A Study on the Effects of Exception Usage in Open-Source C++ Systems

by

Kirsten Celeste Paquette Bradley

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Exception handling (EH) is a feature common to many modern programming languages, including C++, Java, and Python, that allows error handling in client code to be performed in a way that is both systematic and largely detached from the implementation of the main functionality. However, C++ developers sometimes choose not to use EH, as they feel that its use increases complexity of the resulting code: new control flow paths are added to the code, "stack unwinding" adds extra responsibilities for the developer to worry about, and EH arguably detracts from the modular design of the system. In this thesis, we perform an exploratory empirical study of the effects of exceptions usage in 2721 open source C++ systems taken from GitHub. We observed that the number of edges in an augmented call graph increases, on average, by 22% when edges for exception flow are added to a graph. Additionally, about 8 out of 9 functions that may propagate a throw from another function. These results suggest that, in practice, the use of C++ EH can add complexity to the design of the system that developers must strive to be aware of.

Acknowledgements

I'd there is one thing I have learned while completing this thesis, aside from what is presented in this document, it is that "it is dangerous to go alone". I honestly would not have finished this thesis had it not been for the people in my life who support me. I will attempt to thank everyone here, but I feel I could double the page count if I tried to fit in everyone who deserves to be here.

Foremost, I would like to thank my supervisor, Mike Godfrey, for guiding me through this process while also giving me the space and time I needed to do things at my own pace. You're a great supervisor, a master of dad jokes, and I am glad you gave me the chance to be your student.

To Mei Nagappan and Patrick Lam, thank you for being readers for my thesis and giving me valuable feedback to enhance this final versions.

Thank you to my parents, Sandi and Richard, for not getting too annoyed at the fact that I was so wrapped up in school at times that I would seem to go forever without communication. Thanks for all the cute pictures of cats and dogs. Thank you Kyle and Natasha for putting up with my nerdiness for as long as I can remember. Thank you Serenity and Billy for being thinking I'm a cool aunt.

I am eternally grateful for the existence coffee that is black as midnight one a moonless night. I couldn't have done it without you.

There are many groups of people I have left to thank for the various roles you have played in my life.

- The Nerd Herd (Beth, Carolyn, Devin, Lindsay): you were the first group of peers who understood me and helped me learn about myself. We may not talk much these days due to life going separate ways, but we are there for the important things.
- Grovites (Katie, Melissa, Robin, Sandra, Tannis): living at the Grove was one of the first times I really felt like I belonged somewhere in university. We had many great adventures and I hope we have many more. May there always be a couple extra bananas and canoes in your lives.
- Teaching Option (Alex, Bre, Chelsea, Dylan, Grace, Joel, Katherine, Kendra, Niki, Sarah): you are a great group of people who made me part of their family. With such diverse backgrounds and interests, it's amazing we never won at trivia, but we always got question 13.

- ISAs (Aabraham, Chantelle, Dhron, Ed, Kaleb, Kevin, Leonard, Max, Marie, Rob, Sana, Sean): it's been Marvel-ous to work, watch movies, play games, and eat food with all of you. I promise I'm not a spy.

- ISCs: A group of people that saw so much potential in me and continued to hire me for just about every odd job they needed to get done. Without the support from you, I would not have been able to become a Master's student. Thank you for giving me a chance.

- Lecturers (Jeff, Mark, Nomair): one of the most supportive group of people I've ever met. You all let me talk to you far more than you ought to have and always gave me confidence when I needed it most. You believe in me more than I ever have and I have truly enjoyed every coffee and adventure we've had together.

- Labmates (Aaron, Achyudh, Arman, Ashwin, Bushra, Cassiano, Cosmos, Davood, Gema, Jeremy, Magnus, Reza, Laksh, Sunjay): you have made my time in the lab fun. I am sorry that I was the catalyst for so many shenanigans and I'm sure now that I'm gone, there will be much more work that gets done.

- CS Department: I've meet many graduate students, lecturers, instructors, and professors over the several years I've been involved here. While I could almost never get to where I wanted to get to on time due to someone stopping me to ask how things are going (or the dreaded question of "how's the thesis?"), you've made me feel like I belong here.

- My Girl Guides (Amy, Ailbhe, Beatrix, Bella, Cassi, Charlotte Original, Charlotte 2.0, Eve, Fiona, Hanh, Hannah Bea, Gabbi, Isabel, Louise, Maya, Meredith, Morgana, Olivia B, Olivia Q, Parnia, Reta): while I've been your leader, you hooligans have taught me so much. It's been amazing to watch you grow and get to know you all. I look forward to continuing to camp and be there over the next few years. You each have so much personality and will be great leaders one day. Thank you for putting up with how absent minded I've been while finishing my thesis and giving me a break from the rest of my hectic life. To your parents, thank you for sending your girls on adventures with us. Most importantly, thank you Heather and Glory for all the extra work you've put in this year while I've been finishing my thesis. You've been great to lead these girls with and are great leaders!

Finally, I'd like to thank my best friends.

Rob, while I didn't trust you in that first Resistance game, I (mostly) trust you now. While we're tired and have so much work to do, I am glad we find time to chat, play games, and drink coffee.

Brad, since the first time I was an ISA, you've been a consistent influence in my life. Sometimes good and sometimes bad. While you don't remember it, you taught me C++ which has been kind of crucial to my research. I think I can now safely say that I'm done at least 10% of my thesis.

Kristina, you're a wonderful person. From baking cookies, to joyous Mario Kart victories, to late nights talking about nothing in particular, all of our shenanigans have been awesome. I don't think I've ever meet a person I can relate to as much as you. The influence you've had on my life isn't expressible in words.

Thank you everyone for making everyday exceptional.

Dedication

In memory of Earl Paquette. Grandpa, Bumpa, and Good Guy.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Robust software must be able to recover from a variety of unforeseen error conditions that may arise during run-time. Languages such as C++, Java, and Python provide a flexible language feature for this known as *exception handling* (EH), where functions can interrupt normal execution to allow special error handlers to run and, if possible, recover from the error condition. A key feature of EH is that errors are often handled in a part of the code that is removed from where the problem is detected; this allows developers to place error handling code in key spots of the design, rather than insisting that errors be handled when and where they are first detected.

However, using EH has both costs and risks. For example, EH may degrade performance significantly; developers sometimes believe EH makes code harder to understand [1][29] and debug [24]; "stack unwinding" adds extra responsibilities to developers to ensure resources are not lost if an exception if thrown; and if an exception is thrown but not caught, the whole program will abort. Google, for example, recommends in its C++ style guide that EH not be used at all, although this is largely because of the redesign costs that would be incurred if EH were to be added to their existing products [5].

Despite these beliefs, many modern programming languages have EH mechanisms and there are known benefits to exception use such as improving error handling across modules [1]. While there may be some inherent difficulty with exception code, this difficulty could come from the tasks for which exceptions are used [32].

Most previous exception studies have looked mainly at Java systems, so there is a dearth of empirical results about the use of EH in C++ code. While the basic EH approach is similar in Java and C++, there are also important differences. For example, Java supports checked exceptions, which means functions require exception specifications if they throw exceptions, while C++ has deprecated the analogous feature. C++ requires careful consideration of how “stack unwinding” may affect resource management and memory leaks, while Java's use of Garbage Collection largely obviates this need. Finally, C++ allows programmers to throw any kind of object, not just those that inherit from a master exception class.

Because of the extra concerns that C++ EH adds and given the relative absence of studies on EH in C++ systems, we decided to perform an empirical study of how EH affects C++ code. To facilitate this study, we implemented a tool that can extract expression flow through call graphs, based on LLVM tooling infrastructure; we could not find any existing tools that could perform this kind of analysis and produce textual output for further processing. Thus, a secondary contribution of our work is the development of this tool, called Zelda, which we will released as open source.

1.1 Thesis Contributions

This thesis has two main contributions:

- We present an exploratory empirical study on understanding EH usage in real-world C++ code taken from GitHub, and we evaluate the potential impact of EH usage that may not be immediately obvious from reading the code; this study helps to shed light on perceived versus observed issues in using EH in C++ systems.
- We present a static analysis tool — Zelda, which is based on LLVM tooling — that can be used to extract information about EH use in C++ code. Information about call graphs and exception flow are presented in a text format to allow analysis of projects without the need of a visual interface. This allows for many projects to be analyzed and studied, and also for further kinds of analysis to be performed using the output.

1.2 Thesis Organization

This thesis is organized into five parts. Chapter 2 is a summary of related work and C++ features related to exception. This summary starts with the proposal of EH mechanisms in programming languages leading to the first programming language to have an exception mechanism. This is followed by the studies of exceptions in C++ and other languages and a summary of analysis tools that influenced our tool.

Section 2.2 provides background information on the aspects of C++ that are related to EH. This includes a brief summary of C++ features that are not specific to exceptions, but are related to them, and C++ exception handling mechanisms (EHM).

Section 2.3 discusses views and opinions on EH from previous research and a software company. This discussion highlights some of the perceived advantages and disadvantages to using exceptions in a project. Our research questions reflect the discussion and hope to provide meaningful results to the discussion about exception usage.

Chapter 3 explains the development of our static analysis tool and the tasks that it is currently capable of completing. Our final algorithm for exception propagation is given in detail. This chapter also explains exception graphs and throw length, both of which are concepts that we use in the analysis of our research questions.

Chapter 4 contains the results of the exploratory empirical study. Each of the research questions is addressed with an empirical study on an aspect of EH code.

Chapter 5 summarizes the contributions of this thesis, the limitations to this research, and describes future work that could follow.

Chapter 2

Background and related research

2.1 Related Research

2.1.1 History of Exception Research

Prior to the implementation of EHM in programming languages, there was theoretical research discussing what would be important for such features. This section focuses on research that introduced mechanisms that are similar to the approach that C++ uses.

Goodenough was one of the first researchers to discuss adding exception handling to a programming language and developed a formal notation to express exceptions and their semantics in a language [4]. He proposed that the caller of the function should be able to handle the error while the function itself should not be able to recover. The idea that exceptions should be interprocedural is still encouraged today [32]. He also discusses situations that exceptions should be used for, such as domain and range failure, and how exceptions should be implemented, such as error codes. Finally, he talks about the potential control flow of exceptions being either an escape from a procedure or to notify the user of an event occurring so the user can respond and the process can continue.

Levin proposed properties that make an EHM good, such as verifiablity and practicality, and proposed a mechanism with these properties [14]. His mechanism involved raising an exception from within a function to be handled in a calling function based on the type of the raised exception, similar to the mechanism that is used in modern C++. Different from C++, the proposed mechanism included resumption of the code where an exception was raised. Additionally, exception handling was to be performed by a handler procedure being called, as opposed to a local block of code.

PL/I and CLU were the first programming languages to have EHM implemented and was critiqued by researchers at the time. The existence of exceptions in a programming language were first researched by Maclaren by reviewing the EMH in PL/Il[16]. This work was a critique of the mechanisms which existed. The PL/I critique addressed the release of resources and changes in program state after an exception has occurred. Liskov et al. explained the EHM which they implemented in CLU and they discuss decisions which language designers should consider when developing an EHM [15].

2.1.2 Exception Usage

Previous research has addressed several topics related to the use of EH, particularly in Java and C++ systems. This includes empirical studies of how EH code is written, how EH code affects defect in systems, as well as studies of how developers perceive the benefits and drawbacks of using EH.

Shah et al. explored the question of why developers often choose not to use EH at all [30]. They interviewed Java and C++ developers about the topic, and also developed and validated a tool for visualizing EH usage. This work did not involve the analysis of existing code for EH usage.

Xie et al. studied how developers handle exceptions once they have occurred[33]. They observed that EH is often implemented to handle a conditional case, such as a file not existing, and to handle external cases, for example checking the results from a call to an API function. Their work categorizes these situations theoretically without examining real-world code or consulting developers.

Marinescu studied versions of Eclipse and showed that code that uses exceptions is more defect-prone than other code[17]. They used their tool iPlasma to inspect code at a class level to determine if there are functions in a class that throw and catch exceptions[18].

2.1.3 Exception handling in C++ systems

Bonifacio et al. studied C++ exception usage in detail [1]. They analyzed 65 C++ projects and surveyed C++ developers with a range of experience. They investigated the most common types of objects caught, what percentage of the code base is dedicated to EH, and determining what types of actions are performed in `catch` blocks. The survey included questions about whether developers believe C++ programmers avoid exception usage and why it may be avoided. Developers' responses to their survey influenced our discussion on exceptions.

Schilling created a compiler that can optimise the usage of exceptions in C++ code[27]. This work is of interest because EH relies on the types of objects at run-time which is slow to determine in C++. While the goals of their work are different from ours, they present algorithms to determine whether functions can throw exceptions based on condition analysis. Their algorithms are more detailed than our analysis as they look at each expression and determine if it can throw an exception while we concentrate on analyzing throw statements and propagation.

Hasu argued that whether libraries use exceptions or another form of error handling in C++ (and C) should be decided by the user of the library [7]. For instance, the developer might want to have legacy code from C libraries use exceptions.

De Dinechin discussed the aspects of the assembly language for the IA-64 which must be altered to accommodate C++ EH [3]. The optimization of code cannot occur in some situations due to the possibility of a function throwing an exception and additional information which must be maintained to ensure destructors are called as the stack unwinds. Both these situations makes the final machine code more complex.

Prabhu et al. proposed an algorithm for interprocedural exception analysis for C++ [24]. This is the only other work we are aware of that explores call graphs combined with exceptions for C++ programs. Their goal was to translate C++ code with exceptions into an equivalent exception-free C++ program. The algorithm presented in [24] influenced the design of the static analysis tool that we have implemented.

2.1.4 Exception handling in other languages

Kery et al. examined EH in Java [12]. Their work used about 8,000,000 Java repositories on GitHub. They determined that superclasses, such as `Exception` and `IOException`, are the most common catch types in Java. Additionally, they categorized statements in `catch` blocks to determine what actions are taken to recover from exceptions, and found that rethrowing and terminating are the most common responses.

Nakshatri et al. analyzed patterns of EH in Java systems [22]. Using a corpus of 7.8 million Java projects from GitHub, they analyzed the types of exceptions and the bodies of `catch` statements for patterns for exception recovery. While they listed three ways to analyze the types of exceptions, they stated that “an exception thrown [...] will eventually be caught by a caller method using a `try-catch` block”. While this may be true in Java due to compilation failure if checked exceptions are not handled or specified for a function, there is no similar requirement in C++.

Other research has addressed concerns about the robustness of EH. For example, Kechagia et al. studied context-dependent exceptions in Java code [11], while Oliveira et al. explored EH in Android and Java applications [23]. Robillard et al. performed static analysis of exceptions and their flow in Java [25][26]. They introduced the notions of breadth and depth of exceptions as meaningful ways to reason about global exception flow; they also included two case studies and discussed how the code would be improved by reducing both exception depth and breadth.

A concept used in the analysis in this thesis is augmenting a call graph to include exception flow. Previous work by Choi et al. and Sinha et al. suggested augmentation of Java flow graphs to represent exception usage [2][31]. Sinha proposed a control flow graph that condensed potential exception instructions to have less complicated graphs. Choi augmented the call graph by adding edges and nodes that represent exceptions at the function level that could be used to create a graph for interprocedural flow. Both performed an empirical study on their proposed graphs with Choi examining how a typical flow graph is affected by their augmentation and Sinha examining the change in graph size.

2.1.5 Static Analysis Tools

To ensure accuracy of the call graphs we produced, we compared our results to the commercial tool Understand by Scitools [28]. Understand is a static analysis tool that supports several tasks to understand code, such as metric calculation, dependency analysis, and control graphs, in languages including C++, Java, and Python. While the tool outputs detailed call graphs, the graphs do not consider exception flow. Additionally, call and flow graphs are produced as visuals with no easily interpreted textual option available.

LLVM is a compiler framework designed to support dynamic and static analysis designed by Lattner et al.[13]. LLVM consists of sub-projects, such as Clang which is a C and C++ compiler, that are available as open-source code. LLVM Libtooling API is a library designed for writing tools based on Clang. Some tools the library allow developers to specialize parsers, code generation, and Abstract Syntax Tree walkers.

The static analysis tool Rex — developed by Muscedere [20] — served as a foundation for using LLVM's Libtooling API. Rex is a fact extraction tool for detecting hotspots for feature interaction in C++ code. This is accomplished by walking the Abstract Syntax Tree (AST) of a program and recording information about how variables are used. The appendices provide detailed instructions about what is needed to use Libtooling and a process for downloading the required systems. The source code for Rex is publicly available and it was used as a guide and foundation for the construction of our static analysis tool[21].

2.2 C++ Features

This section is a brief summary of features in C++ that are related to EH, followed by the specifics of EH mechanisms in C++. Appendix A contains a code example that contains the aspects of C++ discussed in this section with an explaination of the state of memory while the program executes.

2.2.1 Classes

Classes are a central feature in an object-oriented programming languages like C++. A class is defined as a collection of fields and methods that operate over the internal variables of those fields. An individual instance of a class is called an object. A field is a variable that is associated with each object created. A method is a function that is called on an object. The term member can be used to refer to either a field or a method. An object is created by calling a method called a constructor and it is deleted by a destructor.

Inheritance

Inheritance means that all of the members of one class are members for another class. The class that contains the members of another class is said to inherit from the other class. The class that is inherited from is a superclass and the inheriting class is a subclass. The subclass can also have additional members and override methods to perform differently on a subclass object. A subclass object can be used in place of a superclass object in any context.

A class can have multiple superclasses and subclasses at the same time. A hierarchy is a directed graph where each node represents a class, and edges indicate inheritance. A class A is a subclass of a superclass B if A inherits from B or A inherits from a class that is a subclass of B. Being a recursive definition, a subclass can have an arbitrary number of inheritances between itself and a superclass.

2.2.2 Resource Management

C++ does not have a garbage collector to reclaim heap allocated-memory or other resources that are no longer needed by the executing program. C++ uses two memory pools to store data during execution.

Stack Memory

The *stack* is the default location that a program stores data while it is executing. Stack memory is associated with a block of code, called a scope, such as the body of a function, `if` statement, and loop. When the code associated with the scope is complete, the associated block of memory *goes out of scope*. If an object is allocated the stack, the destructor will be called when the object goes out of scope.

Heap Memory

For data to be persistent outside of the scope it was allocated, the data must be stored on the *heap*. Heap-allocated memory will not be freed when it goes out of scope. Memory is allocated from the heap using a `new` statement, which calls a constructor for the object. Heap-allocated memory should be freed using a `delete` statement, which calls the destructor for the object. While all classes have a trivial built-in destructor, a developer can implement their own destructor to release any resources that the object possesses.

Resource Acquisition Is Initialization

Resource Acquisition Is Initialization (RAII) is an idiom that is meant to ensure that all resources are returned when they are no longer needed. RAII promotes that all resources an object needs should be acquired when the object is created and returned when an object is destroyed. A simple way to ensure both happen is for all resources to be acquired within a constructor and to be freed in the destructor.

However, while a constructor is always executed to create an instance of an object, the destructor is not always implicitly executed. In particular, if an object is heap-allocated, the destructor of the object must be explicitly called to free the object and its resources. Fortunately, the destructor will always be called for stack-allocated objects when the block they exist in goes out of scope. RAII states that all heap-allocated data should be owned in an object on the stack, and the owner's destructor will free the heap-allocated when the object goes out of scope. This ensures that all resources will implicitly be returned when they go out of scope.

2.2.3 Run-time Type Information

Run-time type information (RTTI) is a feature in C++ can determine that makes available the types objects at runtime. While C++ is a statically-typed language, the type of an object is not immediately obvious in all situations. In particular, a superclass pointer or reference can refer to an instance of any of its subclasses. RTTI is used in exception handling to determine if exceptions match catch types. However, RTTI is slow as it involves traversing the entire class hierarchy to determine if an object is of a particular type.

2.2.4 Exceptions

Exception handling mechanisms vary between programming languages but typically are used as a way to interrupt control flow when an event — an exceptional event! — occurs that makes continued normal control flow impossible. For example, reading from a file that does not exist or accessing an array element that is out of bounds are actions that simply cannot be performed; EH allows the program designer to consider what to do in these situations.

The rules about which objects can be thrown and how thrown objects must be handled differ between programming languages. There are four components to exception handling in C++: throw-able data, code that throws and handles exceptions, exception specifications declared by functions, and stack unwinding.

Types of Exceptions

In C++, all objects and primitive instances can be thrown, while Java permits only classes that inherit from `Throwable` to be thrown. C++ provides several pre-defined exception classes, similar to Java's `Exception` hierarchy, all of which descend from the minimal `std::exception` class, presented in Figure 2.1. It is common for standard library utilities to throw these exceptions when invalid input is received.

Since any object can be thrown, developers can create their own exception classes. These classes can inherit from a standard exception, their own hierarchy or class, or can be an independent class. According to Stroustrup, the intermediate exception classes should be inherited from for different purposes [32]. For example, `std::logic_error` is intended to be used when the error could be caught before executing the code, such as an invalid input or division by zero.

Exception Handling Code

There are three language features specific to exceptions: `throw` expressions, `try` statements, and `catch` statements. Figure 2.2 shows a code example that uses these features.

