

C++ Club UK

Gleb Dolgich

2019-05-02

Second Annual C++ Foundation Developer Survey “Lite”

<https://isocpp.org/blog/2019/04/second-annual-cpp-foundation-developer-survey-lite>

<https://www.surveymonkey.com/r/NCMCJDZ>

Top 25 C++ API design mistakes and how to avoid them

<https://www.acodersjourney.com/top-25-cplusplus-api-design-mistakes-and-how-to-avoid-them/>

Book: Martin Reddy - API Design for C++

- ▶ [Amazon US](#)
- ▶ [Amazon UK](#)
- ▶ [Safari Books Online](#)

https://www.reddit.com/r/cpp/comments/bh5b75/top_25_c_api_design_mistakes_and_how_to_avoid_them/

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable

<https://www.youtube.com/watch?v=os7cqJ5qlzo>

https://www.reddit.com/r/cpp/comments/bifsdx/herb_sutter_accu_2019_defragmenting_c_making/

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable

ACCU
2019

De-Fragmenting C++: Making exceptions more affordable and usable - Herb Sutter

isocpp.org 2018-02 survey

- ▶ Most “C++” projects ban exceptions in whole or in part.
 - ▶ ⇒ Not really using Standard C++, which requires exceptions.
 - ▶ Using a divergent incompatible language dialect with different idioms (e.g., factory functions instead of constructors).
 - ▶ Using a divergent incompatible std:: library dialect (e.g., EASTL, _HAS_EXCEPTIONS=0), or none at all (e.g., Epic).

Q7: [Are exceptions] allowed in your current project? (N=3,240)

A donut chart illustrating the distribution of responses to Question 7. The chart is divided into three segments: a large light green segment representing 'Yes: Allowed pretty much everywhere' at 48%, a yellow segment representing 'Partial: Allowed in some parts of the code' at 32%, and a small red segment representing 'No: Not allowed' at 20%.

Response Category	Percentage
No: Not allowed	20%
Partial: Allowed in some parts of the code	32%
Yes: Allowed pretty much everywhere	48%

A photograph of Herb Sutter, a man with glasses and a patterned shirt, standing on a stage and gesturing with his right hand. He is speaking to an audience whose silhouettes are visible in the foreground. A presentation slide is visible behind him, though its content is mostly illegible. The stage lighting is purple and pink.

7

conference.accu.org

@ACCUConf

5/16

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable

The diagram illustrates the state of exception handling in C++ through a series of overlapping circles:

- Standard C++**: A large circle containing:
 - Exceptions enabled
 - Constructors and operators (especially) report failure by throwing
- Dialects**: A cluster of smaller circles:
 - Boost Outcome
 - etc. ...
 - no-exceptions
 - errno
 - std::experimental::expected
 - result<>
 - Boost expected

A small number "13" is located at the bottom right of the diagram area.

ACCU 2019 logo is in the top left corner.

De-Fragmenting C++: Making exceptions more affordable and usable - Herb Sutter is displayed prominently at the top center.

conference.accu.org and **@ACCUConf** are at the bottom.

A photograph of Herb Sutter, a man with glasses and a patterned shirt, standing at a podium and speaking to an audience. The background is purple and features the ACCU 2019 logo.

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable

Root cause: Today's EH not “zero-overhead”

- ▶ Violates C++’s zero-overhead principle in two ways.
- 1. “I can’t afford to **enable** exception handling.”
 - ▶ Just turning on EH incurs space overhead.
 - ▶ Zero overhead principle, part 1: “Don’t pay for what you don’t use.”
- 2. “I can’t afford to **throw** an exception.”
 - ▶ Throwing an exception incurs not-statically-boundable space and time overhead.
 - ▶ Throwing an exception usually less efficient than returning code/expected<> by hand.
 - ▶ Zero overhead principle, part 2: “When you do use it you can’t reasonably write it better by hand” including by using alternatives.

19

conference.accu.org

@accuConf

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable

ACCU
2019

De-Fragmenting C++: Making exceptions more
affordable and usable - Herb Sutter

Taxonomy

	What to use	Report-to handler	Handler species
A. Corruption of the abstract machine (e.g., stack exhaustion)	Terminate	User	Human
B. Programming bug (e.g., precondition violation)	Asserts, log checks, contracts, ...	Programmer	Human
C. Recoverable error (e.g., host not found)	Throw exception, error code, etc.	Calling code	Code



conference.accu.org

@ACCUConf

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable

ACCU 2019 De-Fragmenting C++: Making exceptions more affordable and usable - Herb Sutter

Core issues: Zero-overhead + **determinism**

- ▶ Exceptions are great: Distinct “error” paths, can’t ignore, auto propagation.
- ▶ But: Inherently not zero-overhead, not deterministic.
- ▶ “**Throwing objects of dynamic types...** ⇒ dynamic allocation + type erasure
... and catching using RTTI.” ⇒ dynamic casting (special)

- ▶ Proposal:
 - ▶ “**Throwing values of static types...** ⇒ stack allocation, share return channel
... and catching by value.” ⇒ no dynamic casting, just value comparison
 - ▶ Isomorphic to error codes, identical space/time overhead and **predictability**.
 - ▶ Share return channel ⇒ potential for negative overhead abstraction.
 - ▶ If a function agrees (**opts in**) that any exceptions it emits are **values of one statically known type**, we can implement it with zero dynamic/non-local overheads.
 - ▶ **not a breaking change**



cation, share return channel
nic casting, just value comparison
ead and predictability
ad abstraction.
its are w
non-local

