CODE EXCHANGE

# Abstract:

In the current age of technology it is important for Software Engineers to share their code that they have written with other developers, they try doing this using social media applications such as Facebook or WhatsApp, which don’t preserve the format of the code or they use GitHub gists for this, although GitHub gists is good and allows people to share their code, it doesn’t allow people to ask what went wrong with their code.

Programming today regularly involves searching for source code online, whether through a general search engine such as Google or a specialized code search engine such as SearchCode, Ohloh, or GitHub. Searching typically is an iterative process, with developers adjusting the keywords they use based on the results of the previous query. However, searching in this manner is not ideal, because just using keywords places limits on what developers can express as well as the overall interaction that is required. Based on the observation that the results from one query create a context in which a next is formulated, we present [CodeExchange](http://codeexchange.ics.uci.edu/), a new code search engine that we developed to explicitly leverage this context to support fluid, expressive modifications of queries.

Sometimes, when programmers use a search engine, they know more or less what they need. Other times, programmers use the search engine to look around and generate possible ideas for the programming problem on which they are working. The key insight we explore in this dissertation is that, in the latter case, the results found tend to serve as inspiration or triggers for the next queries issued. We introduced search engine.

CodeExchange which is specifically designed to enable the user to directly leverage results from a previous query in formulating a next query.

CodeExchange does this with a set of four features that enable the programmer to use characteristics of the results to find other code with or without those characteristics. For instance, by selecting characteristics of the results the programmer likes (e.g., libraries used or method calls) or dislikes (e.g., code complexity or size), the programmer can refine their query for results with or without those characteristics.

We evaluated the impact of CodeExchange on the experience, time, and success of the code search process. We compared our iterative approaches with two approaches not explicitly supporting iteration, a baseline and Google, in a laboratory experiment among 24 developers. We found that search engines that support using results to form the next query can improve the programmers’ search experience and different approaches to iteration can provide better experiences depending on the kind of task. The main contributions of this dissertation are six-fold.

# Introduction:

Programming “amounts to determining in advance everything the computer will do” . The process of determining what a computer will do has been likened to the work of an “architect, a composer, or a writer” , but where the programmer translates their ideas into source code and iteratively modifies their creation to meet various criteria . Some criteria could be aesthetic (e.g., look and feel of the program or its source code), functional (e.g., what the program does), or non-functional (e.g., how fast the program runs). The ideas that are programmed depend on the situation. If the programmer is in the position to create software to their liking, then they have full control of what ideas are programmed (e.g., Linus Torvalds and his first release of Linux).

Alternatively, the programmer might work in a company that has teams of various sizes addressing a variety of projects, and the programmer’s ideas are scoped by the project they work on, modified to 2 accommodate other team members’ ideas, and narrowed by the task to which the programmer is assigned . Or, the programmer might join an open source project, where the project scopes the ideas, but the programmer can freely choose what part of the project they want to enhance or fix . Regardless of situation, and influence of team and project, what a developer programs is never fully specified – they still must decide what to program specifically.

Deciding what to program is often a problem solving process. This entails making choices, such as what the architecture, algorithms, and data structures should be. Different needs can drive these decisions. For example, if the program needs to evolve, then modular approaches for architecture might be better. As another example, depending on the data structures chosen, some algorithms might work better or worse (e.g., Radix Sort can sort integers and strings faster than other algorithms, but it cannot be used for certain other types of data structures).

When programmers engage in problem solving, it has been observed that they might sketch on a white board ,talk with colleagues , or just think. They might sketch an architecture or algorithm to examine it or use it as a focal point of discussion. They might talk with colleagues to get their insight or experience on solving a problem. Or, they might just think about their problem by mentally simulating different solutions, considering alternative implementations, reflecting back on previous solutions they have used, and so on. 3

It has also been observed that programmers search for previously written source code. When programmers search for source code, they look for code by matching it with the problem being faced. Sometimes the code is matched by remembering that it was used to solve the problem in the past. Other times, programmers match source code found by making analogies between the problem the source code solves and the current problem they face.

If the source code can be reasonably adapted by the programmer to solve the current problem, then the programmer will often use that source code. This dissertation focuses on the code search activity, and particularly seeks to: (1) understand how developers search and (2) help them in this task. How programmers have searched for code has evolved with computers and software. In the 1970s, programmers would regularly search using command line tools, such as grep, to search through their files for some particular source code .Grep takes a regular expression query to find files that have lines matching the query.