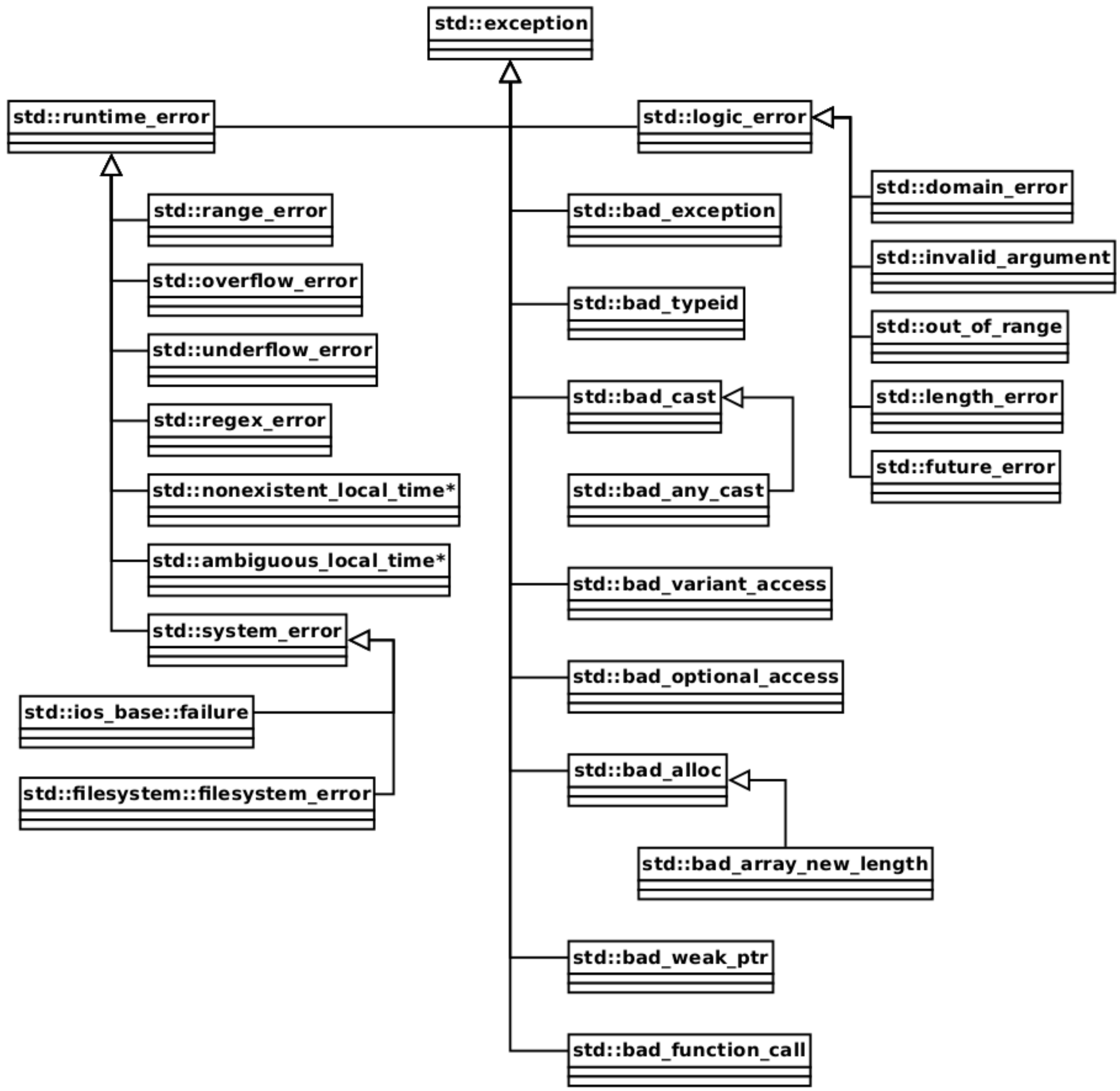

Figure 2.1: Inheritance Hierarchy of Standard Exceptions. Classes marked with an asterisk(*) will be introduced in C++20.

A `throw` expression is used to activate exception handling. The exception causes the normal execution to stop. The runtime system travels through the call stack, popping stack frames for pending function calls, until it finds a catch statement in a calling function that is willing to catch the exception. If the main function throws an exception from its function body, or two exceptions are active at the same time, the program terminates.

A `try` statement surrounds a block of code that typically could throw an exception directly, or containing a function calls that throws. A `catch` statement follows a `try` statement and surrounds a block of code that is to be run if an exception occurs. If an exception from the `try` block matches the type of the `catch` statement, the associated code runs and the program continues running after the last `catch` statement associated with the `try` block of the used handler. A `catch` statement can catch a type if it is of the type specified or if RTTI can identify the thrown object as a subclass of the specified type. A `catch` statement can indicate that it is willing to catch all possible exceptions by using an ellipsis instead of a type. However, if a `catch` statement catches with an ellipsis, the exception and any information it might hold, such as an error message, cannot be accessed. Within any `catch` statement, the caught exception can be rethrown by using a throw statement without specifying the exception.

Function Exception Declaration

In C++ there are two language features that can be used to indicate that a function may throw an exception. A `throw` clause lists all the types of exceptions that might be thrown from a function. However, `throw` clauses are deprecated as of C++11 and removed as of C++17, partially due to relying on the C++ RTTI mechanism which is slow. Declaring whether functions can throw exceptions is preferred to be done with `noexcept(expr)` where `expr` evaluates to a boolean at compile time. If `expr` evaluates to `true`, then the function will not throw exceptions. Otherwise, the function can throw exceptions. If a function violates an exception specification, the program will abort.

Stack Unwinding

An exception is thrown from the context of some scope, which could be the body of a conditional statement, function, `try`, `catch`, etc. As the exception propagates, it may travel through multiple scopes. When an exception leaves a scope, the variables contained in the scope goes out of scope. The process of blocks on the stack going out of scope while an exception is propagating is called *stack unwinding*. When a block goes out of scope, whether due to stack unwinding or the scope being finished, the destructor will be called for

```
int** matrix(int size) noexcept(false) {
   int** n = nullptr;
   try {
      if ( size < 1 ) throw std::out_of_range;
      // calls to new may throw std::bad_alloc
      n = new int*[size];
      for( int i = 0; i < size; ++i ){
         n[i] = nullptr;
      }
      for( int i = 0; i < size; ++i ){
         n[i] = new int[size];
      }
   } catch (std::bad_alloc& ba) {
      if ( n != nullptr ){
         for (int i = 0; i < size; ++i){
            delete[] n[i];
         }
         delete[] n;
      }
      throw;
   }
   return n;
}
```

Figure 2.2: Code example that shows aspects of exception handling written in C++.

each stack-based object and will not be called for heap-based objects. The difference with stack unwinding and a block of code going out of scope is that the remainder of the code in the associated scope will not be executed. A possible implication of not executing code is that manual resource management might not occur. Furthermore, due to destructors being called during stack unwinding and the inability for multiple exceptions to occur at once, destructors should not throw exceptions. If adhering to RAII, all resources will be released while the stack is unwinding because nothing is manually managed.

2.3 Exception Discussion

A widely-held belief that has been discussed by previous researchers — but notably without empirical evidence — is that the use of exceptions makes code more complicated. This section summarizes what various researchers, individuals, and companies have stated about how exceptions might affect design clarity.

As stated by Robillard, EH can improve error recovery across modules [1]. Its use allows the developer to be notified of incorrect usage of a module and to specify recovery actions at a location in the system's design that the developer feels is most appropriate (i.e., not necessarily where the error is detected). The use of EH can also decrease the amount of error checking required when returning from a function, as the error can result in an exception being thrown and the appropriate handler invoked [5]. However, developers from another survey expressed that the flow of exceptions had a negative impact on modularity [1].

An important task while programming is knowing the state of the program during a function call. This requires a developer to be aware of the control flow of the code, including the expectations from any called functions. In a survey of developers, respondents stated that exception flow and handlers add new control flow paths to the code, and may run counter to their natural intuition [1]. This suggests that exceptions may have an impact on the flow of a program that makes it more complicated to understand.

The use of exceptions may also impact the design style of the code [5]. For instance, if a programmer knows that exceptions will be thrown when code is misused, they may decide to forgo checking the validity of their input and to place their error handling code in `catch` statements. Furthermore, to account for stack unwinding when an exception occurs, either intermediate functions have to be prepared to catch all exceptions and free their resources before rethrowing, or code has to be written that strictly adheres to Resource Acquisition Is Initialization (RAII). Apart from RAII, there are other important best practices concerning EH that developers must be aware of, such as "destructors must never throw"[32].

How exceptions are used may also influence how they are perceived by developers. Exceptions are frequently used for error handling due to their ability to travel through the call stack to a location that can recover from the error. As reported by a survey participant, "error and EH (code) is hard"[1]. While it is possible that both are difficult to write and understand, Stroustrup has stated that "exceptions make the complexity of the error handling visible. However, exceptions are not the cause of complexity" [32]. It is possible that the perceived difficulty of exceptions comes from how they are used and not that they are used.

Some programming languages allow the developer to indicate the types of exceptions which a function can throw. For example, Java has checked exceptions that must caught or declared to be thrown from a function, or the program fails to compile. Java also has unchecked exceptions, which a function does not have to catch or declare it throws for the program to compile. This leads to exceptions that can travel through functions without handlers and acknowledgement. Research has shown that unchecked exceptions result in many program crashes in Android applications [11]. C++'s exceptions are equivalent to Java's unchecked exceptions. While a function can be specified as to whether it throws, it is up to the developer to label functions in this manner. Doing so does not force the caller to handle the exceptions at compile time like Java, but forces the program to terminate if an unspecified exception is thrown.

This discussion serves to motivate our exploration of EH practice in C++ systems. From previous research, we observed researchers and developers expressing concerns about exceptions and they stated how they believed exceptions may affect a system. However, there was no empirical evidence to support effects. Our goal was to study existing systems that use EHM and determine how exceptions affect various aspects of a system.

Our research questions were chosen based on concerns mentioned throughout previous research. We studied the localization of exceptions, the effects of exceptions on the control flow and implementation of a system, and the effects of exception specification on a system. These areas reflect previous concerns and distinct ways that the presence of exceptions may impact a system.

2.3.1 Research Questions

We have investigated four research questions:

RQ1 *How localized is exception throwing and catching?*

RQ2 *How does the use of exceptions impact the control flow of a program?*

RQ3 *How does the use of exceptions impact the implementation of a program?*

RQ4 *Do C++ exception specifications affect the outcome of exception handling efforts?*

We next describe our approach to answering these question.

Chapter 3

Methodology

3.1 Static Analysis Tools

To analyze the presence and usage of exceptions in a code base, a static analysis tool is needed. We could not find an existing tool that met our technical needs of performing C++ exception flow analysis and giving textual output. We thus created two static analysis tools, the second of which is used for our research.

3.2 Annie

The first tool we created was named Annie, short for Analyser. Annie read the AST that is produced from clang++ to rebuild a simplified AST containing the nodes necessary for exception analysis. Annie had two goals: determine whether exceptions are present in a code base, and track the flow of exceptions through the call graph. To determine whether exceptions are present, Annie found all nodes `throw`, `catch`, or `try` nodes and recorded their type and presence. This step was performed to ensure that exceptions were used frequently enough in the corpus to provide meaningful results.

The exception flow tracking involved walking the call graph to determine when functions were called and how their exceptions would flow. In this tool, the analysis started at a main function to find all functions that it called. This process was repeated recursively to ensure all called functions were analyzed. This approach was thorough, reflected the entire call graph, and was slow. It involved searching the entire call graph for exceptions, which was tedious for large programs. For these programs, large sections did not need to

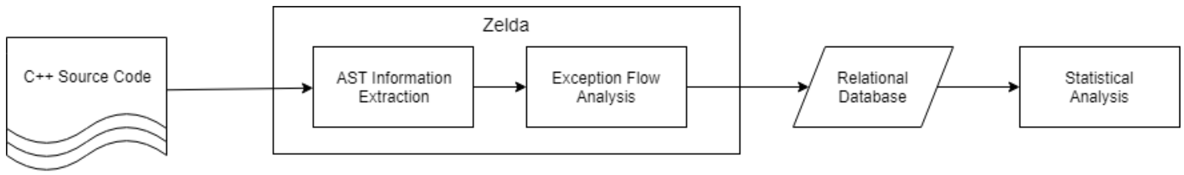

Figure 3.1: Data processing during analysis using Zelda.

be analysed as they had no exceptions. Additionally, the code for uncalled functions was never analysed which resulted in code without a main function, such as libraries, not being analyzed.

While this tool worked for basic analysis, we observed that larger programs were causing Annie to terminate while still processing data. This could have been due to the program using a lot of disk space to store AST files from clang, and tracking the many paths exceptions could flow involving a lot of memory to be used while the program was running. Additionally, from analyzing code, more nodes of interest were identified and the complexity of C++ ASTs were becoming more apparent and, despite our best efforts, the code was becoming unmanagable. Thus, we built a new tool to address these concerns.

3.3 Zee Exception Length and Destination Analyzer

The second tool we created was named Zelda — Zee[1] Exception Length and Destination Analyzer — based on the LLVM Libtooling infrastructure [13]. Libtooling has a tool for walking ASTs and stores detailed information about most statements and expressions for easy access. Zelda consists of a set of AST tree walkers that extract and output information about exception handling, including the flow of exceptions, and the basic classification of expressions. The capabilities and implementation details of Zelda are discussed further below.[2]

3.3.1 Data Extraction

The first task was to determine the presence of exception code. As discussed in Section 2.2.4, this involves the detection of `try` statements, `catch` statements, and `throw` expressions. Each of these has a specific type of AST node, making finding all instances of these

[1] Pronounced with an outrageous French accent [10].
[2] We will release Zelda as open source.

nodes a simple task in a walk of the AST. Furthermore, the declaration of the `catch` statement and expression of the `throw` expression are stored in the node, which simplifies determining the types.

The second task was to collect data about other aspects of the code in the corpus, specifically line counts and statement classification. The line counts of functions and `catch` statements were calculated by counting the number of newline characters present in the pretty print of the body from LLVM. Statement classification involved recording the types of statements present. AST nodes of interest were simple to detect due to having unique nodes in the structure, including `return`s, `break`s, `continue`s, `throw`s, and `delete`s. Further classification involved looking at some function calls from the C++ standard library whose use is known. For instance, calls to `operator<<` and `printf` were classified as prints.

The final task for Zelda was to extract and map exception flow through a system. This involves knowing where exceptions occur, the types of thrown exceptions and `catch` statements, and call information. All of this data can be determined from walking the AST; however, more information is needed than for a typical call graph. Specifically, knowing about the presence of a `throw` statement or a function call in a function is not enough information to understand exception flow. For this purpose, an augmented call graph, which we call a context graph, is used in this analysis. A context graph associates a call with the calling function and the context within the calling function. For our purposes, the context can be either a `try` statement, `catch` statement, or function. An example of a portion of a context graph is presented in Appendix 5.2.3.

While the first two tasks are relatively simple to implement, tracking exception flow through a program is more complicated. To ensure accuracy, the call graph has to be correct. Given the size of the corpus, it was infeasible to check the accuracy of the call graphs on a large sample of projects. The commercial tool Understand by SciTools[28] was used to compare the call graphs of nine projects that were randomly chosen from the corpus. Table 3.1 presents the projects used for comparison. The majority of functions in the call graph were reported by both tools, although both reported functions that the other did not detect. Each of the programs reported calls to library functions that the other did not report. Since we are not interested in exceptions thrown from libraries, this is not a concern. Additionally, Understand reported calls to parent constructors, destructors, overridden methods, and templated functions that Zelda did not report. Zelda does not report destructor calls because developers are discouraged from throwing exceptions from destructors due to a combination of a program terminating if two exceptions are active at the same time and the fact that destructors are implicitly called while the stack unwinds. The remaining calls that Zelda does not report may be important to know about and could

User	Project	Called(Zelda)	Matching
dermesser	libsocket	70	94.5%
Fat-Zer	tdeutils	43	97.7%
greg-hellings	sword	914	80.5%
gulp21	QeoDart	0	NA
licq-im	licq	10	100.0%
nireis	pferd	68	93.2%
Nocte-	rhea	81	100.0%
tfarina	idep	208	97.2%
thaddeusbort	cantranslator	33	94.0%

Table 3.1: Projects used to compare call graphs between Understand and Zelda. Matching represents the number of functions found by Zelda divided by the number of fucntions found by Understand.

be found by improving the tool.

Aside from comparing the call graphs from Zelda to the call graphs of Understand, the data extracted about exceptions and their flow was checked for accuracy. The exception data was checked first by writing small programs which used features during development. Once initial development was complete, the results of a few randomly selected projects was analyzed versus a manual traversal of exceptions. Any problems were addressed by making smaller cases that replicated the observed problem and were fixed.

3.3.2 Exception Propagation Algorithm

Once the call graph is determined, Zelda combines the context and exception information to determine the flow of exceptions through a program. This is accomplished by looking at the context of each `throw` statement with the following algorithm:

Find all ***`throw`*** *statements T that are not rethrows*
For each ***`throw`*** *statement t in T:*
    Find all context edges C for t
    For each context c in C:
        Remove c from C
        If c is a ***`catch`*** *and t is a* ***`throw`*** *in c*
            Add edge(context(c),t) to C
        If c is a ***`catch`*** *and c contains a rethrow*

Add edge(context(c),t) to C
If c is a **`catch`** *and c does not contain rethrow*
Continue
If c is a **`try`**
Find first **`catch`** c_1 *of c of type t*
If no such **`catch`** *exists*
Add edge(context(c),t) to C
else
Add edge(c_1*,t) to C*
If c is a function
Find all function calls to c
For each function f that calls c:
add edge(callContext(c,f),t) to C

Through these steps, all exceptions are traced to either the `catch` statements that catch them, or to the last function that throws them.

For our empirical study, the graphs and code information are output in the Tuple Attribute Language (TA) and queried using Grok [8][9]. Grok is a programming language designed at the University of Waterloo that performs relational algebra. Previous uses of Grok include analyzing factbases representing relationships from a parser, such as Rex [20]. Operations that are present in Grok, such as transitive closure, relational composition, and set operations, make it ideal for manipulating the information from Zelda.

3.4 Exception Graphs

Exceptions unwind the stack of an arbitrary number of function calls, which means the control flow through a program can be drastically changed based on an exception being thrown. However, in a typical call graph, the semantics of returning to the calling function is implied by the graph. Thus, call graphs do not account for exceptional behaviour and the flow is inaccurately expressed. However, due to exceptions travelling up the call stack, if the developer knows a function can throw an exception, it would not typically be difficult to trace the exception through the call graph. However, some additional information is needed in the call graph

Consider a flow graph that includes both calls between functions and returns from functions. Calling a function that does not throw an exception will result in the return to

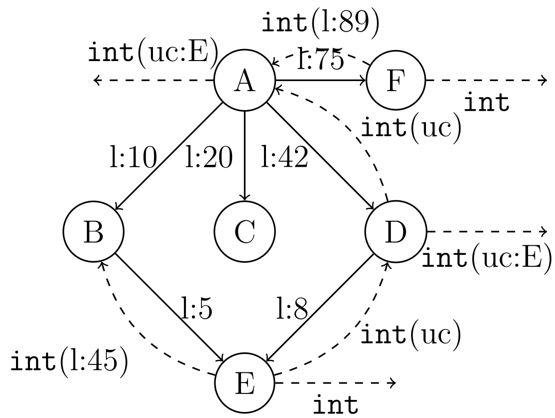

Figure 3.2: Exception graph example. Solid lines indicate a function call and the line the call is from is a label on the edge. Dashed lines indicate a thrown exception. Dashed lines that are not directed at a node indicates the function throws the exception.

the caller to be to the same location as the call to the function. Thus, return edges are not needed and are understood to implicitly be the reverse of call edges.

Consider an exception that is caught in the function that throws it. While this does activate stack unwinding, within a function unwinding the stack is effectively equivalent to breaking out of one or more nested conditionals. Additionally, since control flow remains in the function, the graph would not be altered outside of the throwing function.

Now consider an exception that is not caught in the throwing function. All calls to this function may result in an exception throw. To represent these potential throws an edge that does not have a destination is added from the function to indicate the function throws that type. If the throwing function is called, an edge annotated with the exception type and where it is caught are added from the throwing function to the calling function. The exception being caught will travel back to the call location and then unwind the stack to find a matching `catch` statement. While this is occurring, destructors will run for all stack allocated objects, which means that other function calls will be occurring that should be added to the graph. For simplicity, these function calls are not considered in our exception graphs. Eventually, the exception is caught and the control flow is at a different place in a calling function than where the function call occurred. Thus, this edge, while still returning to the calling function, would be separate from the call edge. Additionally, each exception could have a unique `catch` location that could be represented with a unique throw edge.

Any function that throws an exception results in an edge being added to the exception graph. Furthermore, these functions need to implement an exception handler or the exception will further propagate resulting in more edges being added to the graph.

