Error handling

90% of all catastrophic software failures were caused by poor error handling code.

Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-intensive Systems. Yuan et. al. 2014

Error handling

```
int main(){
   int size;
   std::cin>>size;
   std::vector<int> vec(size);
   // ...
}
```

- What if size is negative?

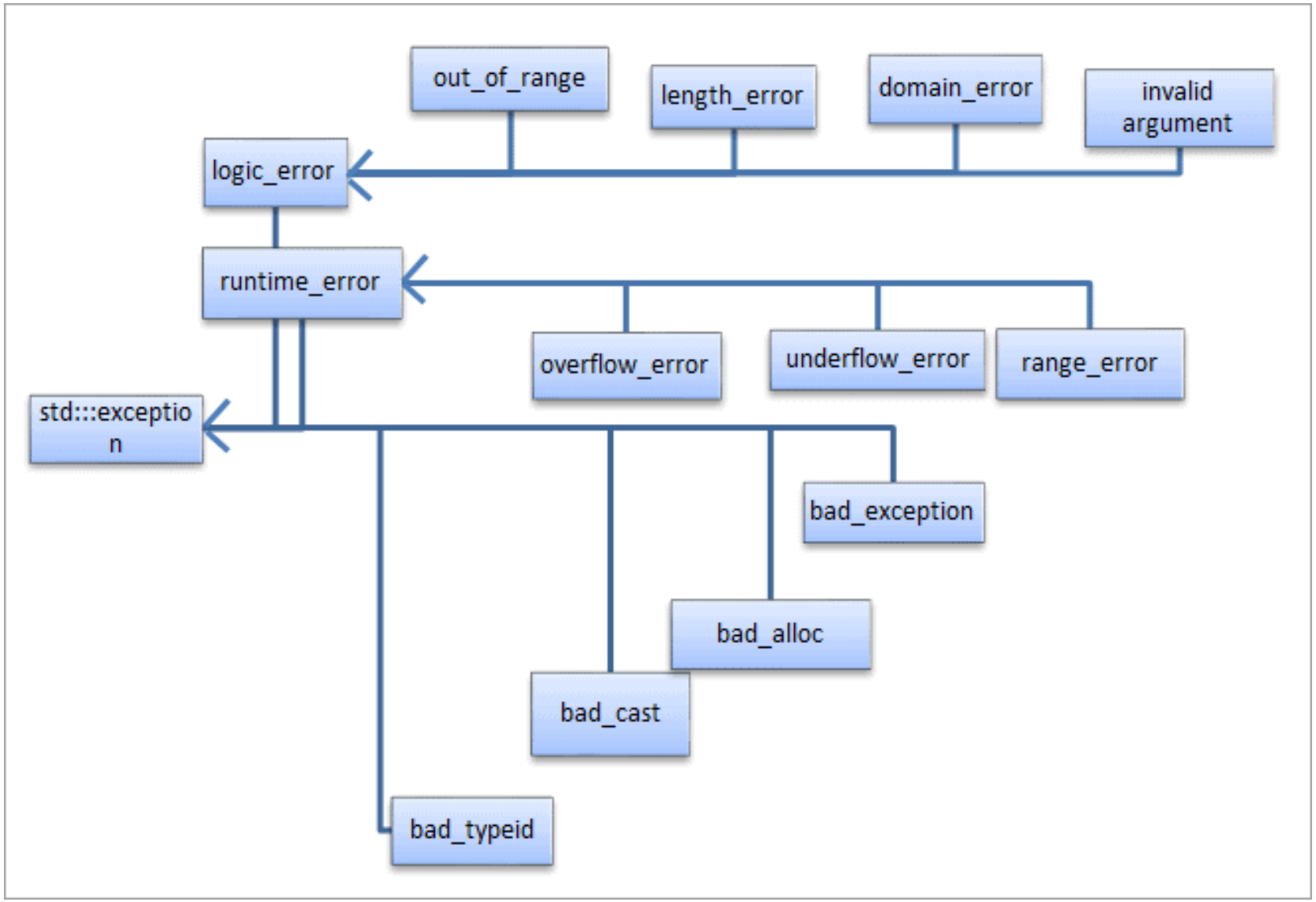
Exceptions

```
try {
  // Code that may throw an exception
 } catch (const ExceptionType1& e) {
  // Handle the exception
 } catch (const ExceptionType2& e) {
  // Handle the exception
 }
```