While the use of grep was helpful in many instances, it is limited to searching local code that is authored by the programmer or their team and accessible on the local file system. Searching source code by other authors elsewhere cannot be done with grep, which is why programmers often would subscribe to magazines, such as Creative Computing or Compute! , that would regularly publish all sorts of source code, including games, AI algorithms, or graphical programs.

When a programmer is searching for code and is not quite sure what they are looking for initially, different aspects of the results returned for a first query might trigger the programmer to modify their query to attempt to incorporate one or more of those aspects. For example, suppose a programmer, Suzie, is looking to build a new adventure game, but she is not certain about how to get started. She wants to see some examples of how to implement characters and how to handle updating game state, but also wants to discover what other people are doing to get ideas for her own game. To do so, she opens her favorite search engine and issues the keywords adventure game.

The first code result she sees implements an Adventurer character that extends a Character parent. Recognizing she will probably have many characters in her game, she thinks extending a Character parent is a good idea and wants to see some more examples of how this is done, so she refines her keywords to adventure game extends Character.

Among the results returned, she notices that 67 there are some classes that extend the Character parent and they all tend to override methods with names starting with “draw”. Upon inspection, she sees each class is responsible for rendering its character graphically on the screen. She thinks this is a good idea, because it makes use of polymorphism, so she could write an update loop that simply iterates over all the characters, regardless of subtype, and call their draw method. To get a better idea of how to render graphics on the screen, she tries a next query adventure game extends Character draw graphics 2D to get to more examples of rendering graphics for adventure games. After searching deeply into code related to characters, Suzie still wants to look around and get some other ideas on how to build adventure games, so she hits the back button several times to return to the original results from the query adventure game. As she scrolls through these results, she discovers code for a text adventure game.

She immediately stops and has to think about this, because she had assumed before searching that her adventure game had to be graphical, but upon seeing a text adventure game result, she discovers a new possible direction for the design of her game. To learn more about text adventure games, Suzie refines her keywords to text adventure game, and once again begins to learn, modify her query, learn some more, and so on, until she is done. The example of Suzie is not unusual and represents a case where, as the programmer examines the results, certain aspects of the results trigger the programmer to reflect and issue modified queries that attempt to encode those aspects as keywords. For instance, as with Suzie, a programmer must add some keywords to focus on code containing desired 68 structures, methods, or on a library they may want to explore further.

They may also, however, choose to remove some keywords to backtrack or modify some keyword to somewhat shift the focus of the search. Regardless, programmers frequently create next queries in response to the results of a previous query. CodeExchange, our first experimental code search engine, was specifically designed to aid the developer when, initially, the programmer is uncertain of exactly what they are searching for and is engaged in a more exploratory search involving the submission of multiple queries through which to explore what examples may be available. In such a search scenario, the insight behind CodeExchange is that the next query tends to be relative to the results, and often to specific aspects of the results of the previous query.

As such, CodeExchange supports the developer in forming the next query by letting them construct it out of aspects of a result (e.g., method calls, parent class, or complexity), rather than trying to encode it as keywords. In this way, CodeExchange changes how a query is constructed by the programmer, from entering just keywords, to one that is created incrementally out of aspects of the results from each query. CodeExchange supports the user with four specific features for creating a next query.

Two of the features support refining the query by aspects of one of any of the results returned. Specifically, language constructs support the developer in selecting structural characteristics of a result (e.g., method calls, interfaces implemented, or code imported) to bring those characteristics into the query. Using a language construct yields a query that is a mix between keywords, if they were a part of the query before selecting a language construct, 69 and characteristics of results.

Unlike keywords, that may or may not retrieve code matching a topic described by the keywords, a language construct constrains the query to retrieve code exactly matching the characteristics specified by that language construct. For example, if the programmer notices a method call of which they want to see more examples (e.g., different ways of parsing parameters from an HTTP request using a method) or wants to see more examples of using a particular library, then they can click on that method call or an import statement to add it to their query. All code returned will have the selected method call or import statement. The second feature, critiques, supports the developer in selecting the value of different technical qualities (complexity, size, number of imports) of a result as a lower or upper bound to bring that bound into the query to constrain the next set of results.

In this way, if the developer feels a code result is lacking (e.g., too long or not complex enough), they can bound the next set of code results to attempt to avoid that quality. In contrast to modifying the query relative to a specific result, query refinement recommendations (the third feature of CodeExchange supporting iteration) presents the user with common aspects (imports, parent classes, or interfaces) or domain related terms across all the results to add to the query.