Figure 3.2 depicts an exception graph for some arbitrary function. From `A`, there are calls to `B`, `C`, `D`, and `F`. There are calls to `E` from `B` and `D`. The functions `E` and `F` each throw an exception of type `int`. From `E`, the function `B` catches the exception on line 45, while `D` does not catch the exception and results in an exception edge being added to `D`, annotated with "uc" for uncaught followed by `E` to indicate where the exception originated. Finally, `A` catches the exception from `F` on line 89 and does not catch the exception from `E` and adds an additional exception edge from `A`.

Another exception graph is in Appendix 5.2.3, representing a code example.

3.5 Corpus and Data

The projects used for analysis are C++ projects that are publicly posted on GitHub. They have a main language of C++, are at least a year old, have had more than 100 commits in their lifetime, and have been committed to within the previous year. The projects were selected using a mirror of GHTorrent [6] from February 2017 with the code being the current versions from July 2017. There were 3,686 projects that matched these criteria. Removing instances of repeated projects, there was a total of 2,721 projects. The projects of the corpus and metrics are presented in Appendix B.

Various subsets of the corpus are used to address our research questions. These subsets are determined based on what aspects of code are being addressed. For example, projects that use a specific feature may be compared to projects that lack that feature, or features are checked between different aspects of the same project. If a specific subset is not specified, then the entire corpus was studied.

3.6 Exception Metrics

A metric called *depth* was defined by Robillard et al. to model the number of paths an exception could travel in a system [26]. Their metric inspired our metric, the *throw length* of an exception, which represents the number of unique functions that the exception will travel through before it is caught. This includes all functions through any path the exception could flow through. If an exception is rethrown, it is considered to be the same exception and further propagation increases the distance it travels, while an exception thrown from a `catch` statement is a unique exception.

The metric is influenced by McCabe's cyclomatic complexity [19]. The intuition behind cyclomatic complexity is that more potential paths through a program likely implies that the program is more complex. The existence of additional paths due to exceptions suggests that the program may be more complex which reflects the previously expressed concern from developers.

Similar to cyclomatic complexity, throw length is about the flow of the program. However, exception flow works in the reverse order of function calls. Additionally, cyclomatic complexity is calculated as the sum of the number of flows through each function, which means each function is considered individually. Throw length is calculated by investigating all the paths exceptions could take through a program. The order of called functions is crucial to this measurement unlike in cyclomatic complexity. While the order of function calls is important to the tracking the flow of exceptions, the metric is concerned with the number of functions visited and not the order of visitation. Using this metric, we can determine how many functions are affected by the introduction of exceptions.

3.7 Analysis Approach

In this section, we describe the approach used to analyze each of our research questions.

RQ1 *How localized is exception throwing and catching?*

To determine whether an exception is intermodule, the `throw` nodes in our context graph contains the file name of the `throw` statement and a boolean indicating the `throw` is not intermodule yet. As the exception propagates, each time the exception visits to a function, the file of the function and `throw` are compared. If these two files do not match, the `throw` is labeled as intermodule. Similarly, when an exception visits a `catch` node, the file of the `catch` and the `throw` are used to determine if there is an intermodule catch.

We analyzed these results to determine how many exceptions are intermodule and compare the catch rates between intermodule exceptions and non-intermodule exceptions.

RQ2 *How does the use of exceptions impact the control flow of a program?*

To evaluate the impact exception have on control flow, we consider the paths that an exception may travel during the execution of a program. We identified the number of functions that throw due to a `throw` statement contained within in it, as well as to having an uncaught exception from a function it called. We also consider the number of calls to each function that potentially throws an exception. These results are used to determine how many edges would be added to a call graph to make it an exception graph to evaluate how many control flow paths are added due to exceptions.

RQ3 *How does the use of exceptions impact the implementation of a program?*

During the analysis of programs, statements that dictate control flow and heap allocation are counted to determine the number of occurrences in each project. Whether these nodes occur in exception code is also tracked. This data is used to determine if these statements are used differently in projects that contain exceptions and whether exception code uses these statements differently. Whether statements are used at different frequencies between different types of code may give insight into how exceptions are typically used.

RQ4 *Do C++ exception specifications affect the outcome of exception handling efforts?*

We identified all functions that contain an exception specification as part of their signature. The presence of a specification is used to group functions and projects. The overall throw length of exceptions is compared between these groups to determine whether there is a correlation between exception specification and throw length. Whether there is a difference between throw length based on exception specification could reflect when exceptions specifications should be used.

In the next chapter, we discuss the results of these studies.

Chapter 4

Results

In this chapter, we discuss the results of our study.

4.1 Data Set Curation

We initially considered all projects in Github that were written mainly in C++, were at least a year old, had had more than 100 commits in their lifetime, and had been committed to within the previous year; this resulted in a preliminary set of 2721 distinct projects containing over 99 MLOC. We then ran these projects through our extractor to filter out those that did not use any EH features; that is, we discarded projects that did not contain at least one `catch`, `try`, or `throw` node in their AST. This left us with a set of 1182 projects comprising over 73 MLOC; Table 4.1 shows the total and median size of these systems, and the total and median number of `catch`, `try`, and `throw` nodes. The 1,539 projects that did not use exceptions contained about 22 MLOC with a median of n lines of code. This suggests that larger projects tend to use exceptions more often.

	Total	Median
Number of projects	1,182	-
Lines of code	73,309,259	8,299
`catch` statements	90,947	13
`try` statements	67,686	12
`throw` expressions	63,269	10

Table 4.1: Metrics of projects with exceptions.

While exception usage seems relatively common across the corpus, with an average of 53.5 throws per project with exceptions, the median number of throws is ten. This suggests that the majority of projects that use exceptions use them seldomly. In fact, there are n projects that have no `throw` expressions but have `catch` statements, and m projects that have no `catch` statements but have throw statements. A project that only catches exceptions implies that either the developer is attempting to ensure that no exceptions could be thrown without knowing if exceptions are possible, or the project uses third-party libraries that throw exceptions. Only having throw expressions suggests that either the program is meant to terminate if an exception is thrown, or the project could be a library.

4.2 Exception Localization

RQ1 *How localized is exception throwing and catching?*

Developers are encouraged to separate code into multiple files. A single file should contain only code which is related to a specific task. For brevity, we call all the code contained in a single file a *module*.

Exceptions are an encouraged way to express that an error has occurred between module. When used in this way, exceptions force the developer to be aware of and respond to improper use of code without having to check for return codes signifying an error after each call. An intermodule throw is a `throw` statement that unwinds the stack to or past a function that is part of a different module. If an exception is caught in at least one module it did not originate from, the `catch` is an *intermodule catch*. An exception is classified as being able to be caught if there is at least one catch statement that the exception could travel to that can catch the exception.

We found that intermodular exceptions occurred infrequently within the corpus. Of the 78,403 possible exceptions, only 9,241 (11.8%) are intermodular. This means that the vast majority of exceptions are thrown and caught within the same module. Furthermore, of the intermodule exceptions, 2,356 (25.5%) can potentially be an intermodule catch, while only 6,203 (9.9%) of exceptions that are not intermodular can be caught. Using a chi-squared test, the thrown exceptions were split by whether they were intermodule and whether they were caught. With a p-value of < 0.0001, we conclude that intermodularity and being caught are not independent variables with exceptions being caught more often if they are thrown intermodularly.

We conclude that intermodule exceptions are relatively uncommon, but still present. Considering intermodule error handling is seen as a benefit of exception handling, it is strange that it is uncommon for exceptions to travel between modules. This may suggest that exceptions are a relatively common way to handle errors within a module while other means are used to express errors outside of a module. Developers writing a module would have the freedom to use exceptions within their module while not forcing a developer using the module to interact with exceptions.

While intermodule exceptions are uncommon, they are about 2.5 times as likely to be caught than exceptions within a module. This suggests that developers do respond to exceptions from other modules more commonly than within a module. However, the fact that exceptions are seldomly caught within a module is a contradiction to the idea that exceptions may be used within module for error handling.

The result that it is uncommon for exceptions to travel between modules and that exceptions that stay within are module are uncaught is interesting. If these exceptions are left uncaught, they must eventually propagate to the `main` function and cause the program to crash. This suggests that many throwing functions are in the same file as a `main` function, or the function is not called from any functions that can be executed. If most throws are in the same file as a `main`, the program could be likely be better modularized. If most throwing functions are not called, it's possible that developers avoid these functions because they may throw exceptions.

This evidence suggests that most of the systems in our study do not handle exceptions well. The majority of exceptions remain uncaught regardless of if the exceptions travel intermodule. Further investigation could determine whether these exceptions could be executed or are dead code and give insight into whether developers fail to catch thrown exceptions, or avoid potentially throwing code. Considering thrown exceptions between modules, further analysis could be done to determine whether functions are called that perform the same task as throwing exceptions without the potential of a exception, such as `vector`'s methods `at` and `operator[]`.

4.3 Exception Flow

RQ2 *How does the use of exceptions impact the control flow of a program?*

A major concern about using exceptions is the potential for increasing the complexity of control flow throughout a program. Every exception begins with a `throw` statement. However, the path that an uncaught exception travels depends on the state of execution when the exception is thrown. The exception could be caught by any function on the current call stack. If a function does not catch an exception thrown by a function that it

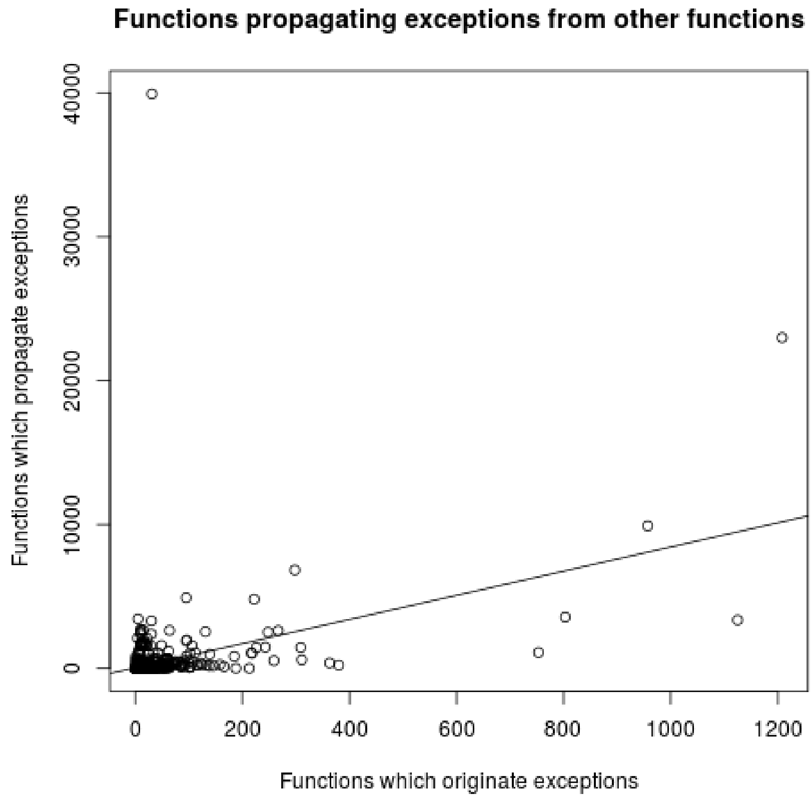

Figure 4.1: Functions which throw exceptions that originate from another function.

calls, it implicitly throws the exception as well. This could lead to functions that throw exceptions that do not have `throw` statements and are not obvious candidates for exception analysis.

While we did not address if exceptions add complexity to a program, we studied how the precense of exceptions could affect control flow. We studied this in two ways: first by, looking at functions that throw exceptions that originated in another function; and second, by examining edges that would be added to a flow graph to express paths that are due to exceptions.

	Direct	Indirect	Combined
Mean	22.9	237.9	260.8
Median	7.0	10.0	19.0
Standard Deviation	74.3	1491.4	1523.9
95th percentile	72.0	1100.2	1260.4

Table 4.2: Prevalence of functions that throw exceptions.

4.3.1 Throwing Functions

In general, we were curious about how many calls occur to throwing functions. We consider exceptions to be thrown directly if they originate from the function being considered. Functions throw indirectly if a function throws due to an exception travels to the function and is not handled. We addressed the following questions:

1. How many functions throw directly or indirectly?
2. How many throwing functions are called?
3. How many calls exist to throwing functions?

First, we inspected the number of throwing functions in the project. Table 4.2 presents the results. While most of the results vary drastically between directly and indirectly throwing functions, they have similar medians. Considering the median is 19 when combined, more than half of the projects have fewer than 20 throwing functions present. Furthermore, we compare the count of each within a project in Figure 4.1. From a linear regression with a p-value of < 0.0001, there are 8.39 functions that propagate an exception thrown from a function. This suggests that there are approximately 8 functions that indirectly throw an exception for every directly throwing function.

Knowing that there are many occurrences of throwing functions across the corpus, we wanted to know how many functions call throwing functions. Table 4.3 presents the results. When considering which functions call these functions, we are not concerned if the caller throws as well. Additionally, a function with multiple calls to throwing functions is counted as one function call. We observed that the median number of calls to directly throwing functions is higher than indirectly throwing calls, while the mean is significantly lower. This is interesting since there are generally more indirectly throwing functions as seen in Table 4.2.

Finally, we consider how many calls there are to each throwing function. Unlike the previous question, this count takes both caller and callee into account and represents the

	Calls to Direct	Calls to Indirect	Throwing Calls	All Calls
Mean	37.2	107.0	208.5	2417.0
Median	5.0	2.0	11.0	493.0
Standard Deviation	100.2	360.2	675.3	5370.3
95^{th} percentile	232.8	562.4	1270.6	1996.0

Table 4.3: Prevalence of calls to throwing functions. The first two columns represent the number of unique functions that call exceptional functions. The last two columns represent the total number of calls.

total number of calls. This considers all directly and indirectly throwing functions. There is an order of magnitude more calls to all functions than to throwing functions. The impact of the throwing function calls is discussed with respect to exception graphs.

4.3.2 Exception Graphs

Given the data involved in determining the information of throwing functions, how a flow graph would be altered due to exceptions can be expressed numerically. In particular, the number of affected nodes and added edges can be determined.

The number of exception edges that do not lead to another node is the number of directly and indirectly throwing functions. This is also the number of nodes that are out-edges for exception edges. The calls to directly and indirectly throwing functions is the number of nodes that are in-edges for exception edges. Also, the number of added edges is the the number of calls to any throwing function plus the number of throwing functions. The number of edges in the graph prior to the augmentation is the number of function calls and is used to normalize the edges.

Finally, we investigated how the augmented call graph is changed due to exceptions. The mean growth of the call graph is 22.1% with a median of 5.1%. From a linear regression with a p-value of < 0.0001, there is a slight positive correlation between the number of functions present and the number of exception edges added, with a slope of 0.060. Comparing the number of edges added to the graph to the number of directly throwing functions, a linear regression with a p-value of < 0.0001 shows a positive correlation with a slope of 14.2. Thus, we conclude that the number of edges added to the exception graph is linearly correlated to both the number of functions present and the number of throwing functions present.

We can conclude that it is common for the number of edges in a call graphs to grow linearly when exception edges are included. While the rate at which the graph grows it relatively small, this is concerning due to about 8 out of 9 of all throwing functions throw indirectly. This suggests that in the majority of instances, it is not immediately clear from the code that an exception could be thrown.

4.4 Implementation

RQ3 *How does the use of exceptions impact the implementation of a program?*

There are several ways that using exceptions in a project could result in a global change to the structure of code. The most obvious change is the presence of EHM. Aside from these mechanisms, the may choose to implement other aspects of their program differently, such as using instead of checking for invalid input before a function call, they may choose write a exception handler. Also, if exceptions are used, developers are encouraged to follow RAII suggests releasing of resources to occur in destructors, and if not, deletes are likely to occur in `catch` statements. To answer this question, we studied how the use of statements changes depending on the presence of exceptions.

Before looking at whether exceptions being used affects the implementation of a project, we investigated the prevalence of EH code. This includes both the number of projects that use exceptions and how much of a project is dedicated to EH. EH code was defined to be the code within `catch` blocks. The median percent of code in a project with exceptions that is dedicated to EH is 0.64% and a median across all projects of 0.02%. We compared this to the previous result from Bonifacio [1] of 0.03% of their corpus being dedicated to exception recovery. The amount of exception handling code is consistent between the two studies.

4.4.1 Statement Usage

Comparing across projects in the corpus, we studied whether there was a difference in control statement usage based on whether any exceptions are present. The measure used is the total number of occurrences of each statement normalized by the number of lines in the project. When comparing exception code within a project to other code in the project, both are normalized by the number of lines of code present for the respective category.

The results in Table 4.4 compare the usage of statements between projects that use exceptions and those that do not. The difference in control statements used is statistically significant using a Wilcoxon rank sum test for all the statements that were studied. This

Statement	exception	non-exception	p-value
`break`	0.0054	0.0060	< 0.0001
`continue`	0.0008	0.0008	< 0.0001
`delete`	0.0007	0.0008	< 0.0001
`loops`	0.0059	0.0344	< 0.0001
`if`	0.0417	0.0430	< 0.0001
`return`	0.0270	0.0283	< 0.0001
`switch`	0.0083	0.0077	< 0.0001

Table 4.4: Occurrences of control statements used in projects with and without exceptions normalized by number of lines. The p-values are calculated from Wilcoxon rank sum tests.

Category	NEH	`try`	`catch`	NEH vs. `catch`	NEH vs. `try`	`try` vs. `catch`
`break`	0.0051	0.0094	0.0018	< 0.0001	< 0.0001	< 0.0001
`continue`	0.0008	0.0012	0.0016	< 0.0001	< 0.0001	< 0.0001
`delete`	0.0007	0.0031	0.0046	< 0.0001	< 0.0001	0.0072
`loops`	0.0119	0.0235	0.0106	< 0.0001	< 0.0001	< 0.0001
`if`	0.0400	0.1044	0.0161	0.0085	< 0.0001	< 0.0001
`return`	0.0268	0.0942	0.0815	< 0.0001	0.0001	< 0.0001
`switch`	0.0081	0.0094	0.0006	< 0.0001	< 0.0001	< 0.0001
`throw`	0.0007	0.0089	0.0233	< 0.0001	< 0.0001	< 0.0001

Table 4.5: Occurrences of control statements in different types of code blocks. The p-values are calculated from Wilcoxon rank sum test. NEH is short for non-exception handling code.

shows that the presence of exception usage has a significant global effect on the control structures used.

From these results, the statements with the largest difference of use are loops, which includes `do-while`, `for`, and `while` loops. Loops are significantly more common in systems that do not use exceptions. While we did not investigate why loops may be more common, our intuition is that loops may be used to continually prompt a user until valid input is given and similar error situations.

Studying within a project, the use of control structures is separated into occurring within `catch` statements, `try` statements, or neither, and is presented in Table 4.5. From the results, it is clear that these control structures contribute to the code in each category differently. This suggests that these three types of code are written differently which may reflect their purposes. There are several observations that can be made about these differences.

The first category of statements considered was conditional statements. In general, all conditional statements were less common in `catch` statements. This may suggest that once an exception is caught, recovery is the same regardless of where the exception originated. `break` statements are more prevalent in `try` statements than anywhere else which could be related to `try` statements having more loops. Finally, `if` statements occur most frequently in `try` statements and could reflect extra error checking before executing code that may throw an exception.

`delete`, `throw`, and `return` statements were the other statements considered. `delete` statements were considered due to the potential for `catch` statements to be used for clean-up of heap allocated variables and were found to be most prevalent in `try` and `catch` statements. This result is interesting due to RAII suggesting developers using exceptions should release the heap allocated memory that is not wrapped in a class. This may suggest that developers using exceptions tend to release resources manually before rethrowing. Additionally, `delete` statements occurring more frequently in both than in non-exception handling code may suggest that `catch` statements often repeat code that would be executed at the end of a `try`.

`throw` statements occurred most often in `catch` blocks. This suggests that once an exception is thrown, it is likely for further exceptions to be thrown. Alternatively, the exception could be rethrown which suggests that the function cleans up what it can and continues the propagation of the exception. The only instance that was statistically significant for the difference in `return` statements was between `catch` statements and non-exceptional code. This suggests that functions are often exited upon catching an exception.

We conclude that using exceptions is correlated with the overall structure of a program. Additionally, the structure of `try` and `catch` statements are distinct from other code in a program. Overall, how control flow is handled in `catch` statements is not similar to general code.

Noting that EH code is distinct from other code, it is possible that the difference in the used statements could be related to the tasks that EH commonly performs. For example, perhaps error handling code is inherently different from other code as suggested by Stroustrup who suggested that the perceived complexity of exceptions comes from the inherent complexity of error handling[32]. We saw that conditional statements were most common in `try` blocks. Considering cyclomatic complexity, this does suggest that `try` blocks are more complex. However, the exception handling occurs in `catch` blocks which had the least conditional statements. Thus, it is possible that the complexity is not due to exceptions and is actually caused by error handling as suggested.