Exceptions

```
try {
  // Code that may throw an exception
 } catch (const std::exception& e) {
  // Handle the exception
 } catch(...) {
  // Handle any other exception
 }
```

Exceptions



https://www.softwaretestinghelp.com/exception-handling-cpp/

Exceptions

```cpp
int main() {
  std::vector vec = {1, 2, 3};
  try {
    int value = vec.at(10);
  } catch (const std::out_of_range& e) {
    std::cerr << "Out of range error: ";
    std::cerr << e.what() << '\n';
  }
}
```

- See `std::vector::at` doc

Exceptions

```cpp
#include <vector>
#include <iostream>
int main() {
  std::vector vec = {1, 2, 3};
  try {
    int value = vec.at(10); // This will throw
  } catch (std::exception& e) {
    std::cerr << "Out of range error: ";
    std::cerr << e.what() << '\n';
  }
}
```

- Exceptions are derived from `std::exception`

Exceptions

```cpp
#include <vector>
#include <iostream>
int main() {
  std::vector vec = {1, 2, 3};
  try {
    int value = vec.at(10); // This will throw
  } catch (...) {
    std::cerr << "An error occurred\n";
  }
}
```

- ... catches all exceptions

Exceptions

```cpp
#include <vector>
#include <iostream>
int main() {
  std::vector vec = {1, 2, 3};
  try {
    int value = vec.at(10); // This will throw
  } catch (std::exception& e) {
    std::cerr << "Out of range error: ";
    std::cerr << e.what() << '\n';
  } catch (...) {
    std::cerr << "An unkown error occurred\n";
  }
}
```

Exceptions

```cpp
#include <vector>
#include <iostream>
int main() {
  try {
    // May run out of mem
    std::vector<int> large_vector(1'000'000'000);
  } catch (const std::bad_alloc& e) {
    std::cerr << "Memory allocation failed: ";
    std::cerr << e.what() << '\n';
    throw; // Propagate
  }
}
```

Exceptions

```
try {
  int number = std::stoi("1234551212312 41");
 } catch (std::invalid_argument& e) {
  std::cerr << "Invalid argument\n";
 } catch (std::out_of_range& e) {
  std::cerr << "Out of range error\n";
 }
```

Exceptions

```
try {
  int number = std::stoi("abc");
 } catch (std::invalid_argument& e) {
  std::cerr << "Invalid argument\n";
 } catch (std::out_of_range& e) {
  std::cerr << "Out of range error\n";
 }
```

Define errors out of existence

```cpp
// A function that only works for positive
 ↪  numbers
void f(int i) {
  if (i < 0)
    throw std::runtime_error("f(): negative
     ↪  argument");
  // ...
}
```

- More code → More chances to mess up

Define errors out of existence

```
// A function that only works for positive
 ↪  numbers
void f(unsigned int i) noexcept {
  // ...
}
```

- More code → More chances to mess up

Define errors out of existence

```
void f(const char* p) {
  if (!p)
    throw std::runtime_error("f(): null
     ↪  pointer");
  // ...
}
```

- More code → More chances to mess up

Define errors out of existence

```
void f(std::string p) noexcept {
  //for(char c: p) { /* ... */ }
  // ...
}
```

- More code → More chances to mess up

Assertions

```
#include <cassert>
void f(int i) {
  assert(i >= 0);
  // ...
}
```

- Disabled if compiled with -DNDEBUG
- CMake: `-DCMAKE_BUILD_TYPE=Release`

Static Assertions

```
static_assert(sizeof(int) == 4,
              "int is not 4 bytes");
```

- Compile time check

Static Assertions

```
template <typename T>
void f(T t) {
  static_assert(std::is_integral<T>::value,
                "T must be integral");
  // ...
}
```

- Useful for templates

C++20 Concepts

```
template <std::integral T>
void f(T t) {
  // ...
}
```

- Cleaner syntax

C++20 Concepts

```
void f(std::integral auto t) {
   // ...
}
```

- Even cleaner syntax