The recommendations help make visible to the programmer common aspects of results that are difficult to infer just from the top results that are actually visible. For instance, if the third result uses a particular library and is the only result among the top ten to do so, yet hundreds of other results not visible also use that library, then a query refinement recommendation is likely to present that library to the 70 programmer, making its common appearance visible.

After adding a recommendation to the query and getting the results, the recommendations are updated again using the newly returned results. In this way, the programmer can iteratively search by continually selecting recommendations.

For example, if the developer issues the keyword query chess, they may receive refinement recommendations for keywords pieces and move and parent class ChessPiece.

When the recommendation for parent class ChessPiece is chosen by the programmer to add to the query, further recommendations are returned, one being the keyword piececolor.

The final feature, query parts, modularizes the programmer’s query each time a programmer adds to a query. Whether by providing one or more new keywords or using one of the new features of CodeExchange, the addition is separately identified by CodeExchange in its interface.

Each query, then, consists of a set of separate parts that, together (logically AND’ed), form the actual query issued, but to the programmer remain individual components. Query parts leverage this by enabling a programmer to turn off / turn back on each of these parts separately. In this way, after a programmer gets new results, they can respond by trying different combinations of their query parts to search in different “directions”. For example, if the programmer issued a query comprised of three query parts, “adventure game” (initial keywords), “import java.awt.Graphics2D” (a characteristic added with a recommendation), and “method call playSound()” (a characteristic added with a language construct), but has decided they want sound in their adventure game, but 2D graphics are not required, then they can toggle off the query part “import java.awt.Graphics2D” from their query to search for code that is about adventure games and 71 plays sound. At any moment thereafter they can return to previous results by toggling the query part “import java.awt.Graphics2D” back on. Alternately, if they decide to want to learn more about graphics and sound, they could toggle off the query part “adventure game”to find more code about rendering graphics and playing sound that may or may not be related to adventure games.

# Functionalities:

Allows developers to share their code.

Users can request solutions to their problems in the code .

Allows users to test and run their code before uplodaing.

Code can be accessed either by public or can be kept a secret.

Allows user to store their code snippets for furthur use.

# Existing system:

GitHub is a web-based version-control and collaboration platform for software developers. Microsoft, the biggest single contributor to GitHub, initiated an acquisition of GitHub for $7.5 billion in June, 2018. GitHub, which is delivered through a software-as-a-service ([SaaS](https://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service)) business model, was started in 2008 and was founded on Git, an open source code management system created by [Linus Torvalds](https://whatis.techtarget.com/definition/Linus-Torvalds) to make software builds faster.

Git is used to store the source code for a project and track the complete history of all changes to that code. It allows developers to collaborate on a project more effectively by providing tools for managing possibly conflicting changes from multiple developers. GitHub allows developers to change, adapt and improve software from its public repositories for free, but it charges for private repositories, offering various paid plans. Each public or private repository contains all of a project's files, as well as each file's revision history. Repositories can have multiple collaborators and can be either public or private.

GitHub facilitates [social coding](https://whatis.techtarget.com/definition/social-coding) by providing a web interface to the Git [code](https://whatis.techtarget.com/definition/code) [repository](https://searchoracle.techtarget.com/definition/repository) and management tools for collaboration. GitHub can be thought of as a serious [social networking](https://whatis.techtarget.com/definition/social-networking) site for software developers. Members can follow each other, rate each other's work, receive updates for specific projects and communicate publicly or privately.

Three important terms used by developers in GitHub are fork, pull request and merge. A fork, also known as a branch, is simply a repository that has been copied from one member's account to another member's account. Forks and branches allow a developer to make modifications without affecting the original code. If the developer would like to share the modifications, she can send a pull request to the owner of the original repository. If, after reviewing the modifications, the original owner would like to pull the modifications into the repository, she can accept the modifications and merge them with the original repository. Commits are, by default, all retained and interleaved onto the master project, or can be combined into a simpler merge via commit squashing.

Because GitHub is so intuitive to use and its version-control tools are so useful for collaboration, nonprogrammers have also begun to use GitHub to work on document-based and multimedia projects. GitLab is an open source alternative to GitHub.

GitHub products and features

GitHub offers an on-premises version in addition to the well-known SaaS product. GitHub Enterprise supports integrated development environments and continuous integration tool integration, as well as a litany of third-party apps and services. It offers increased security and auditability than the SaaS version.

# Implementation details

# Proposed system

# Prototype of our proposed system