4.5 Function Annotation

RQ4 *Do C++ exception specifications affect the outcome of exception handling efforts?*

Documentation can alleviate some of the requirement for developers to know which functions can throw exceptions. While a project may have style guides that dictate how exceptions should be documented, there are built-in features in C++ to annotate functions as discussed in Section 2.2.4. Both `throw` and `noexcept` document and enforce exception usage and we investigated them together and their change to exception behaviour is not considered. Within the context of this question, stating a function is documented indicates that one of these two features is present for the function.

The presence of these features was first considered across the whole corpus. There were 462 projects that throw exceptions and have at least one function marked with exception information. There were also 71 projects that did not use exceptions and had exception specifications present for some functions. It was unexpected to find exception specifications in projects that do not throw exceptions.

We are interested in whether there is a difference in exception distance depending on whether annotated functions are correlated with throw length. In particular, whether a function indicates exception usage is used to categorize projects. To determine if exceptions travel further if the throwing function where some has exception specifications and some did not were investigated, of which there were 66. The average throw length from a documented and undocumented function was 3.167 and 3.746 functions respectively. Comparing with a paired Wilcox rank sum test gives resulted in a p-value of 0.0605. Thus, we cannot conclude that documenting a function affects the number of functions an exception travels through.

We next considered whether function annotation existing within a project influences throw length. This is different from the first analysis as all projects that throw exceptions were categorized based on the presence of exceptions. The mean throw length was 9.6 functions if there were no exception specifications, and 23.1 functions if there was at least one function with exception specification, which a t-test show was a statistically significant difference in means with a p-value of less than 0.0001.

We hypothesized that annotating functions would decrease how far exceptions are thrown. The fact that exceptions travel similar lengths within a project, regardless of exception annotation, may suggest that the observed correlation between exception annotations and throw length may actually be due to a some other variable. For instance, a larger project may have more possible functions for an exception to travel through and annotating functions could quickly convey exception information. These reasons combined could explain the significantly higher average throw length when annotations are present.

Overall, we cannot conclude that annotating functions within a project affects how far exceptions will travel. However, projects that annotate functions tend to have exceptions with longer throw lengths. Thus, we do not see an effect of exception specification on individual functions, while there is an effect at the project level.

4.6 Summary

We performed an empirical study on open-source projects from Github to determine how exceptions affect projects. We asked how exceptions may flow between files and through a program in general. We also considered how the presence of EH may change the structure of code and whether specifying functions throwing exceptions using exception specifications may alter the how often the exceptions are caught.

We found the following insights into how exceptions may affect a program.

- Most exceptions do not travel outside of the file they are thrown from, but those that do can be caught more often.
- Exceptions add a number of control flow paths to a program that is linear correlated with the number of functions in the program.
- The presence of exceptions in a system changes the use of control flow or memory management. Code outside of exception blocks is distinct from exception blocks based on the use of these statements.
- We cannot conclude that exception specifications on a function affect its throw length. Projects that use exception specifications tend to have exceptions with higher throw lengths.

Overall, exceptions have an impact on the control flow of a program that is not represented in a typical call graph. Exception handling code uses language features differently than other code. Additionally, while a benefit of exceptions is said to be the ability to handle errors between modules, this is an common case for exception usage.

Chapter 5

Conclusions

Our work presents two major contributions:

- A case study involving the use of throw length, exception graphs, and Zelda. The study involved analyzing a corpus of C++ code for exception usage and flow to determine how exceptions are used and their unseen effects on the project. We addressed exception flow between files and exception flow using throw length and exception graphs. We also studied the effects of exception usage on code structure through use of C++ features in exception code and how exception annotation on a function impacts exception flow.
- The development of Zelda, which performs static analysis about exceptions on C++ source code. Zelda will be released as open source code and can serve as a starting point for further exception research. Zelda currently performs exception detection and propagation of exceptions through possible paths in a program. Furthermore, the output from Zelda involves text-based data to faciliate studies on large data sets.

We found that exception usage impacts various aspects of C++ programs including exception flow increasing the size of a call graph by an average of 22% and that most exception handling is localized to a file, but exceptions are handled more frequently when traveling between files. Furthermore, there are about eight functions that throw exceptions indirectly for every one function that directly throws an exception. Code contained with `try` and `catch` blocks is distinct from non-exceptional code which could reflect the goals of the code. We could not conclude that exception specifications made a difference in the throw length of an exception.

Overall, it seems that the hesitance from developers to use exceptions may be justified. The effects of exceptions seem to be significant to several aspects of a program. The flow of exceptions may not be easily noticed or tracked as systems grow in size. Using software, such as Zelda, can alleviate some of the burden on developers to track exception flow and ensure the robustness of their software.

5.1 Limitations

The corpus studied is taken from the portion of GitHub that is publicly available. The projects in the corpus had a main language of C++, existed for at least one year, had at least 100 commits, and had been committed to in the last year to ensure the projects were real, current projects. Additionally, projects that had similar names were examined to ensure there were not forks of projects already contained in in the corpus. The results may not generalize to other bodies of code.

The C++ Standard Library contains many classes that developers commonly use such as `string`, `vector`, input and output, as well as other common utilities. Many of these utilities are designed to throw exceptions when used inappropriately. However, exceptions from built-in libraries are not considered. Thus, the results in this thesis reflect user exceptions and may not apply to the usage of the standard library code. We decided that this was a good trade-off, as we are primarily interested in how ordinary C++ developers use exceptions, rather that C++ library designers.

Similarly, exceptions originating from third-party libraries are not considered in the analysis unless the code is included within a project. This is due to the large number of libraries that exist and the difficulty of ensuring all required information would be provided, such as compiler flags. While this information could be determined from `Makefile`s and other compilation systems, there are many systems used within the corpus making, determining the required libraries difficult.

Thus, only the code present and exceptions written within the project are analyzed. While this does not ensure that the code is written as part of the project, it does reflect the code that is used. This also means that if projects include code for a library, the project is analyzed with the version of the code that it would use.

Templated functions are not analyzed by Zelda. This is due to the unique AST structure of such functions. We would also have to consider whether `throw` statements from multiple instances of templated functions should be considered different exceptions. There is uncertainty as to how these exceptions should be considered and there is added complexity that makes handling these functions more difficult. We expect that templated functions would not greatly change the results unless they are implemented significantly differently than typical functions.

Our analysis also does not account for the use of function pointers. Thus, it is possible that functions are called through pointers that we were unable to take into consideration. However, this is a general problem when working with function pointers and higher order functions. The only ways to address this concern is to either keep track of the possible variables that have been assigned to a function pointer and assume that a call can be to any such function, or to assume that any function whose signature matches the pointer could be called. Either would not be accurate due to over estimating the number of functions that could be called.

For the analysis of intermodule exceptions, we defined a module to be all code written within a particular file. Our definition of modules may differ from others' definitions. However, our definition reflects that a developer would have to look outside of a file to determine exception information about code.

There are other possible definitions for a module that were considered. For instance, functions within a header file could used as a definition for a module. However, this would not account for static functions that may be present. Classes and namespaces could also be used which results in different rules about modules being considered. For instance, would two classes in the same namespace be in the same module despite potentially having no relation? Would nested classes be considered to be in the same module as the class they are nested? Thus, we used files as our definition of modules.

While Zelda was tested against another known tool for correctness of call graphs, further correctness of exception flow was not easily tested because there is no tool available to compare the results against. However, between the described algorithm, testing during the development process, and manually checking results from projects in the corpus, we are confident the tool works as described and intended. However, there is the possibility that Zelda does not work in situations that we have not encountered.

5.2 Future Work

5.2.1 Improvements to Zelda

While Zelda works as intended, there are aspects of C++ that are not addressed currently. In particular, improvements could include more inclusive analysis of functions, dead code analysis, and the analysis of language features.

Analysis of Functions

The analysis of templated functions, calls to virtual methods and parent constructors, and implicit destructor calls would require additional information about class hierarchies than the current analysis.

Another point of improvement is to distinguish between functions that are marked to throw exceptions and those that will never throw exceptions. As the C++20 standard, the only language component will be `noexcept(expr)`, where `expr` is an expression that evaluates to true or false. Further analysis of `noexcept` statements would determine if the function is marked to never throw, potentially throw, or throw based on some property of a class. This would both determine how this program feature is used and improve exception flow analysis.

Dead Code Analysis

At this point, the analysis process assumes any code that is present will be executed at some point. However, there is certainly code that is not executed included in the analysis. For example, there are functions that are never called and conditions that can never be met. Adding basic dead code detection could be used to determine if `throw` statements could be executed. With the exception analysis involved, `catch` statements and code after a `throw` statement could also be detected as dead code. Determining what exception code is dead code, could give insight into where developers use exceptions that are not necessary and how to use EHM more effectively.

Language Features

Pursuing additional information about the use of language features could lead to a better understanding of when people use exceptions. Instead of looking at what features are used in parts of a program, a study could be performed to address what contexts exceptions are used in. For instance, are exceptions commonly thrown from failed conditional statements, or how often are functions called to facilitate the throwing of exceptions.

Our work has focused on exceptions that are caught and thrown by the same project. While one of our questions addressed whether exceptions are thrown between modules, this question could be extended to between libraries. This would involve having the source code for third party libraries available for analysis with projects. Ideally, the library code could be checked separately from a project that includes it, and the exception flow could be analyzed by combining the information from the project and the library. This approach could also facilitate the analysis of exceptions from the standard C++ library.

5.2.2 Additional Programming Languages

We chose to focus on C++ exceptions when addressing exception concerns. The results in this work may not reflect how exceptions affect other languages. The research questions from our work could be answered for other programming languages with exceptions. Due to exceptions being more restrictive in most programming languages, the results could vary drastically.

5.2.3 Code Defects

We focused on the effects of exceptions in C++ code without looking at how exceptions may affect the robustness of a program. The analysis performed could be linked to other information about code. For instance, one could ask whether code with exceptions is more likely to lead to code defects and address this by comparing bug reports and pull requests with exception usage. Combining the knowledge of exception flow from Zelda with such reports could show that exceptions have a wider impact than it seems.

References

[1] Rodrigo Bonifacio, Fausto Carvalho, Guilherme N. Ramos, Uira Kulesza, and Roberta Coelho. The use of C++ exception handling constructs: A comprehensive study. In *Proc. of the 2015 IEEE International Working Conference on Source Code Analysis and Manipulation, (SCAM-2015)*, pages 21–30, 2015.

[2] Jong-Deok Choi, David Grove, Michael Hind, and Vivek Sarkar. Efficient and precise modeling of exceptions for the analysis of Java programs. In *Proc. of the 1999 ACM SIGPLAN-SIGSOFT workshop on Program Analysis for Software Tools and Engineering (PASTE-1999)*, pages 21–31, 2004.

[3] Christophe De Dinechin. C++ exception handling. *IEEE Concurrency*, 8(4):72–79, 2000.

[4] John B. Goodenough. Exception handling: issues and a proposed notation. *Communications of the ACM*, 18(12):683–696, 1975.

[5] Google. Google C++ Style Guide: Exceptions. `https://google.github.io/styleguide/cppguide.html#Exceptions`.

[6] Georgios Gousios and Diomidis Spinellis. GHTorrent : Github ' s Data from a Firehose. In *Proc. of the 9th IEEE Working Conference on Mining Software Repositories (MSR-2012)*, pages 12–21, 2012.

[7] Tero Hasu. Concrete error handling mechanisms should be configurable. In *Proc. of the 5th International Workshop on Exception Handling*, WEH '12, pages 46–48, Piscataway, NJ, USA, 2012. IEEE Press.

[8] Rick Holt. Ta: The tuple attribute language, 1997.

[9] Rick Holt. Introduction to the grok language, 2002.

[10] T. Jones and T. Gilliam. *Monty Python and the Holy Grail*, motion picture, Python (Monty) Pictures, 1975.

[11] Maria Kechagia, Tushar Sharma, and Diomidis Spinellis. Towards a context dependent Java exceptions hierarchy. In *Proc. of the 39th International Conference on Software Engineering Companion (ICSE-C-2017)*, pages 347–349. IEEE Press, 2017.

[12] Mary Beth Kery, Claire Le Goues, and Brad A. Myers. Examining programmer practices for locally handling exceptions. In *Proc. of the 13th International Workshop on Mining Software Repositories (MSR-2016)*, pages 484–487, 2016.

[13] Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis and transformation. In *Proc. of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, pages 75–88, Mar 2004.

[14] Roy Levin. *Program Structures for Exceptional Condition Handling.* PhD thesis, Carnegie Mellon University, 1977.

[15] Barbara H. Liskov and Alan Snyder. Exception Handling in CLU. *IEEE Transactions on Software Engineering*, SE-5(6):546–558, 1979.

[16] M Donald Maclaren. Exception Handling in PL/I. In *Proc. of an ACM conference on Language design for reliable software*, pages 101–104, Massachusetts, USA, 1977.

[17] Cristina Marinescu. Are the classes that use exceptions defect prone? In *Proc. of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution (IWPSE-EVOL-11)*, pages 56–60, New York, New York, USA, 2011. ACM Press.

[18] Cristina Marinescu, Radu Marinescu, Petru Florin Mihancea, Daniel Ratiu, and Richard Wettel. iplasma: An integrated platform for quality assessment of object-oriented design. In *Proc. IEEE International Conference on Software Maintenance (ICSM Industrial and Tool Volume)*, pages 77–80. Society Press, 2005.

[19] Thomas J McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976.

[20] Bryan J Muscedere. Detecting Feature-Interaction Hotspots in Automotive Software using Relational Algebra by. Master's thesis, University of Waterloo, 2018.

[21] Bryan J Muscedere. Rex. `https://github.com/bmuscede/Rex`, 2018.

[22] Suman Nakshatri, Maithri Hegde, and Sahithi Thandra. Analysis of exception handling patterns in Java projects. In *Proc. of the 13th International Workshop on Mining Software Repositories (MSR-2016)*, pages 500–503, 2016.

[23] Juliana Oliveira, Nelio Cacho, Deise Borges, Thaisa Silva, and Fernando Castor. An Exploratory Study of Exception Handling Behavior in Evolving Android and Java Applications. In *Proc. of the 30th Brazilian Symposium on Software Engineering - SBES '16*, pages 23–32, 2016.

[24] Prakash Prabhu, Naoto Maeda, Gogul Balakrishnan, Franjo Ivanči, and Aarti Gupta. Interprocedural Exception Analysis for C++. In *Proc. of 25th European Conference on Object-Oriented Programming (ECOOP-2011)*, pages 583–608, 2011.

[25] Martin P. Robillard and Gail C. Murphy. Designing robust Java programs with exceptions. In *Proc. of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineerings*, pages 2–10, 2000.

[26] Martin P. Robillard and Gail C. Murphy. Static analysis to support the evolution of exception structure in object-oriented systems. *ACM Transactions on Software Engineering and Methodology*, pages 191–221, 2003.

[27] Jonathan L. Schilling. Optimizing away C++ exception handling. *SIGPLAN Not.*, 33(8):40–47, August 1998.

[28] Scitools. Scitools' Understand. https://scitools.com, 2019.

[29] Hina Shah, Carsten Görg, and Mary Jean Harrold. Visualization of exception handling constructs to support program understanding. In *Proc. of the 4th ACM Symposium on Software Visualization*, page 19, 2008.

[30] Hina Shah, Carsten Görg, and Mary Jean Harrold. Why Do Developers Neglect Exception Handling? In *Proc. of the 4th International Workshop on Exception Handling*, pages 62–68, 2008.

[31] Saurabh Sinha and Mary Jean Harrold. Analysis and testing of programs with exception handling constructs. *IEEE Transactions on Software Engineering*, pages 849–871, 2000.

[32] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, 4th edition, 2013.

[33] Tao Xie and Suresh Thummalapenta. Making Exceptions on Exception Handling. In *Proc. of the 5th International Workshop on Exception Handling*, pages 1–3, 2012.

Appendix A

Exception Code Example

This appendix provides an extended code example that details the state of memory during the execution of a program and how exceptions effect the state of the program.

Listing A.1: C++ Rational Number Implementation

```
class DivisionByZero{};

class InvalidInput{};

class Rational{
    public:
        Rational(): num{0}, den{1} {}

        Rational(int _num, int _den): num{_num}, den{_den}{
            divisionByZeroCheck();
            reduce();
        }

        Rational inverse(){
            return Rational{den,num};
        }

        Rational operator*(const int& k){
            return Rational{num* k, den);
        }

```

```cpp
        Rational operator*(const Rational& rhs){
            return Rational{num * rhs.num, den * rhs.den};
        }

        Rational operator/(const Rational& rhs){
            return (*this) * rhs.inverse();
        }

        Rational operator+(const Rational& rhs){
            return Rational{(*this) * rhs.den + rhs.num * den,
                                den * rhs.den};
        }

        Rational operator-(const Rational& rhs){
            return (*this) + (rhs * -1);
        }

        friend ostream& operator<<(ostream&, const Rational&);
    private:
        int num, den;

        void divisionByZeroCheck(){
            if ( den == 0 ) throw DivisionByZero{};
        }

        void reduce(){
            int gcd = GCD(num, den);
            num /= gcd; den /= gcd;
            if ( den < 0 ){ num *= -1; den *= -1; }
        }

        int GCD(int a, int b){
            if ( b == 0 ) return a;
            if ( a == 0 ) return b;
            return GCD( b, a % b );
        }
};

```

```
ostream& operator<<(ostream& out, const Rational& r){
    out << r.num << "/" << r.den;
    return out;
}

Rational readRational(){
    int num, den;
    cout << "Enter two numbers:" << endl;
    while (cin >> num >> den){
        try {
            Rational temp{num, den};
            return temp;
        } catch (DivisionByZero& dbz){
            cout << "The second number cannot be zero. "
            cout << "Enter two numbers:" << endl;
        }
    }
    throw InvalidInput{};
}

int main(){
    Rational r
    Rational s;
    try {
        r = readRational();
        s = readRational();
        cout << "Calculating " << r << " divided by "
                 << s << ":" << endl;
        Rational t = r / s;
    } catch (DivisionByZero& dbz) {
        cerr << "Cannot divide by " << s << endl;
        return 1;
    }
}
```

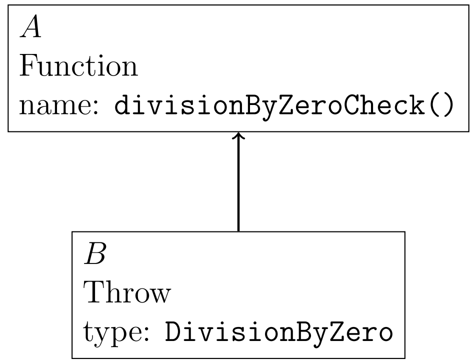

Figure A.1: Context graph for `divisionByZeroCheck()`. Arrows shows the context of a node. In this graph, A is the context of B.

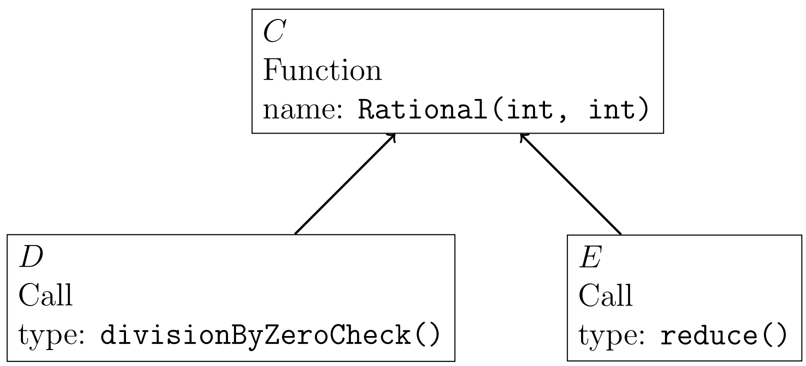

Figure A.2: Context graph for `Rational(int, int)`.

A.1 Context Graph

A context graph is an interpretation of scope required to propagate an exception through the call stack accurately. For brevity, the calls to output and input operators are not displayed. The following examples includes the context graphs for `divisionByZeroCheck()`, `Rational(int,int)`, and `readRational()`.

The function `divisionByZeroCheck()` is displayed in Figure A.1 and contains two nodes. Node A for the function, and node B for the `throw` on line 44 with type `DivisionByZero`. The `throw` expression is not in a `try` or `catch` block, so the context for the `throw` is the function.

