twitter posts asynchronously.  
2 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     return network  
4         .request(makeNetworkRequest(opts))  
5         .then([](std::string_view jsonData) {  
6             TwitterPosts result;  
7             deserialize(Json::parse(jsonData), &result);  
8             return result;  
9         });  
10 }
```

VS

```
1 // Fetch twitter posts asynchronously!  
2 CoResult<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     auto jsonData = co_await network.request(makeNetworkRequest(opts));  
4     TwitterPosts result;  
5     deserialize(Json::parse(jsonData), &result);  
6     co_return result;  
7 }
```

```
1 // Fetch twitter posts asynchronously.  
2 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     return network  
4         .request(makeNetworkRequest(opts))  
5         .then([](std::string_view jsonData) {  
6             TwitterPosts result;  
7             deserialize(Json::parse(jsonData), &result);  
8             return result;  
9         });  
10 }
```

VS

```
1 // Calls allocate a coroutine frame.  
2 CoResult<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     auto jsonData = co_await network.request(makeNetworkRequest(opts)); // Also allocates.  
4     TwitterPosts result;  
5     deserialize(Json::parse(jsonData), &result);  
6     co_return result;  
7 }
```

```
1 // Fetch twitter posts asynchronously.  
2 Task<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     return network  
4         .request(makeNetworkRequest(opts))  
5         .then([](std::string_view jsonData) {  
6             TwitterPosts result;  
7             deserialize(Json::parse(jsonData), &result);  
8             return result;  
9         });  
10 }
```

VS

```
1 // Should be called with co_await, turning caller function into a coroutine ...  
2 CoResult<TwitterPosts> fetchTwitterPosts(Network &network, const TwitterFetchOptions &opts) {  
3     auto jsonData = co_await network.request(makeNetworkRequest(opts));  
4     TwitterPosts result;  
5     deserialize(Json::parse(jsonData), &result);  
6     co_return result;  
7 }
```

Cm. [What Color is Your Function](#) by Bob Nystrom .

Rule #5:

Итерируйтесь!

- Начинайте с написания клиентского кода, пробуйте разные варианты API, оставляйте тот, что лучше подходит под ваши use cases.
- Иногда решения нет. Для приятной глазу асинхронности нужна поддержка stackful корутин из коробки. *Или можно сходить на доклад @apolukhin.*
- Иногда можно поправить core language (привет `mdspan`).
- Иногда у вас уже есть плохой API — не стесняйтесь писать адаптеры (привет `std :: chrono`).
- Ваша цель — достаточно хороший API, соответствующий вашим требованиям. Идеальных API не бывает.

Cheat Sheet

- Проектируйте API так, чтобы его нельзя было использовать неправильно.

Все возможные способы использования вашего API должны или отрабатывать корректно, или завершаться ошибкой!

- Divide & Conquer: Дробите!

На классы, на модули, на функции, на слои абстракции.

- Тратьте время на придумывание хороших имен!

Если не получается придумать нормальное имя — значит вы придумали плохую абстракцию.

- Создавайте ортогональные и взаимозаменяемые абстракции.

Хорошее правило которое почти всегда верно — данные ортогональны логике. Вдохновляйтесь STL!

- Итерируйтесь!

И ждите статью на Хабре.

END

END