# C++ static code analysis: "std::uncaught_exception" should not be used

2 minutes

`bool std::uncaught_exception()` allows you to know whether a thread is in an exception stack unwinding context. However, its practical functionality was restricted.

C++17 deprecates `bool std::uncaught_exception()` and introduces `int std::uncaught_exceptions()` which returns the number of uncaught exceptions. The code example below shows how you can benefit from this new improved function.

`std::uncaught_exception` has been removed in C++20.

This rule will flag any usage of `std::uncaught_exception`.

## Noncompliant Code Example

```
class Transaction {

  // ...

  ~Transaction() {
    if (!std::uncaught_exception()) { // Noncompliant, replace
std::uncaught_exception by std::uncaught_exceptions
      // commit
    } else {
      // rollback
    }
  }
};
```

## Compliant Solution

The following example shows how `std::uncaught_exceptions` can be used to determine in ~`Transaction` if a new exception was thrown since `t1`/`t2` creation.

```
class Transaction {

  // ...

  ~Transaction() {
    if (initialUncaughtExceptions == std::uncaught_exceptions()) {
      // commit
    } else {
      // rollback
    }
  }

  // ...

  int initialUncaughtExceptions = std::uncaught_exceptions();
};

int f() {
  try {
    Transaction t1;
    // ... something here could throw
  } catch(...) {
    Transaction t2;
    // ... something here could throw
```

```
 }
}
```