The function `Rational(int,int)` is displayed in Figure A.2 and contains three nodes. Node C for the function, node D for the call to `divisionByZeroCheck()` on line 10, and

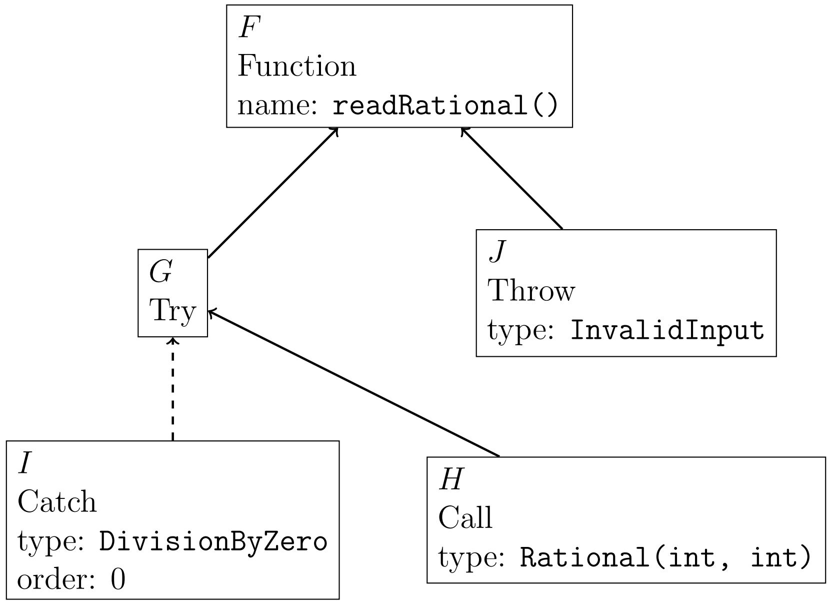

Figure A.3: Context graph for `readRational()`. A dashed line shows that a catch statement is associated with a try statement.

node E for the call to `reduce()` on line 11. Node C is the context for D and E.

The function `readRational()` is displayed in Figure A.3 and contains five nodes. Node F for the function, node G for the `try` statement on lines 69 to 72, node H for the call to `Rational(int,int)` on line 70, node I for the `catch` statement on lines 72 to 75 with `catch` type `divisionByZero`, and node J for the `throw` on line 777 with type `InvalidInput`. Node F is the context for G and J. Node G is the context for H. Node I is a `catch` statement for Node G.

A.2 Exception Flow Algorithm

Using the context graphs created in the last section, we will demonstrate how exception flow will be calculated for the three functions in this section. The algorithm is presented in Section 3.3.2.

The first step in the algorithm is to find all `throw` nodes which are not rethrows, thus $T =< B, J >$. For each `throw` in T, all context edges are found. For B there is $C_B =< (A, B) >$, and for J there is $C_J =< (F, J) >$.

To process B, consult each context in C_B. The first context is (A, B) and $C_B =<>$. A is a function, and the context of each call is added to C_B. The only call to A is D, whose context is C, thus $C_B =< (C, B) >$. The next context is (C, B) and $C_B =<>$. C is also a function and is called from H, whose context is G, thus $C_B =< (G, B) >$. The next context is (G, B) and $C_B =<>$. G is a `try` statement with `catch` statement I. The type of B is `divisionByZero` which matches the type of I. Thus (I, B) is added to C_B. The next context is (I, B) and $C_B =<>$. I is a `catch`, B is not a `throw` from I, and I does not contain a rethrow, thus B is caught by I. Since C_B is empty, processing B is complete.

To process J, consult each context in C_J. The first context is (F, J) and $C_J =<>$. F is a function, and the context of each call is added to C_J. There are no calls to F in this subset of the context graph, thus exception propagation is complete.

A.3 Stack Unwinding

In this section, we simulate the execution of the given program with two different input. For the example, the input is "1 0" and the stacks are displayed in Figures A.4 to A.6.

The first point of interest while executing this program is the initialization of `r` and `s` on line 81 and 82 `main()`. This adds to variables to the context of main. Next, a scope is added for the `try` statement.

The call on line 84 to `readRational()` is added to the stack with variable `num` and `den` on line 66. The scope for the `while` loop on line 68, the variables `num` and `den` are assigned 1 and 0 respectively from reading the first two inputs, and the `try` statement on line 69 are added.

The call on line 70 to `Rational(int,int)` is added to the stack which allocates the parameters `_num` and `num` to be 1, and `_den` and `den` to be 0. The call to `divisionByZeroCheck()` on line 10 is added to the the stack, represented in Figure A.4.

Due to `den` being 0, a `DivisionByZero` exception is thrown from line 44. This exception unwinds the stack until is reaches the `try` block from line 69, Figure A.5. The associated `catch` has a catch type of `DivisionByZero`. The `catch` body is run and control flow continues at line 75.

The `while` loop executes again. With no input, the reads from `cin` fail and the loop is done. An `InvalidInput` exception is thrown from line 77. There is no handler on the stack for this exception. The entire stack unwinds and the program terminates, Figure A.6.

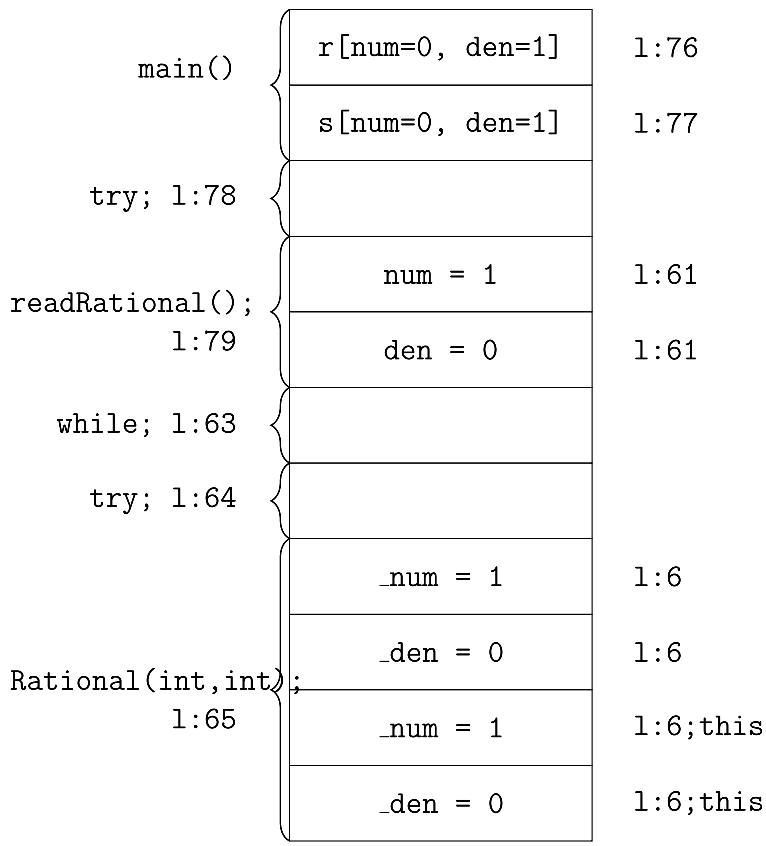

Figure A.4: Stack unwinding example 1.

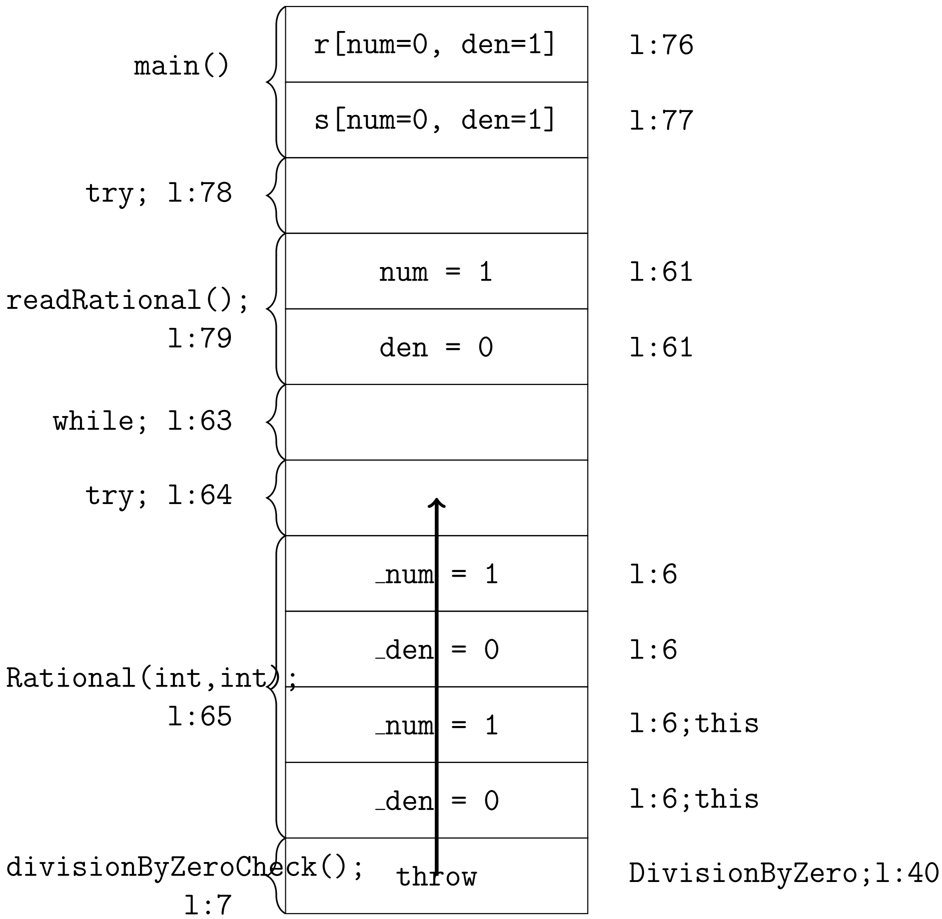

Figure A.5: Stack 2. The `DivisionByZero` exception is caught by the handler for the `try` on line 64. The stack frames for `divisionByZeroCheck()` and `Rational(int,int)` are removed by stack unwinding.

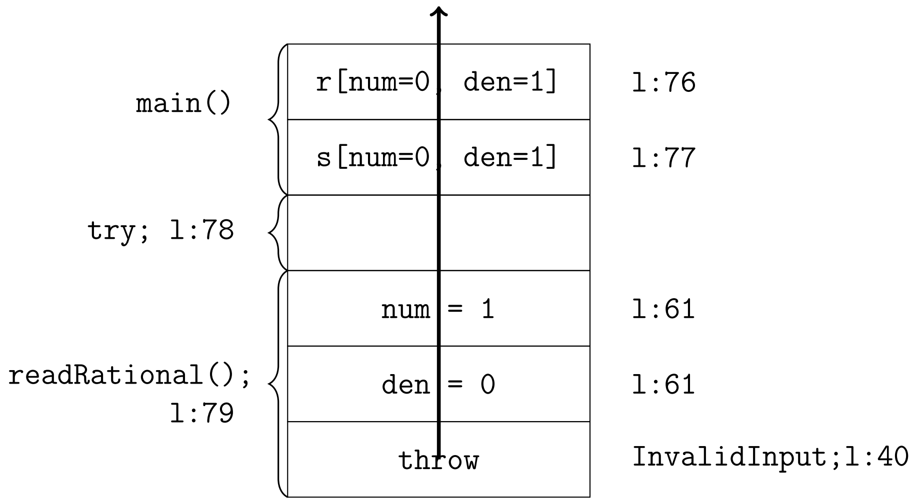

Figure A.6: Stack 3. The handler in the main function only handles `DivisionByZero` exceptions. The `InvalidInput` exception remains uncaught. The stack unwinds main and the program terminates.

A.4 Exception Graph

Figure A.7 depicts the exception graph for the code example.

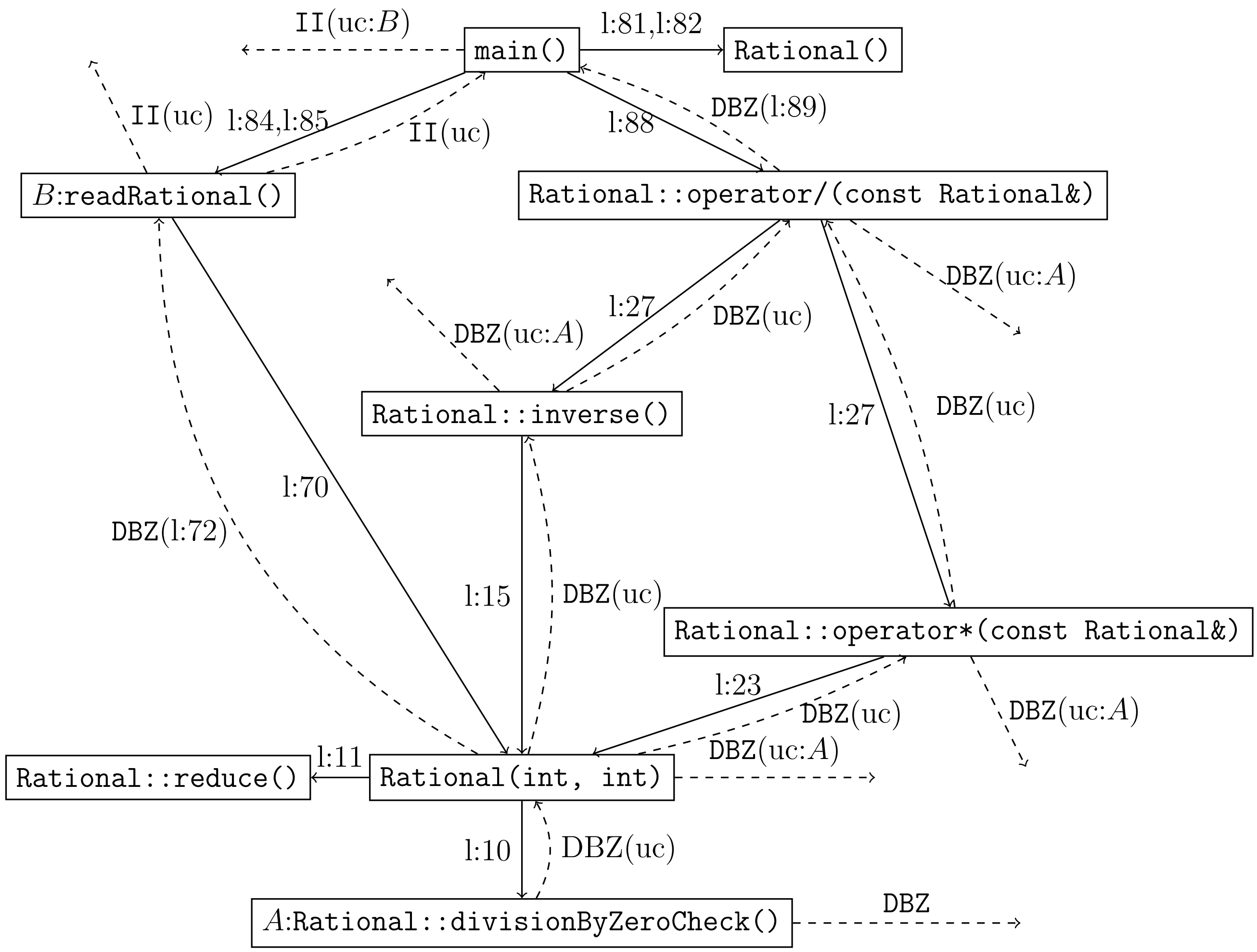

Figure A.7: Exception graph example. Solid lines indicate a function call and the line the call is from is a label on the edge. Dashed lines indicate a thrown exception. Dashed lines that are not directed at a node indicates the function throws the exception. Only functions reachable from `main` are included and calls to input and output operators are ignored. `DBZ` is short for `DivisionByZero` and `II` is short for `InvalidInput`.

Appendix B

Corpus

This appendix provides a summary of the GitHub projects studied, and some metrics which were used. The project column lists the owner of the project followed by the project's name. The project column is the GitHub username followed by the project name. Projects can be found at www.github.com/Username/Project where Username and Project is the entry in the Project column of the table.

	LineCount			Occurrences		
Project	Total	try	catch	try	catch	throw
matplotlib basemap	37396	2203	2323	394	554	187
cocos2d cocos2d-x	185718	0	0	1	0	1
mana manaserv	7381	12	13	5	5	8
Kitware ParaView	126037	127	21	8	8	32
blackberry WebWorks-Community-APIs	114082	339	55	17	17	60
neubig kytea	6310	18	10	2	2	3
SignFinder FaceCore	194895	22	9	3	3	95
Kimbanjang Mong2Core	180622	60	29	9	9	102
AMDmi3 streetmangler	3570	40	72	11	18	15
freicoin freicoin	23613	678	290	90	86	163
arx ArxLibertatis	25036	31	14	7	6	3
facebook folly	40375	684	624	164	244	70
hpcc-systems HPCC-Platform	307456	13681	4976	1409	1674	554
nmap nmap	100701	3	4	1	1	0
in-silico libann	3686	233	55	15	16	88
mapnik mapnik	15773	1070	298	88	99	106
machinalis protobuf-python3	22767	0	0	0	0	1
enigma-dev enigma-dev	60249	16	7	3	2	1
alcoheca xbmc	316586	5440	4304	556	1488	68
puppetlabs facter	4252	94	67	29	22	2
OKullmann oklibrary	44364	558	246	55	72	63
gerstrong Commander-Genius	39534	9	6	6	4	3
opentibia server	25181	114	89	29	22	5
seomoz simhash-cpp	539	0	0	0	0	3
moai moai-dev	593472	93	67	25	25	1
Singular spielwiese-ci	82972	108	34	34	17	1
lemire FastPFor	47514	0	0	5	0	29
cpp-netlib cpp-netlib	1414	90	51	16	18	3
edubart otclient	14870	343	115	52	45	2
dictoon appleseed	27242	233	184	76	74	69
znc znc	23802	71	64	9	9	2
uspgamedev ugdk	95039	3	3	1	1	23
openframeworks openFrameworks	53319	193	122	45	44	60
zli236 voltdb	10398	480	417	123	123	30
nonolith connect	3873	157	22	16	9	13
fritzo pomagma	819462	10633	11184	1208	2125	29
LibreCAD LibreCAD	63219	47	21	6	6	8
membase ep-engine	22099	323	210	67	92	112
inspircd inspircd	24942	157	99	35	35	93
nuigroup ccv2	17622	262	129	33	32	17
mjwybrow adaptagrams	25587	154	51	18	17	13
garrison vmc	1446	0	0	0	0	5
vlag solarus	91148	125	248	61	124	5
solarus-games solarus	92355	125	248	61	124	5
aseba-community aseba	17811	279	208	72	74	97
pioneerspacesim pioneer	61999	82	66	18	25	229
twitter mysql	781936	2	2	13	1	3
noname22 spank	3712	191	46	4	4	13
kvahed codeare	9995	304	168	55	63	61
apache thrift	78651	1586	1219	257	340	677
visionworkbench visionworkbench	62916	1377	1108	266	355	5
nickbnf glogg	4534	37	31	5	5	5
weyrick roadsend-php-raven	20510	48	30	10	9	33
ispc ispc	28820	21	10	3	3	10
jonigata caper	6751	61	32	6	6	43
ondras TeaJS	470349	2535	2511	369	649	57
toastedcrumpets DynamO	2535	154	62	15	20	14
postgres pgadmin3	57434	14	25	6	8	11
mana mana	11656	49	25	8	8	20
mysmartgrid hexabus	18554	388	354	79	95	119
Eyescale Equalizer	21290	93	29	12	11	8
Razor-qt razor-qt	13720	95	7	3	3	3
Eyescale Collage	6559	63	22	8	7	2
lightspark lightspark	40263	433	223	86	87	6
lindahua light-matrix	9587	0	0	0	0	1
mickem nscp	66966	4017	3169	490	787	157
project8 monarch	1907	361	88	24	31	33
rescrv HyperDex	5416	379	97	24	31	1
broune mathic	978	3	9	1	1	3
blackrim phlawd	4463	0	0	0	0	3
poulson Clique	27418	1104	22	4	11	2
espressomd espresso	14305	108	33	10	9	39
fador mineserver	12345	64	19	4	6	3
oftc oftc-ircd	1482	21	6	2	2	11
dworkin dgd	23520	584	103	28	28	3
p3 regal	684333	92	10	3	4	4
openscenegraph OpenSceneGraph	25282988	1728	310	91	105	183