26

conference.accu.org @accuConf

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable



De-Fragmenting C++: Making exceptions more
affordable and usable - Herb Sutter

1. Throw values, not types

- ▶ As-if returning `union{ Success; Error; } + bool`, using the same return channel (incl. registers + CPU flag for discriminant).
 - ▶ Best of exceptions and error codes (and fully prior-art):
 - Exactly exceptions' programming model (`throw, try, catch`).
 - Exactly error codes' return-value implementation (w/o monopolizing channel).
 - ▶ Doubles down on value semantics. (Cf: C++11 move semantics.)
- ▶ If you love:
 - ▶ Exceptions: Can use them more widely, removing perf reasons to avoid/ban.
 - ▶ Expected/Outcome: Gets language support, propagates automatically.
 - ▶ Error codes: Doesn't monopolize return channel, propagates automatically, and the caller can't forget to check it and gets distinct success/error paths.
 - ▶ Termination (fail-fast): Hook the propagation notification (see §4.1.4).



27

conference.accu.org

@accuConf

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable



Core proposal summary

- ▶ A static-exception-specification `throws` ⇒ function can throw `std::error`, an evolution of `std::error_code` + SG14-driven improvements already underway.

```
string f() throws {
    if (flip_a_coin()) throw arithmetic_error::something;
    return "xyzzy"s + "plover";           // bad_alloc → std::errc::ENOMEM
}
string g() throws { return f() + "plugh"; }      // bad_alloc → std::errc::ENOMEM
int main() {
    try {
        auto result = g();
        cout << "success, result is: " << result;
    } catch(error err) {                  // catch by value
        cout << "failed, error is: " << err.error();
    }
}
```



28

conference.accu.org

@accuConf

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable

ACCU 2019 De-Fragmenting C++: Making exceptions more affordable and usable - Herb Sutter

Dynamic type(-*erased*) vs. static type

Today (pseudocode)	Proposed (pseudocode)
// throw site: "throw MyException(value)" return (void*) new MyException(value);	// throw site: "throw std::error(domain,value)" return std::error (domain,value); // no alloc
// ...	// ...
// propagate	// propagate
// ...	// ...
// catch site: "catch (EBase& e) /*...*/" by reference if (auto e = special_dynamic_cast <EBase*>(pvoid; e) /*...*/)	// catch site: "catch (std::error e)" by value if (e.failed) /*...*/ // no RTTI



conference.accu.org @accuConf

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable



Language-independent fact

- ▶ Many heap allocation failures (aka out of memory, OOM) are unrecoverable
 - ▶ Appears to be inherent: e.g., impossible to thoroughly test, or must be written carefully
 - ▶ But some current/future code is OOM-safe, and we don't want to lose that
- ▶ Case 1: Failure to allocate big buffer || opt-in allocator (e.g., `new[100000], MyAlloc`)
 - ▶ Causes: Optimistic or unsanitized size input
 - ▶ **Unwinding+recovering by running “normal code” is possible: Throwing/returning is OK**
 - ▶ Recovery possible: Fall back to smaller buffer, or fail requested operation
- ▶ Case 2: Failure to perform “small” allocation && default allocator (e.g., `new int`)
 - ▶ Cause: Resource limit (actual exhaustion or fragmentation): So like stack exhaustion
 - ▶ **Unwinding+recovering) by running “normal code” not possible: Throwing/returning not OK**
 - ▶ ⇒ By default: Don't throw, terminate (with `terminate_handler` support to opt out)



40

conference.accu.org

@accuConf

Herb Sutter [ACCU 2019] De-fragmenting C++: Making exceptions more affordable and usable



4. Proposed extension: *try* expressions

- ▶ Good news: Exceptional control flow is **automatic**.
- ▶ Bad news: Exception control flow is **invisible**.
 - ▶ Hard to reason about exceptions, especially in legacy code.
- ▶ Proposal: **try** before an expression/statement where a subexpression can throw.
 - ▶ Makes exceptional paths visible.
 - ▶ If we required it in new code: **Compile-time guarantees** (e.g., no “throw” ⇒ noexcept).

```
string f() throws {  
    if (flip_a_coin()) throw arithmetic_error::something;  
    return try "xyzzy"s + "plover";           // greppable  
    try string s("xyzzy");                  // same, just showing statement form too  
    try return s + "plover";  
}  
  
string g() throws { return try f() + "plugh"; }
```

45



Error Codes and Error Handling

<https://www.randygaul.net/2019/04/26/error-codes-and-error-handling/>

Error codes are better.

https://www.reddit.com/r/cpp/comments/bhysup/error_codes_and_error_handling/

Twitter

 **qntm**
@qntm

A group of hedgehogs is called an array, which means that everything in an array is a hedgehog, I don't make the rules pic.twitter.com/b74ht98yQh

Etymology

The name *hedgehog* came into use around the year 1450, derived from the Middle English *heyghoge*, from *heyg*, *hegge* ("hedge"), because it frequents hedgerows, and *hoge*, *hogge* ("hog"), from its piglike snout.^[4] Other names include *urchin*, *hedgepig* and *furze-pig*. **The collective noun for a group of hedgehogs is array.**

2,744 Likes 1,286 Retweets

12 Mar 2019 at 01:44 via Twitter Web Client