	LineCount			Occurrences		
Project	Total	try	catch	try	catch	throw
pgRouting pgrouting	5805	756	395	43	108	4
krofna Warrior-of-Dreamworld	1301	18	18	10	8	14
celeron55 minetest	47264	996	689	209	301	22
Ummon D-LAN	5258	86	59	16	28	7
BlueBrain codash	381	3	3	2	1	1
openSUSE one-click-installer	699	21	34	6	10	2
kcat openmw	62447	735	244	74	77	392
adobe webkit	304455	3344	563	233	227	4
aburch simutrans	109452	25	12	3	3	4
Dwachs simutrans	101028	25	12	3	3	4
commontk CTK	25610	726	527	179	204	23
Sterios SS_Core	181810	68	36	9	9	102
hmmr cnrun	1964	21	22	9	9	21
rsjtdrjgfuzkfg nightingale-hacking	25281	0	0	0	0	1
Slicer Slicer	64313	472	377	121	149	2
OpenCPN OpenCPN	195425	208	21	4	6	11
openSUSE libzypp	16441	817	407	122	127	10
danielkeller vec	3026	2	2	1	1	4
project8 katydid	12539	194	82	35	29	2
worldforge cyphesis	42971	134	69	24	26	10
sjrd mozart2	4117	4	6	8	2	2
tklab-tud umundo	10413	78	24	9	10	7
yetanothergeek fxite	62826	46	32	9	10	6
PongUIO AndroidWars	3315	0	0	1	0	1
bhaak HackedUpReader	154791	168	23	6	7	25
jaaro Shuffla	2791	26	17	11	5	2
YggdrasiI KinectGrid	33378	471	41	17	14	70
mojca xetex	499071	131	32	15	8	36
hpcc-systems GraphControl	3500	56	47	24	17	8
xapian xapian	33333	1521	668	134	205	64
whackashoe cprocessing	2888	0	0	0	0	1
dterei Scraps	4256	24	41	10	12	5
NeoGeographyToolkit StereoPipeline	9147	169	90	27	29	5
lojack5 CBash	29621	451	358	96	157	10
shikadilord morningstar	775	7	8	2	3	4
openexr openexr	24838	2114	1137	261	308	231
wjwwood serial	1182	5	5	2	2	33
rakshasa rtorrent	5680	268	79	33	32	17
rakshasa libtorrent	7297	406	141	30	50	28
audacious-media-player audacious-plugins	75477	9	6	3	2	1
vslavik winsparkle	1262	135	153	27	51	37
diclophis MemoryLeak	57601	431	130	41	40	24
Forkk MultiMC4	10506	168	71	18	27	2
maidsafe-archive MaidSafe-Common	3463	157	91	42	34	3
adamnew123456 SmallWM	7410	1300	1624	375	459	8
plasmodic ecto_opencv	1835	0	0	3	0	16
mgyucht Summer_2012	1326	0	0	1	0	1
Twinklebear LPCGame	4595	10	6	2	2	5
BYVoid OpenCC	45790	1204	1216	180	323	35
ailue Shuihusha	31805	3	4	1	1	0
taskwarrior task	12470	652	405	150	163	2
xyzzy-022 xyzzy	55248	786	428	123	123	52
myfavouritekk iGEM2012	3019	0	0	0	0	4
pyne pyne	178725	12	18	4	6	101
Orphus TrinityCore	179888	68	36	9	9	102
klorel clusterisator	2108	0	0	1	0	2
tahoe-lafs pycryptopp	20739	33	28	11	11	80
openwebos db8	56821	1104	1121	163	296	14
Sankore Sankore-3.1	24031	56	17	6	6	18
OpenClovis SAFplus-Availability-Scalability-Platform	137507	868	685	108	99	133
lattice quda	34558	62	18	2	4	3
Lirusaito SingularityViewer	224073	248	204	66	55	35
codels TrinityNya	193049	22	9	3	3	95
mweidler Inverita	2265	4	0	4	4	15
CJThomson MD-Stepped-Potential-Simulator	7624	0	0	1	0	6
0ad 0ad	88023	344	112	40	37	57
SamuelCho Freespace-Open-Swifty	184407	3128	2121	542	610	1
dscharrer innoextract	1781	71	48	15	17	10
shewu h4ck4th0n	44952	1106	1124	164	297	35
knz mgsim	6322	81	66	16	16	11
Kitware VTK	749484	4942	1894	363	364	29
hojonathanho bulletsim	194317	305	107	29	29	20
zussel oos	20464	423	419	62	78	76
MITK MITK	141961	13053	2903	791	1122	87
carson airconvision	21858	23	9	5	4	14

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
adaptivegenome openge	7652	18	11	5	3	4
NIF-au TissueStack	6392	541	392	90	100	4
Jildor ZoneLimit	165401	68	36	9	9	102
gcross Nutcracker	2286	112	125	20	24	25
ipa-fmw cob_driver	7151	45	23	17	10	1
danmar cppcheck	93830	899	1090	224	427	60
tonioni WinUAE	287607	975	236	57	58	100
elixir67 Sandbox	6776	14	16	9	5	0
lmccalman reverend	331	0	0	0	0	8
dblock dotnetinstaller	8369	235	132	55	57	43
openSUSE snapper	4837	141	156	25	59	42
wg-perception object_recognition_core	1415	11	11	5	4	10
ehebrard Mistral-2.0	37432	526	105	28	32	12
MihailJP MiHaJong	18056	169	117	34	38	39
mobile-shell mosh	4447	233	88	21	26	9
cebix macemu	69041	9	44	3	11	15
pixelballoon pixelboost	22006	0	0	0	0	6
robn pioneer	43007	82	66	18	25	227
laarmen pioneer	62077	82	66	18	25	229
wg-perception libmv	71386	189	234	35	69	6
thomasmoelhave tpie	12363	281	137	45	49	5
Brianetta pioneer	61693	82	66	18	25	230
Eigenlabs EigenD	212415	1023	962	193	437	29
supercollider supercollider	0	142	0	153	142	208
alanbriolat clementine-subsonic	98598	24	19	7	7	44
yedaoq YedaoqCodeSpace	16906	18	18	10	9	21
RhobanProject Common	1613	105	72	16	24	4
mozilla-services services-central	1518719	3597	627	243	249	74
Jackarain avplayer	51396	5425	4680	1190	2329	18
RhobanProject Utils	61815	150	89	21	36	52
dumganhar nui3	314325	8	3	10	1	1
ciyam ciyam	151323	7408	760	141	177	2602
chenshuo muduo	6384	45	27	5	8	5
zaphoyd websocketpp	6890	229	207	74	78	2
Intline9 IntPe9	6809	52	38	8	12	5
Intline9 IntWars	185634	40	22	7	7	0
jckarter clay	13784	74	12	6	4	1
originell jpype	7734	843	1301	213	384	1
yxl Fire-IE	8568	82	29	12	11	3
coolwanglu pdf2htmlEX	3427	61	19	4	6	16
tomahawk-player tomahawk-resolvers	93848	354	254	52	73	5
lteacy maxsum-cpp	1919	323	65	26	26	4
luispedro imread	1661	64	36	3	12	44
Aloshi EmulationStation	4679	4	8	3	2	2
snes9xgit snes9x	106702	43	23	8	8	21
snes9x-rr snes9x	104363	43	23	8	8	21
qreal qreal	56321	216	288	44	84	4
kmatheussen radium	269433	592	471	95	206	75
aldebaran libalmath	3423	0	0	1	0	33
lvinken MuseScore	207968	554	34	12	14	10
Bromeon Thor	2099	2	3	1	1	5
onyx-intl booxsdk	40269	53	13	4	4	2
qpdf qpdf	14820	618	253	61	66	179
imocha passenger	9569	470	337	60	74	27
sbergen ConductorFollower	2892	13	12	11	4	2
OpenVSP OpenVSP	120344	58	43	9	9	7
cook- Ve280	0	31	0	31	31	38
ReneNyffenegger development_misc	2097	33	12	4	5	6
Ratstail91 Codebase	440	6	4	1	1	27
bkloppenborg simtoi	9966	20	13	4	4	9
muhrin STools	7065	127	84	32	31	5
msgpack msgpack-c	8045	127	136	41	64	10
Imroy photofinish	2058	50	46	17	16	17
sthalik headtracker	691	0	0	0	0	1
OpenFOAM OpenFOAM-2.1.x	86287	37	28	13	10	4
proycon colibri	7536	3	3	1	1	65
rsnitsch degate	9884	191	127	44	42	194
csete gqrx	5054	2	6	2	2	1
malaterre GDCM	74079	75	33	11	12	51
miquelramirez simulpast-cs1	272184	778	324	115	132	18
maya2renderer maya2renderer	90008	2240	2069	358	678	36
omegaonline oocore	96565	3212	3862	849	1206	473
pilkch library	43309	104	17	5	4	9
erwincoumans bullet3	134333	10	19	4	6	38
drk1wi portspoof	1088	5	5	1	1	4
wichtounet eddic	14431	34	15	3	3	10
bendudson BOUT	30024	126	55	18	18	165

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
youbot youbot_driver	4901	26	7	2	3	3
cyclops1982 powerdns	47094	3886	905	219	280	296
scrawl shiny	2456	112	31	12	6	34
sirikata sirikata	77679	203	339	65	55	15
doughodson OpenEaagles	499431	97	63	20	22	62
wojdyr fityk	17276	217	171	42	46	100
jasonroelofs rice	23326	5779	127	132	498	111
paulscode mupen64plus-ae	204929	49	29	8	7	1
vezzi FRC_align	22709	233	103	21	26	82
Riateche ridual	6363	94	102	14	16	3
xalpha iris	2496	56	23	10	10	8
mworks mworks	7134	51	45	31	19	5
scalien scaliendb	28724	44	84	15	17	4
alanxz SimpleAmqpClient	839	0	0	1	0	13
Mankarse TINS2012Game	537	5	9	1	3	16
iut-ibk DynaMind	48145	1119	1160	170	306	14
raymond-w-ko omegacomplete.vim	1364	0	0	8	0	1
aria2 aria2	46008	1601	947	260	260	330
Tasssadar Lorris	599507	176	113	40	43	2
gang-chen samoyed	0	151	0	122	151	116
Darkpeninsula Darkcore	210741	60	31	9	9	102
ccoffing OcherBook	2909	6	3	1	1	4
freelan-developers freelan	12930	720	181	54	51	152
libLAS libLAS	14847	1563	541	132	152	113
reginakim BRAINSStandAlone	22584	628	450	287	307	125
bytbox EVAN	1784	65	15	3	4	38
mrquincle aim_modules	89867	932	112	48	47	93
girving pentago	1692	4	9	2	2	2
DDMAL libmei	13572	2	7	2	1	8
3breadt UPB-ADT-Automata-Tools	2131	0	0	0	0	9
usc-clmc usc-clmc-ros-pkg	35315	164	122	57	48	1
percolator percolator	21186	276	88	17	29	63
bzar spacerocks	1544	0	0	0	0	1
nextgen-astrodata DAL	2674	51	62	20	23	84
vast-io vast	5506	68	37	14	14	9
yast yast-core	26697	20	20	3	2	14
livingstream madlib	5037	3	2	1	1	5
ViviCoder GM-Assistant	2526	38	36	11	13	21
kbinani libvsq	21810	42	34	13	13	6
cfit cfit	5966	0	0	0	0	104
emdeha Star-Game	250464	27	30	7	6	45
robertmaynard VisIt-Bridge	141804	451	81	29	28	36
g1257 spf	4503	0	0	0	0	3
llvm-mirror lldb	274755	12	10	7	2	3
pelican pelican-lofar	5098	487	111	56	58	1
nasa World-Wind-Java	182248	405	122	48	52	1
mikrosimage duke	2730	13	9	2	4	23
llvm-mirror libcxx	187764	1695	1546	488	507	107
broune mathicgb	6193	39	18	16	6	13
MythTV mythtv	794141	3447	2054	193	369	25
openbabel openbabel	107071	10	8	4	3	1
phys-tools pi-qmc	60883	1104	1121	170	296	14
simsong bulk_extractor	18399	270	111	35	36	27
daviddoria PatchComparison	186	0	0	1	0	1
fasterisk BA-Teil2	193830	2336	751	177	192	530
robotconscience ofxLibwebsockets	1552	40	4	2	2	0
xboxdrv xboxdrv	9664	123	64	15	14	33
DavidPH DH-acc	38659	108	67	16	21	15
jacob1 The-Powder-Toy	29063	279	38	11	11	1
nsmoooose csp	10279	36	43	11	15	8
Constellation iv	30814	3	9	1	3	3
w2schmitt depth-complexity2	4947	78	23	4	8	17
jarad gpuIntroduction	4753	12	21	10	10	10
anttisalonen libcommon	1576	0	0	1	0	11
Obi-Wan vARCH	3271	93	61	11	12	60
mosra magnum	10870	0	0	10	0	1
vsiivola variKN	2764	31	8	4	3	22
patrickmmartin winampremote	5054	1292	1518	88	168	2
OpenNebula one	30492	397	163	47	47	62
simonfuhrmann mve	4554	95	89	34	34	19
chenshuo recipes	19604	108	28	12	7	6
NUbots robocup	101042	994	519	107	147	140
irods irods	115103	1135	858	248	255	3
deek0146 framework2d	5550	94	49	16	16	39
nightingale-media-player nightingale-hacking	26647	0	0	0	0	1
pathscale stdcxx	646366	5439	3807	809	1033	152

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
ioquatix dream	4721	92	25	6	6	30
JelleZijlstra EH	5816	66	55	12	13	12
alanwww1 xbmc-txupdate	4710	78	5	1	1	1
zeux pugixml	53811	597	252	109	110	27
psi-im psi	32518	0	0	8	0	1
marvins Code_Sandbox	4617	43	11	5	4	19
jackaudio jack2	24735	147	87	26	26	5
workflo shaback	11968	107	33	12	12	45
petroules silverlock	7658	502	164	50	58	19
fluxbox fluxbox	23807	76	32	8	13	10
felleh cdec	30711	246	61	26	20	7
kpu kenlm	8304	356	93	34	28	4
chenm001 thevc	20893	30	13	3	4	22
libyui libyui-ncurses	7493	19	21	7	6	8
theunknownxy slin	404	21	12	7	6	2
losalamos CLAMR	26582	74	21	11	7	4
iwiwi programming-contests	86411	133	70	21	21	37
adiknoth ffado	27892	70	47	14	16	1
gunnarbeutner shroudbnc	16461	0	0	0	0	68
sashman terrain_generator	2323	18	18	6	6	24
KPWhiver DimLib	12637	187	3	1	1	20
peteratt ida	15286	207	339	49	86	134
elidupree Lasercake	23004	201	122	52	38	13
rousseau fbrain	0	131	0	103	131	5
OpenChemistry avogadrolibs	18677	0	0	6	0	7
mertdikmen ViVid	8125	15	25	4	3	16
scummvm scummvm-tools	11595	171	65	24	24	8
eurotech edc-examples	82651	354	253	52	73	4
ukoethe vigra	42154	1060	364	100	100	6
otcshare automotive-message-broker	8249	79	39	23	19	26
davidmandle MADTraC	12319	131	17	13	7	2
avxsynth avxsynth	17235	709	383	85	92	47
janm399 akka-patterns	3656	6	12	9	4	18
makerbot Miracle-Grue	31740	571	482	194	192	63
wheresjames winglib	77518	374	193	60	71	2
plasmodic ecto	3894	86	228	37	63	25
CamelliaDPG Camellia	41400	11	10	3	3	6
geovanisouza92 ares	811	12	6	3	2	1
zotero zotero-standalone-build	5720	167	40	9	13	18
anttisalonen brigades2	2598	0	0	1	0	4
jglaser metadynamics-plugin	685	0	0	1	0	10
sfiera libsfz	3974	0	0	0	0	24
blackvladimir hermes-dev	277136	368	148	31	31	101
shaybarak HQMP	978	0	0	6	0	10
GerHobbelt uncrustify	24556	34	28	10	11	8
LORDofDOOM MMOCore	181364	64	31	10	10	102
lyase witty-tutorial	783	16	15	14	7	9
RuslanKutdusov dinosaur	6744	696	507	108	134	188
whiledoing Out_Of_Core_Module	711	41	18	6	5	1
sakhnik gpwsafe	1384	12	9	4	3	35
sgolodetz hesperus2	9752	519	114	54	54	62
audacious-media-player audacious	7618	0	0	0	0	3
aedansilver HD-TCore	168213	68	36	9	9	102
prefetchnta questlab	4925	8	8	1	4	8
Kazade KGLT	37050	48	61	21	21	87
asoroa ukb	5060	868	114	24	28	61
ewxrjk rsbackup	4013	368	149	55	57	30
LASzip LASzip	14696	1726	798	192	192	10
dsth capmon	2108	66	81	10	15	62
wITTus Brute-Force-Game-Engine	6959	514	366	102	108	26
licq-im licq	67525	436	136	36	35	20
takke MZ3	22765	13	9	3	3	0
impulze team_one	2043	297	165	42	51	106
macBdog game	133495	18	21	4	6	15
Monceber Task-1.1	4435	533	299	51	60	27
SergeyStrukov CCore	19593	336	175	69	64	2
gnychis gnuradio-3.5.0-dmr	40435	35	34	14	8	319
pauldoo scratch	4856	198	57	16	20	37
kentron imprudence	160233	209	39	16	16	27
ahiguti pxc	14840	96	81	11	12	21
openstreetmap osm2pgsql	2938	63	85	15	19	36
TeddyDesTodes openttd	125697	484	182	29	34	37
timbaker pzworlded	29098	3	4	1	1	0
bacek xscript	15830	1504	835	217	331	245
FreeRDP FreeRDP-WebConnect	2386	214	34	19	12	10
gnuradio gnuradio	68710	68	60	26	18	554
ddemidov vexcl	2490	24	7	10	3	1

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
paul-h mythtv	796546	3447	2054	193	369	25
shin1m xemmai	33451	135	244	38	43	22
Mendeley Update-Installer	21115	72	36	13	14	1
ruuda deadlock	1225	49	82	15	21	68
jdebrabant h-store	150540	783	619	176	195	14
yudjin87 carousel	3083	0	0	0	0	11
parpwhick Distanze_Entropiche	4659	4	4	1	1	1
m-mizutani swarm	9831	3	9	1	3	3
dermesser libsocket	1913	142	44	14	15	50
vilarion Illarion-Server	10962	1051	302	105	106	64
Illarion-eV Illarion-Server	10878	1047	302	105	106	64
davidiw Dissent	15308	9	17	5	7	3
BerndGabriel simutrans-experimental	125151	29	18	4	5	4
bkaradzic bgfx	169636	6166	7071	525	974	7
aarizaq inetmanet-3.x	15256	16	15	6	6	9
JAChapmanII pbrane	938	23	15	6	5	31
AlexMax odamex	68235	8	9	3	3	1
Gear2D gear2d	7515	6	8	2	2	3
mynew WingsEMU	175751	68	36	9	9	102
shurcooL Conception	6830	32	11	4	4	23
echofourpapa RealTimeTactics	5060	37	18	6	6	3
markboots peg	1270	158	6	2	2	25
Henne dosbox-svn	17210	142	26	4	7	19
BigSisl metoritewars	723	0	0	0	0	1
herumi cybozulib	13463	617	346	78	117	3
mrotondo SuperCollider	124615	813	328	94	104	233
onyx-intl boox-opensource	242679	2847	2762	453	738	48
fonsinchen openttd-cargodist	122202	443	179	29	34	36
datasift zmqpp	2763	41	46	12	13	34
go4and lib	8211	243	127	31	32	15
imageworks OpenColorIO	3011	381	137	31	46	4
but-spanel srs_public	14240	160	166	57	50	16
mediastandardstrust superfastmatch	8737	38	54	8	16	4
jiwonshin aseba	11590	157	142	57	55	80
GordonSmith GraphControl	3480	56	47	24	17	8
tvwerkhoven libsiu	2333	14	7	3	2	22
tvwerkhoven FOAM	4644	21	41	9	15	12
metabrainz libmusicbrainz	4105	84	144	4	24	14
mlang bmc	4662	3	3	2	1	3
setiQuest SonATA	53873	950	355	110	138	15
makerbot MightyBoardFirmware	40320	354	254	52	73	4
npadmana nputils	894	23	10	4	3	1
plexydesk plexydesk	12509	11	3	1	1	2
VoltDB voltdb-client-cpp	4160	48	67	22	16	57
mthomure glimpse-project	357	0	0	0	0	1
xmcpp Cppguru	4241	18	12	12	6	38
fangism hackt	6145	49	50	18	18	9
ankush-me sandbox444	1370	4	6	9	2	6
khwillia repss	5454	133	12	3	3	1
BeginnerSlob TouhouTripleSha	49484	24	18	7	7	0
skyshaw snippets	2051	7	5	3	2	2
4gsim 4Gsim	29806	21	21	8	8	11
pjmikkol bwtc	8702	64	32	5	8	5
i-saint scribble	9989	3	3	1	1	1
MelanieBittl hermes-1	276915	368	148	30	31	81
akrennmair newsbeuter	16679	1166	1152	372	382	39
mawww kakoune	7111	145	78	29	31	79
mariusroets Audit-Agent	7740	0	0	1	0	5
openBliSSART openBliSSART	10330	331	82	17	25	34
danomatika ofxLua	19215	3	4	2	1	36
sawjlab hcana	2951	0	0	0	0	34
goc9000 megas2	5122	10	3	1	1	2
jackyf cupt	11291	633	148	53	54	3
ttsou openbts-p2.8	25596	384	191	36	48	78
firestarter firestarter	347	38	46	18	16	1
svalaskevicius ionPulse	951	27	22	8	10	4
Visomics Visomics	6671	3	3	1	1	1
gec dnp3	21490	599	576	153	149	46
villagereach mScan	2158	13	24	6	6	1
Fadis hermit	3532	0	0	1	0	5
furious-luke libhpc	1288	3	3	8	1	2
DigitalPulseSoftware NazaraEngine	1312533	1053	1083	360	360	4126
anope anope	34192	442	283	106	108	202
couchbase libcouchbase	55153	35	43	14	15	44
geometer FBReader	48032	3	8	13	2	2
Chiru naali	39053	733	296	115	118	13
ipa-rmb cob_people_perception	5051	141	17	15	8	1

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
pixhawk mavconn	11624	138	57	19	18	4
OpenJabNab OpenJabNab	13329	92	15	5	5	11
ufz ogs	44220	246	173	48	57	3
K2InformaticsGmbH erloci	2234	109	285	13	39	35
iut-ibk CityDrain3	8400	450	295	88	88	31
LancasterLogRes Noted	1386	0	0	7	0	9
Henne Bright-Eyes	49697	143	26	4	7	18
mcvsama haruhi	3630	25	13	6	4	6
martingt89 OniboConverter2	4318	19	13	5	5	3
daviddoria Mask	409	0	0	0	0	1
daviddoria PoissonEditing	287	38	31	9	11	2
dmitryduka quadcopter	2349	34	4	3	2	21
ickby openDCM	1929	217	94	20	22	1
kerautret DGtal	19939	111	56	19	20	5
falconpl falcon	146446	684	452	86	89	1691
freundlich fcppt	10696	156	86	28	26	3
OPM opm-core	5920	438	93	28	27	10
arsf lag	8004	83	26	7	9	15
ruisleipa kp2	2475	78	23	11	12	22
wallix redemption	13825	196	184	37	48	6
synfig synfig	67166	1800	528	163	213	98
nebw gsoc2012	102364	1196	1157	177	309	42
makerbot jsonrpc	66	0	0	0	0	2
maidsafe MaidSafe-DHT	4498	124	19	12	5	5
broesdecat Minisatid	79273	2006	1671	259	449	94
apoloval open-airbus-cockpit	17843	702	676	209	221	6
g1257 PsimagLite	1027	0	0	1	0	22
lfranchi tomahawk	102813	454	273	60	82	10
jbcoe CppSandbox	5342	97	89	24	26	16
scoopr vectorial	1458	3	9	1	3	1
encukou desmume	127365	391	206	77	79	83
cpputest cpputest	29816	41	72	13	24	2
robertop pelet	150360	16323	18379	5168	5648	8
arq5x bedtools	17547	247	91	21	21	72
TheJosh chaotic-rage	550928	72	27	9	9	8
alan-wu FieldML-API	6676	3	3	1	1	1
c2s C2Serve	2297	23	15	5	4	5
lucab vermont	10644	37	22	11	10	7
metno wdb	6252	436	203	70	84	62
veltzer demos-linux	24117	28	26	16	9	14
cocaine cocaine-core	2346	84	81	33	29	16
iut-ibk DynaMind-ToolBox	354538	6800	5718	827	1241	249
dc2011 td	6105	0	0	0	0	7
mvan td	6105	0	0	0	0	7
inkooboo areks	78255	3	4	1	1	0
Gris87 ProtocolCreator	3732	83	71	35	35	162
lemire Code-used-on-Daniel-Lemire-s-blog	41542	6	8	2	2	24
sbooth SFBAudioEngine	6464	13	11	4	4	9
KDAB Charm	5967	147	73	17	21	2
Noxalus YAPOG	75973	182	73	23	27	76
qbittorrent qBittorrent	11754	70	29	14	13	5
popcornmix omxplayer	6571	43	17	4	5	2
yllekkram xbpl4kyn	2920	441	823	124	207	9
mrdooz kumi	14853	60	5	1	1	0
theY4Kman viper	4905	99	42	22	21	345
ALive-WoW RC2	166556	60	31	9	9	102
LK8000 LK8000	35016	14	17	4	5	2
ipa-fxm cob_manipulation	27393	6	4	9	2	0
melpon wandbox	2469	90	32	10	9	9
opengm opengm	30015	2519	67	23	22	13
ipa-bnm cob_driver	7098	49	26	18	11	1
micknoise Maximilian	39047	7	7	2	2	53
nu774 qaac	13970	995	1059	277	515	54
vovoid vsxu	53855	440	137	37	39	8
jinchizhong qt-kso-integration	556895	135	161	52	48	22
pvbrowser pvb	268840	3	4	5	1	0
rescrv e	2608	12	11	5	5	1
vogel kadu	22287	105	38	14	15	19
tpaviot oce	116215	1594	1369	246	401	14
Hoikas dirtsand	5869	452	97	26	33	11
MaZderMind osmium	1659	47	20	9	8	1
pkunavin ncxmms2	11264	95	26	5	8	21
openmeeg openmeeg	2085	32	40	7	17	9
CTSRD-CHERI gxemul	66203	3	4	1	1	32
BramvdKroef clessc	4762	83	28	4	8	40
pokerth pokerth	80537	278	12	12	4	13
ElementalAlchemist RoBoBo	3369	30	48	6	13	55

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
encukou pokemon-online	19389	77	38	15	16	10
pogliamarci robotower	5985	13	5	2	2	24
psi-plus psi-plus-snapshots	72675	0	0	20	0	1
eartle liblastfm	1992	23	7	3	3	22
freelan-developers freelan	12930	720	181	54	51	152
benni0815 SchafKopf	2958	0	0	0	0	9
tomaszmrugalski dibbler	27616	271	101	21	22	127
kylelutz chemkit	90566	8	12	11	4	1
TyRoXx Sandboxx	5074	32	22	7	6	49
timschmidt repsnapper	22641	104	100	24	17	17
NDN-Routing ndnSIM	1675	0	0	1	0	1
Malvineous libgamecommon	7300	148	103	33	24	46
Malvineous libgamearchive	6202	127	75	20	23	4
albertjiang gambit	49951	440	201	57	69	170
libbitcoin libbitcoin	360353	264	221	52	37	11
kees-jan scroom	4329	160	43	20	12	20
ashiaro ashiaro	44549	170	182	29	34	212
Marian0 ICFich	6754	0	0	0	0	1
palmer-dabbelt mhng	0	1	0	1	1	1
falconindy ponymix	858	15	15	5	5	4
gcross Illuminate	1675	38	28	13	9	6
Kezeali Fusion	24086	841	333	122	122	20
smogpill dataspace	35970	3	4	1	1	0
eartle lastfm-desktop	15977	397	222	71	78	92
mumurik xyzzy	59782	779	440	125	125	53
hackcraft-de linwarrior	6649	55	25	11	12	21
srz-zumix iutest	357370	2992	7057	706	1395	941
makerbot G3Firmware	68393	635	508	52	73	4
Granjow slowmoVideo	5410	63	51	19	21	2
muchenshou HorseReader	145592	168	23	6	7	25
karlbennett Transcode	1325	123	115	29	28	0
gunoodaddy coconut	5503	309	69	35	31	42
norihiro-w ogs	37071	209	154	41	50	3
jamescoxon dl-fldigi	76062	787	432	82	87	78
thegrandpoobah voronoi	2430	33	14	5	4	7
sipa bitcoin-seeder	2343	13	4	1	1	3
mrlukeparry freecad	246278	1972	950	312	375	61
c-ares c-ares	88604	1246	1289	184	336	14
EternalWind The-Dark-Crystal	3633	0	0	0	0	16
mikael-s-persson ReaK	12406	1399	959	282	287	22
EternalWind ducttape-engine	1558	0	0	0	0	3
ermaker sheep	1885	64	12	3	3	11
flipcoder bitplanes	876	17	13	6	5	13
FroboLab frobomind	9706	76	52	23	19	6
DC2012 DC2012	1929	0	0	0	0	2
chipdude libten	32164	552	179	50	55	21
noam-c EDEn	9282	19	20	4	5	1
MEPP-team MEPP	14982	65	74	22	20	62
clarkcyt QSanguosha	27234	14	10	3	3	0
blackberry-webworks Ripple-Framework	71448	1334	1351	191	352	14
Shawn-Smith InspIRCd	24095	157	95	25	25	50
Wassasin librusql	617	141	123	49	43	1
Dgzt knapsen	1382	97	14	23	7	20
kaos ecos	187814	1289	649	126	182	67
gambitproject gambit	62463	617	195	62	69	222
sempuki code	7377	33	16	31	3	10
evemuproject evemu_crucible	50113	12	12	2	2	140
deltafrog drops	11344	1332	149	80	79	1
patentnetwork CPP_Disambiguation	4525	214	45	10	11	124
Zordey pioneer	61944	82	66	18	25	230
MaxKellermann MPD	21451	1165	777	236	250	226
luminans MultipleViewPipeline	7389	84	124	19	33	3
peadar pstack	2203	38	34	12	12	4
sdayu nokkhum-processor	1888	16	12	6	5	5
luceneplusplus LucenePlusPlus	123932	2605	2574	486	624	11
MHesham IStrategizer	5773	47	15	5	6	2
krivenko triqs	4142	116	27	14	13	6
peter-ch MultiNEAT	4426	3	3	8	1	3
dvanthienen youbot-ros-pkg-erf-demo	4842	19	23	14	7	2
rpavlik loki-lib	13925	706	357	82	102	50
Heeks libarea	10491	89	27	8	7	12
crocdialer KinskiGL	42352	122	72	43	36	11
mborne SFCGAL	8998	323	381	122	122	1
Beman filesystem-proposal	3799	134	155	41	44	2
zmike shotgun	8935	0	0	0	0	6
balint256 gr-baz	6244	0	0	7	0	5
soundradix JUCE	155291	378	399	58	190	7

	LineCount			Occurrences		
Project	Total	try	catch	try	catch	throw
vancegroup vr-jugglua	20282	594	1040	262	481	29
ecell epdp	6204	77	69	19	18	23
Jintram egfrd	7895	92	69	19	18	29
ITKTools ITKTools	0	65	0	63	65	3
saleyn eixx	2769	129	126	40	32	2
georgmartius lpzrobots	99887	174	88	20	26	13
Aico mudlet	5722	4	25	3	2	1
davetcoleman clam	47039	445	231	68	78	183
HongjianLi idock	2458	152	11	4	3	1
homann Quantum-GIS	207444	376	190	63	60	25
hexhex mcsieplugin	662	0	0	1	0	1
angavrilov dfhack	59883	50	99	21	25	8
apolukhin type_index	850	16	12	4	5	1
youbot youbot-ros-pkg	5515	42	20	12	5	2
mateuszboryn mrrocpp	17055	693	961	96	293	29
bdsullivan INDDGO	8942	161	42	18	26	5
kouretes Monas	96619	459	355	104	118	11
visore Visore	88559	0	0	0	0	1
uwssg APS	10977	271	184	47	47	38
elmindreda Nori	50930	2	4	1	1	14
enGits engrid	47192	851	194	37	37	0
korslund Tiggit	2873	229	94	34	37	7
ChaiScript ChaiScript	2326	320	487	91	126	6
GamePad64 p2pnet	1587	66	52	19	18	9
andersk mosh	3946	205	32	7	10	19
ajtack riak-cpp	74633	1246	1289	190	336	14
statgen libStatGen	20174	135	174	58	58	71
jzarl kphotoalbum	12583	21	14	6	6	1
GerHobbelt hamsterdb	84262	93	192	37	40	1
y2q-actionman zatuscheme	4186	115	56	14	15	13
schwehr libais	78438	1286	1302	190	342	11
pvpgn pvpgn-server	39474	117	42	17	18	9
benlabs sassena	8294	43	70	17	18	125
pkok bsc-pga	49849	203	49	22	23	6
Olga-Yakovleva RHVoice	15649	276	121	41	45	104
PMBio peer	150917	6272	5879	1067	1887	1739
H4311 Projet-Grammaire-Langages	1783	245	105	20	19	6
Ratstail91 Sketch	825	6	3	1	1	29
vinzenz vsqlite–	546	42	9	5	4	25
Robnocop parlevision	4542	50	20	5	6	35
renxi-cu srs_public	14240	160	166	57	50	16
Kazade kazbase	1703	55	56	16	16	31
kmx mirror-iup	78405	86	10	5	5	52
peterwittek trotter-suzuki-mpi	2466	0	0	0	0	1
pelican pelican	3230	89	49	14	19	3
daisukekoba sakura-editor-trunk2	24450	209	108	26	37	6
namecoin namecoin-legacy	12465	183	126	40	43	127
khalahan namecoin	7571	107	78	27	25	100
libgeos libgeos	40325	2093	2220	442	603	141
CSB-at-ZIB PARKINcpp	102879	2497	1025	190	257	282
vozbu libslave	2087	25	11	4	3	34
markusfisch PieDock	2983	117	8	2	2	39
flipcoder qor	4054	102	103	38	41	8
orlandoacevedo MCGPU	53222	1118	1149	168	304	14
timvdalen OGO-2.3	6619	2	3	1	1	2
inventos OpenHttpStreamer	1315	70	3	2	1	17
Danvil dasp	2688	4	6	9	2	10
zrax Plasma	181073	1617	1209	189	323	57
filipkunc MeshMaker	4130	0	0	0	0	3
guruofquality grextras	806	4	4	8	2	10
legnaleurc komix	537	0	0	0	0	2
airekans Tpool	2243	61	31	11	11	10
hmmr aghermann	6265	259	123	34	39	79
j0sh crtmpserver	30169	3	4	1	1	0
spring mingwlibs	6711	200	12	5	4	5
rug-compling dact	1255	24	8	2	3	5
pank7 pank7-test	11215	17	9	3	3	7
HEROES-GSFC SAS	13316	150	39	13	15	28
ehsteve SAS	13316	150	39	13	15	28
bachan coda	2636	10	4	1	1	4
rhdunn cainteoir-engine	15595	1024	294	80	81	75
mscdex node-mariasql	33008	0	0	0	0	3
jhasse jntetri	1752	15	13	5	4	5
Conedy Conedy	2178	9	6	3	2	17
spearse FSOM	5184	5	3	2	1	17
ppcoin ppcoin	14400	871	184	54	56	206
martinrunge muroa	22712	696	171	61	59	30

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
jhasse poly2tri	1106	0	0	0	0	1
jwatte robotcode	20260	144	96	19	15	145
Detegr pwskoag	634	2	3	1	1	4
guns rxvt-unicode	2521	9	6	2	2	1
kallisti5 sheepshear	31385	3	16	1	4	4
silvest HEPfit	53980	435	72	22	22	386
ioquatix kai	4946	41	37	9	10	42
kpu lazy	16197	188	61	25	19	5
storance dcpu16	4296	46	22	7	7	32
Cyberbeing xy-VSFilter	75457	1538	1273	193	334	29
NixOS patchelf	1001	10	8	2	2	7
cocaine cocaine-plugins	4528	210	146	67	64	34
kuzmas razor-qt	13681	95	7	3	3	3
iannix IanniX	14086	464	299	86	94	30
supergillis tibia-hook	791	25	5	2	2	8
fankee snigdha123	938	9	9	3	3	4
genome breakdancer	971	52	6	3	2	7
Foran Descent-Bot	1346	0	0	0	0	9
MiKom karstgen	625	62	43	7	11	20
ahorn libse	51279	1120	1145	166	302	14
rubenvb Ambrosia	0	3	0	1	3	15
OPM opm-upscaling	3800	2163	178	60	63	40
Tapsa genieutils	3587	68	19	5	4	14
gatgui gcore	11566	88	30	11	10	92
matthewfl ilang	4039	71	71	23	12	6
Loki-Astari ThorsSerializer	1907	0	0	0	0	52
nickrmc83 ioc_container	278	79	56	19	19	1
lantimilan topcoder	54815	87	24	3	3	7
StarvingMarvin llvmj	752	8	12	2	4	5
wg-perception linemod	221	0	0	1	0	1
Frederic-bioinfo rTANDEM	13648	44	7	2	2	6
tomka mitsuba-renderer	91774	896	216	52	61	11
easterbunny273 Project-Cube	6188	5	2	1	1	2
daviddoria PatchBasedInpainting	3679	0	0	1	0	32
sboli twmn	137	21	6	3	2	1
SimonWallner kocmoc-core	109339	18	21	4	6	15
evenator swri-ros-pkg	23500	16	24	14	8	0
yavdr vdr-plugin-restfulapi	4897	27	18	9	9	1
steinwurf gauge	381	3	3	8	1	8
godexsoft x2d	43199	486	959	233	447	24
uboot stromx	1265	24	17	13	6	4
mderezynski Youki2	17532	438	259	92	102	109
manitou-mail manitou-mail-ui	14142	1675	610	146	149	25
pixie16 pixie_scan	13943	263	100	10	14	68
pelyot Amicale-TD	3535	16	13	6	5	15
felipemontefuscolo FEPiCpp	24324	442	406	172	171	14
jameshanlon tool_axe	6858	9	3	1	1	2
AndreLouisCaron w32	6085	261	109	34	41	31
kjax Stroika	88332	443	345	76	97	169
ledyba Cycloa	3339	54	33	5	8	30
msoos cryptominisat	10440	87	75	20	25	8
cheetah0216 CodeRepository	992	7	3	5	1	2
vbeffara Simulations	7808	0	0	2	0	6
Nocte- hexahedra	18353	222	138	48	60	73
openigtlink OpenIGTLink	10974	15	10	3	4	2
cyclus cyclus	55829	151	103	37	37	119
ntoussaint Cardiac-Prolate-Spheroidal-ToolKit	0	18	0	18	18	6
moshbear mosh-fcgi	6155	132	46	23	15	61
santazhang sandbox	67176	268	268	69	84	33
libspatialindex libspatialindex	58288	2480	2250	296	542	457
bkloppenborg liboi	75128	1634	1600	247	431	30
JayDz PPP-answers	8977	1698	400	69	115	13
kallaballa Janosh	2322	231	98	18	26	2
InMobi scribe	2844	210	268	16	22	9
herumi xbyak	4274	251	134	35	52	7
samindaa RLLib	8478	14	3	7	1	1
kmaehashi jubatus	4300	331	155	43	61	7
Slicer SlicerExecutionModel	2170	291	30	8	10	3
rug-compling alpinocorpus	2229	151	178	65	59	62
FernetMenta xbmc-pvr-addons	41795	245	21	7	7	7
plfs plfs-core	14906	240	119	32	59	49
apache xerces-c	21862	792	884	136	233	28
ros ros_comm	20644	121	393	41	37	8
ldmt-muri alignment-with-openfst	5333	15	9	12	3	6
frankyeh DSI-Studio	9727	115	19	6	7	2
deeplearningais CUV	5315	0	0	7	0	6

	LineCount			Occurrences		
Project	Total	try	catch	try	catch	throw
Kangz epyx	6934	293	54	22	19	68
snaewe omniorb	73717	2372	1799	304	490	522
wg-perception tabletop	178476	2204	881	269	268	649
chrispap TKN	1296	2	4	1	1	15
dmbryson apto	17232	3	9	1	3	3
rrnntt SmallProject	61741	1213	1228	184	328	18
ulrichard ftgl	3620	184	14	3	3	9
rafewenger sharpiso	0	57	0	29	57	6834
ctSkennerton crass	6657	935	358	71	102	88
okard depot	2735	0	0	0	0	21
tacaswell tracking	8218	750	462	110	145	255
LibRaw LibRaw	15071	2	2	1	1	34
x37v datajockey	4336	211	73	29	32	6
lballabio QuantLib	19526	462	125	36	52	3
starpos ioreth	897	47	18	4	6	9
william0wang meditor	36492	272	7	1	1	0
cjacoby libmanta	5019	139	98	19	32	18
dimock chess	6758	7	4	2	2	0
barak djvulibre	34302	1869	812	138	138	2
plasmodic ecto_pcl	407	0	0	7	0	3
myint perceptualdiff	492	90	18	2	4	4
GNOME gnote	6907	356	214	79	88	9
PrinceCreed TrilliumEMU	222162	957	174	38	51	170
XhmikosR notepad2-mod	41417	300	34	10	13	8
zigarrre asteroids	638	0	0	1	0	1
andrewfenn Hardwar	59336	1352	1660	395	472	18
it-workshop UniSched	1309	46	56	19	20	14
martiert Pandora3D	77434	1490	1561	221	412	14
jwmatthys rtcmix-in-pd	113909	0	0	0	0	1
cjlano eliot	7433	354	135	50	53	12
falkTX dssi-vst	2534	143	72	18	21	40
spinos aphid	64955	159	101	40	37	7
adegroote hyper	13024	202	84	24	17	15
GNOME gparted	13904	28	26	6	10	6
BartVandewoestyne Cpp	7753	83	46	17	16	13
smistad SIPL	635	0	0	0	0	9
w-bamberger CPProb	4750	158	147	22	23	10
oniko ok-snap	3072	64	54	11	25	41
guruofquality gras	1955	49	39	19	14	8
n319 xPL	7047	16	9	4	4	6
ccrma chugins	9573	0	0	0	0	1
toddsundsted stunt	33727	269	51	15	15	5
martinhaefner simppl	1004	18	14	8	7	2
telnet2 gradworks	83154	2160	1066	280	326	36
schnorr pajeng	3960	42	6	3	2	125
patrickfrey textwolf	430	107	11	3	3	5
salilab rmf	6713	361	193	53	45	1
striezel pmdb	2612	158	69	19	19	3
zakinster detiq-t	7967	59	24	7	10	108
zakinster eiimage	3863	31	23	8	8	38
bakwc Epsilon5	8780	148	73	29	29	15
m1kc mkvtoolnix	20545	1179	133	51	52	39
ianj-als mosesdecoder	48164	433	129	47	40	75
chazmatazz proto-mirror	12942	22	27	4	7	20
t-crest patmos	22263	249	49	10	9	17
ushakov mapsoft	22767	366	142	54	53	41
Revolutionary-Games Thrive	1836	9	11	11	4	27
nodakai exp	7105	7	6	8	2	38
plasmodic ecto_ros	217	0	0	7	0	1
GraphicsEmpire FemBrain	125931	34	27	11	10	94
rofl0r exult	55000	630	236	75	72	39
kudkudak Growing-Neural-Gas	1186	13	13	5	3	8
jkovacic math	49591	6098	1433	435	463	2089
Zguy ProtoZed	1282	0	0	0	0	10
openwebos libpbnjson	10113	39	3	2	1	2
hfiguiere libopenraw	9405	168	95	21	24	21
makerbot json-cpp	4088	23	7	2	2	0
Thomas1205 RegAligner	7519	0	0	0	0	4
PacificBiosciences ConsensusCore	7969	21	22	5	6	12
vancegroup util-headers	23687	551	526	139	138	2
thp numptyphysics	17218	2	2	1	1	2
mta1309 mulberry-main	8236	101	43	17	17	14
tclarke opticks-extras-Spectral	12658	292	56	27	17	2
apache activemq-cpp	24998	3276	2454	1069	1150	24
mungerd latbuilder	4464	94	33	13	11	37
pmiecio Smart_game	2060	0	0	0	0	3
chikuzen avs2pipemod	0	7	0	4	7	6

	LineCount			Occurrences		
Project	Total	try	catch	try	catch	throw
0xfeedface grdfs	1875	46	79	10	10	23
inducer meshpy	35412	4	8	3	3	3
aparent QCViewer	4739	78	46	13	14	27
nazgee libosock	2533	41	15	13	6	37
sholsapp gallocy	53958	4	6	5	2	1
salvestrini haze	1183	27	76	3	5	15
reenigne reenigne	42643	459	81	41	35	1
jirkamarsik trainable-tokenizer	1096	15	16	12	4	3
quietfanatic rata	0	1	0	1	1	3
pal-robotics perception_blort	27913	11	14	12	5	285
pavlinux ctorrents-plx	9653	5	4	1	1	11
mrtazz restclient-cpp	48121	1115	1153	167	304	17
keithw sprout	2342	61	13	5	4	2
metno wdb-feltload	588	64	41	13	16	12
vinniefalco LuaBridgeDemo	132891	132	42	21	17	12
fengwang random_variate_generator	6449	161	117	27	30	31
rawler bithorde	4021	60	42	25	14	17
nschloe nosh	4417	159	36	13	15	3
araqnid pqwx	1820	0	0	0	0	2
rdanbrook nestopia	54604	1600	357	54	109	94
Overdrivr ZNoise	909	0	0	1	0	5
thjaeger easystroke	2573	36	24	8	7	5
gingi fastbit	138489	6513	3451	345	553	419
squeesh hex-grid-test	1070	84	21	7	7	15
sequencing gvcftools	3135	118	168	36	45	11
yetanothergeek fxscintilla	34949	43	28	8	9	6
ahamez libsdd	3339	61	60	30	30	5
westlab negi	48266	1135	1160	168	305	11
artm WatchThatSound	1112	0	0	0	0	2
veprbl libepecur	2768	27	23	8	7	44
dln medida	43879	1104	1121	163	296	14
vecna sniffjoke	3794	47	22	7	7	0
sim82 papara_nt	5245	0	0	7	0	52
dataseries DataSeries	18285	65	48	15	16	0
lemire EWAHBoolArray	2138	0	0	0	0	2
mnmlstc unittest	843	167	391	46	79	3
xrubio pandora	13941	359	149	42	48	38
uentity bluesky	5808	93	54	22	20	24
lucas8 Cancer-game	1626	17	12	2	4	7
lettis Kubix	2458	15	8	2	3	15
dicarlolab-mworks NIDAQ	531	5	16	9	6	2
matiu2 witty-plus	365	8	7	8	3	2
aconley pofd_affine	14599	985	254	57	70	79
pwr Sigil	58653	335	113	42	41	10
codemer libtpt	0	18	0	8	18	16
paulasmuth brokerd	3393	85	48	13	13	2
lwinkler markus	3790	200	115	33	42	4
gunoodaddy SharedPainter	7276	77	6	11	3	4
SysFera libdadi	5061	607	468	106	99	2
nireis pferd	6716	0	0	0	0	4
i-rinat reiserfs-defrag	2566	73	14	2	4	7
glipari rtscan	2831	207	268	71	74	1
jasonmccampbell scipy-refactor	164751	0	0	0	0	2
mateuszboryn DisCODe	1638	65	62	27	23	2
jezhiggins arabica	1654	24	25	7	9	2
sim82 ivy_mike	989	6	5	1	1	27
jfnavarro PrimeTV2	5667	236	60	10	16	54
emeryberger Heap-Layers	989	0	0	0	0	4
lotten daoopt	12531	154	8	2	2	5
ericprud SWObjects	39445	9312	4329	394	1024	96
Grumbel viewer	4726	2	3	2	1	10
bytemaster tornet	2777	270	130	50	59	11
striezel libstriezel	6458	504	99	28	30	35
dreamsxin Gnoll	2874	87	88	35	29	12
mpusz Condor2Nav	1147	48	29	12	12	4
feelx88 Explore	2069	24	20	8	7	4
CppMicroServices CppMicroServices	29605	1876	696	167	246	44
Error323 E323AI	3952	0	0	7	0	2
ethz-asl libpointmatcher	12295	169	170	41	52	101
sebjameswml futil	2889	87	49	18	18	38
marcovc casper	5079	12	14	6	5	28
metno wdb-libwdbload	211	21	16	9	8	1
zerebubuth openstreetmap-cgimap	5470	335	240	60	76	130
codespear GameEx	3370	25	30	6	12	24
tclarke opticks-extras-IDL	2485	77	14	6	6	9
sarum9in DEPRECATED_yandex_contest_invoker_flowctl_game	511	25	11	6	7	1

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
stnava ANTs	0	107	0	101	107	22
guyrt WFUBMC	52099	1567	2093	300	592	650
gitj dspsr	16914	2135	308	103	122	1
ibd1279 logjammin	2850	93	100	30	42	10
goodfella libtq	361	25	16	4	4	5
mathieu dtEntity	25603	1319	1680	381	468	8
CyborgSummoners Summoner-Wars	2909	28	17	5	5	39
wedesoft hornetseye-ffmpeg	802	257	54	26	25	2
britram libfc	4840	10	18	5	5	16
ibsh libKeyFinder	858	0	0	0	0	37
segfault87 Konstruktor	4815	17	35	7	7	3
mgbellemare Arcade-Learning-Environment	21967	477	312	52	104	10
paoloambrosio cucumber-cpp	2023	69	42	16	14	12
bunsanorg common	4230	390	667	53	165	48
shumatech BOSSA	3200	234	57	13	15	83
graehl carmel	3562	407	15	5	5	2
Ethatron squish-ccr	33772	522	12	4	4	4
Oberon00 luabind	3665	635	643	311	311	10
amorilia formast	755	8	9	3	2	4
byon myrrh	4119	118	92	37	30	8
Yubico yubikey-personalization-gui	3292	145	18	6	6	22
olavolav te-causality	5323	1479	1278	84	145	0
couchdeveloper JPJson	5238	18	8	4	3	7
sedna sedna	51746	494	321	62	115	20
PMBio limix	39473	14	16	2	2	17
arktools arkmath	819	0	0	6	0	1
eNoise pichi	27677	23	27	12	9	9
ghedo p5-FFI-Raw	16066	9	9	3	3	3
jean-marc objrdf	2308	6	9	2	4	31
fclaude libcds2	46185	1144	1046	166	299	24
vancegroup stlport-avr	33691	1434	907	140	248	63
ruven iipsrv	6622	249	88	10	14	69
bekaus pgmlink	1200	0	0	1	0	19
tonttu Shaderkit	166819	1746	750	230	229	572
pkarasev3 kslice	14075	34	34	14	13	21
dascandy hippomocks	1480	100	240	28	30	4
schwa423 Sketchy	5519	13	19	5	6	11
godai0519 BoostConnect	1121	6	3	2	1	10
OpenWaterAnalytics epanet-rtx	8659	123	54	28	21	7
ros nodelet_core	619	5	15	10	3	3
timowest flauta	3935	3	3	1	1	2
colobot colobot	9506	36	50	15	20	8
yangacer BehaviorDB	1547	15	19	7	7	13
jehugaleahsa spider-cpp	1021	82	28	13	8	3
HongjianLi igrow	541	52	4	8	1	1
esrf-bliss Lima	11138	256	110	30	37	73
imvu-open istatd	15826	563	232	65	63	82
James-Jones HLSLCrossCompiler	177533	1751	760	231	232	581
svn2github vmpk	66205	1380	224	94	95	8
ramntry homeworks	3837	71	20	5	5	15
vancegroup-mirrors eigen	17718	458	412	175	174	1
fperrad tvm	14289	87	53	9	9	8
gman0 fsync	1322	9	11	4	5	6
AoD314 pat	184	8	3	1	1	5
hfiguiere niepce	2914	101	76	29	30	1
victorparmar zsearch	40249	19	10	4	5	7
osh gr-eventstream	915	0	0	7	0	11
steinwurf gtest	39911	1112	1144	166	301	11
WrinklyNinja libloadorder	619	94	81	29	31	12
Amxx MDMA	5397	29	16	5	5	6
neuront stekin	1645	3	3	1	1	9
jafyvilla vrpn	35112	178	35	17	16	15
BrewPi brewpi-avr	7274	3	9	1	3	3
luispedro elgreco	1341	0	0	1	0	9
pgengler pinot	11100	0	0	0	0	1
alopatindev ponic	15302	0	0	0	0	6
nebirhos yaml-cpp	7051	108	39	18	19	54
android platform_external_protobuf	142787	354	254	52	73	7
Yubico yubikey-personalization-gui-dpkg	3265	145	18	6	6	22
reverbrain elliptics	22383	618	393	70	80	27
rdnelson Libra	77163	1617	1574	239	421	19
tanjeff agentXcpp	784	9	12	3	5	33
glastonbridge SuperCollider-Android	38234	49	36	9	17	7
ros-perception vision_opencv	1345	15	13	6	5	5
Nocte- rhea	2972	58	61	11	10	18
jinmei queryperfpp	1168	102	14	6	5	5
JerrySievert plv8	121	8	6	1	2	3

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
avakar libyb	3181	120	66	24	25	87
Wicker25 Rpi-hw	1761	0	0	0	0	15
cbsrbiobank dmscanlib	22981	26	26	5	8	13
donut-lang Chisa	14074	26	41	5	12	10
scyptnex computing	10384	107	15	4	3	19
ros bond_core	379	43	40	16	10	11
ros-perception image_pipeline	1764	116	57	31	24	5
evido wotreplay-parser	5719	17	15	4	4	10
danopernis hcc	5394	46	23	5	5	30
zsiciarz aquila	15020	2479	6508	777	1632	11
fhoefling halmd	5346	334	300	76	77	22
fhoefling h5xx	686	111	100	26	26	1
m-mcgowan brewpi-avr	7757	3	9	1	3	3
worldforge metaserver-ng	1134	207	18	8	7	2
Alphax nifskope	26469	208	25	9	9	1
cdunn2001 jsoncpp	7329	134	133	44	44	2
psoetens orocos-rtt	27649	422	465	103	129	12
volkszaehler vzlogger	8701	1082	665	138	225	130
DamianZaremba cluebotng	287	35	9	10	3	5
uesp tes5lib	11457	0	0	0	0	1
jlaire dlx-cpp	2568	33	23	3	3	40
tsvaton hermes	56601	555	135	37	49	534
openscenegraph VirtualPlanetBuilder	33779	98	24	4	6	56
sofianehaddad privot	93172	14785	2093	432	532	1435
openstreetmap merkaartor	22745	16	18	6	7	1
d3cod3 GAmuza	33578	120	50	17	18	0
kripken intensityengine	111654	28	29	19	13	7
tmolteno necpp	12491	282	281	77	80	68
Flusspferd flusspferd	7832	648	663	156	172	193
crishoj OpenPNL	150705	1088	846	119	212	1
borisbrodski sevenzipjbinding	139850	3504	1022	187	334	292
Medo42 Faucet-Networking-Extension	1196	49	32	23	16	1
slowfrog chickenpix	6412	28	26	9	11	19
o11c tmwa	126626	730	519	108	139	99
themiwi OpenFOAM-1.7.x-OSX	75733	51	36	15	12	4
exavideo exacore	5153	71	39	14	13	68
RealBadAngel minetestHD	47968	1004	695	211	304	13
ledger ledger	14644	811	365	96	97	26
raceintospace raceintospace	37783	3	11	2	3	8
BizarreCake hCraft	21275	276	159	56	55	83
NoLifeDev NoLifeStory	2292	0	0	0	0	35
larsnystrom alma	11972	28	23	3	6	20
snogglethorpe snogray	9963	54	22	13	9	30
jetty840 Sailfish-G3Firmware	71125	635	508	52	73	4
mateuszbaran ECG-analyzer	70917	3377	1859	625	618	975
ThQ memc	4299	3	9	1	3	3
ilpincy argos3	575066	19444	7544	90	92	0
iut-ibk DynaMind-Extensions	43123	1104	1121	163	296	14
vle-forge vle	8490	252	118	48	44	6
pronobis rocs	76721	2446	1902	282	418	31
mcvsama xefis	5332	123	82	31	37	8
YggdrasiI TinyPrint	2949	18	16	6	5	6
MicroMagnum MicroMagnum	3453	0	0	0	0	81
wichtounet gooda-to-afdo-converter	2550	120	10	4	3	16
rorywalsh cabbage	133344	327	399	56	190	6
pmiddend fruitcut	2068	16	8	2	2	16
jbarreneche 75.74-Aeropuerto	6448	733	416	92	160	44
sakrejda Lux	220	0	0	0	0	4
c42f displaz	21266	211	157	34	32	11
allan-simon tatowiki	602	0	0	0	0	1
sdressler EPerF	531	24	57	8	11	9
cvjena nice-core	3773	15	16	6	6	5
guruofquality PMC	1366	281	23	5	4	6
slra slra	1653	160	27	5	5	31
romankuznietsov phyz	697	17	9	16	3	1
pr061012 pr061012	5027	40	22	7	7	16
robertramey safe_numerics	68358	221	187	48	52	2
AeroNotix freepoint	260	8	13	3	4	2
wichtounet btrees	1304	0	0	0	0	1
ros-planning navigation	8428	98	84	33	31	4
fabianschuiki Auris	3219	112	56	12	21	31
pjgrenyer aeryn	2336	435	393	79	128	18
bkloppenborg ccoifits	73270	1626	1592	243	427	30
daritter OpenMPIFitter	880	53	29	9	9	2
performous performous	10348	496	171	67	66	153
r-lyeh moon9	216266	1284	688	142	181	148
HoverRace HoverRace	35659	686	1142	287	514	90

	LineCount			Occurrences		
Project	Total	`try`	`catch`	`try`	`catch`	`throw`
icecc icecream	7093	198	51	7	8	7
monocasual giada	7984	60	62	18	18	1
msiemens UH-INF-StrategicGame	682	6	2	8	1	1
lvmguy thin-provisioning-tools	3381	63	32	12	11	26
rtrepos vle	8490	252	118	48	44	6
sebkur swp12	1090	0	0	0	0	4
limhyon Pangolin	4855	3	3	8	1	9
dannyedel dspdfviewer	627	16	12	4	3	7
felipealmeida mORBid	1681	0	0	7	0	5
xypron skyldav	1674	79	25	10	10	18
forsaken1 compiler	2284	45	2	1	1	76
april-org april-ann	31521	6	13	6	4	4
erikreed HadoopBNEM	35719	1339	810	119	118	38
iHateInventNames synergy-through-usb	95623	1769	1755	257	451	192
ros diagnostics	7835	10	20	9	4	5
ytakano catenaccio_dpi	2834	21	8	8	2	13
Team3512 DriverStationDisplay	1461	0	0	0	0	2
invor space-lion	10224	0	0	0	0	32
wingfiring xirang	4950	93	86	24	29	1
rr- CRC-manipulator	467	7	6	2	2	7
tysonite asn1-compiler	12815	1512	1402	352	352	34
pellegre libcrafter	5903	0	0	0	0	23
k-stachowiak space-shooter	2230	6	3	1	1	24
systemdatarecorder recording	14818	0	0	0	0	6
chrislu schism	10572	101	104	34	26	39
ros-gbp shape_tools-release	33	0	0	0	0	1
rhaberkorn sciteco	3258	86	116	30	35	30
lheckemann licht-raiders	332	2	9	2	1	1
pisto laurea	980	199	63	9	11	19
patrickdemond Alder	0	4	0	4	4	36
SuperV1234 SSVOpenHexagon	1027	0	0	13	0	5
anttisalonen freekick3	1187	24	21	5	7	2
jktjkt trojita	14963	53	36	16	16	16
bsutton openscad	10394	36	23	6	5	2
vslavik xmlwrapp	0	32	0	33	32	28
manucorporat FORZE2D	22353	51	64	14	14	8
pemryan Ipopt	10238	240	116	18	38	77
YoruNoHikage CaSFML-Defender	857	32	15	7	7	2
Krigu Gaze	2053	19	16	3	4	6
vitei moon	7125	59	220	18	19	44
doomtech gzdoom	198891	543	155	24	28	50
ralph-mcardell dibase-rpi-peripherals	2011	218	63	15	17	74
naoyam physis	12348	3	9	2	3	3
Morwenn POLDER	7008	43	8	3	3	23
studentls MolSimDS	2922	5	5	2	2	3
TheWatcher twscript	1801	3	6	1	2	2
brucekarsh PhotoSelect	3483	22	46	16	13	1
statismo statismo	0	42	0	44	42	6
Oxyd APNS	2687	66	18	7	3	22
maeikei xclang	13948	79	10	4	4	0
Neoracle DymonRepo	3352	20	12	6	6	31
dirkm cheapshot	1970	63	21	8	7	9
ptroja orocos-rtt-qnx	23875	288	304	69	86	8
shlagbaum quantlib	14503	342	89	26	37	1
sasvariagoston SG2PS	17470	14	49	3	13	17
wuye9036 SalviaRenderer	50787	0	0	7	0	1
visualfc liteide	38888	54	18	8	8	4
evincarofautumn protodata	346	13	12	3	4	2
zbigg tinfra	2808	8	13	3	6	10
djeedjay BoostTestUi	1680	22	14	12	5	11
wibus MouvementDeMasse	904	6	5	1	2	3
rhcad vglite	14281	0	0	0	0	23
skelcl skelcl	3898	118	105	17	19	10
riemervdzee TrafficSimulator	13174	65	34	4	4	32
lfranchi libechonest	2615	22	8	4	4	39
whudson The-Game	605	3	5	1	1	5
TyounanMOTI ARD	42893	1104	1121	164	296	14
hirisc m2dec	17257	0	0	0	0	1
fraunhoferipk tuiframework	10164	183	107	36	41	2
mirsoleimani SAMMicrobenchmark	12030	206	125	37	42	5
kloper openvrml	2932	175	55	24	20	2
kloper scooter	1413	16	8	10	3	4
meshula LabSound	7745	4	6	2	2	29
etano library	52066	3	15	2	2	2
bchareyre trunk	9207	19	18	11	5	38
yast yast-pkg-bindings	3490	442	228	93	100	5
code-canvas webapp-xul-wrapper	5720	167	40	9	13	18

	LineCount			Occurrences		
Project	Total	try	catch	try	catch	throw
maxdebayser SelfPortrait	4654	150	9	3	4	30
ripples paol	2704	2	2	1	1	1
springlobby springlobby	10624	440	171	84	79	4
reverbrain elliptics-fastcgi	692	83	34	11	13	10
dogbert2 bro	48581	88	54	21	21	1
pgerdt timed	6868	70	44	5	10	13
soundcloud barn	118566	2378	2460	353	633	14
PaulPrice healpy	7919	0	0	0	0	1
bbellon nbites	13601	85	69	30	29	29
AnomalyDetection2012 AnomalyDetection2012PWR	2371	65	10	3	3	1
jeeb avisynth	30365	1199	360	95	102	180
cpaproth sk	1497	75	21	20	10	6
janpaulus BRICS_OODL	4016	165	137	39	48	17
vecnatechnologies navigation	7212	101	78	30	28	6
simsong be13_api	1519	12	11	4	4	2
ddemidov amgcl	1811	33	6	3	2	2
stonier ecl_core	4051	258	174	77	77	1
cristal StrawberryCore_Old	189556	60	31	9	9	102
funnyfan c10t	1968	41	61	19	18	11
zhangchn sunpinyin	13126	39	13	2	2	3
DigitalInBlue Celero	46019	1131	1169	170	308	14
dakeyrasKhan gravityBot	1749	9	9	3	3	5
mpreisler ember	37442	729	462	171	185	11
el-bart avr_servo	761	73	36	5	7	2
amate unDonut	572	68	13	12	5	4
mohammadzakwan inetmanet	19068	0	0	0	0	4
nanoant Catch	9940	2149	2107	685	688	9
schmunzel mmoserver	11716	81	41	27	15	7
FrankPetrov Vault-Tec-Multiplayer-Mod	37787	451	79	17	17	112