Codifier: A Programmer-Centric Search User Interface

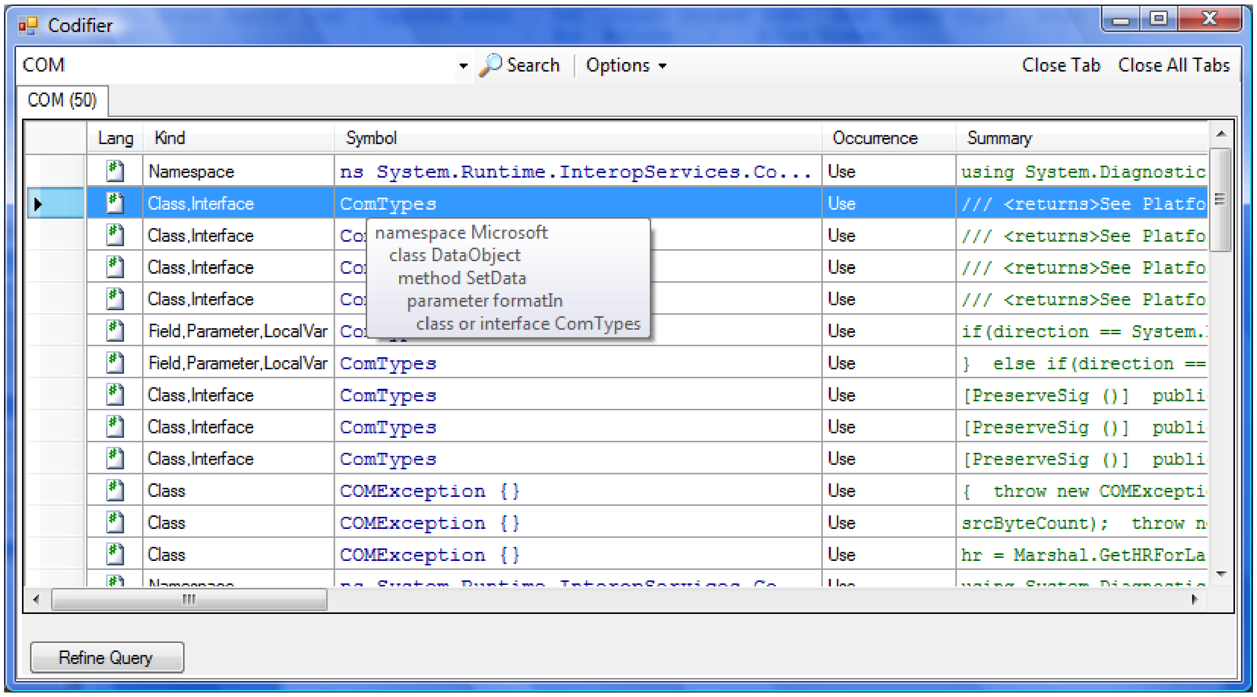
Andrew Begel
Microsoft Research
Redmond, WA 98052
andrew.begel@microsoft.com

Search tools have transformed knowledge discovery by exposing information from previously hidden repositories to the workers who need it. Search engines like Google and Live.com provide search capabilities via a simple one-line text query box, and present results in a paged HTML list. When the repository being searched contains structured information with extractable metadata (e.g. program source code), it can be advantageous to index the metadata and use it to enable queries that are more task-centric and suitable for an domain-specific audience.

Codifier is a programmer-centric search user interface that enables software developers to ask domain-specific questions related to programming languages and software. For example, developers might ask

- Where is this API or data structure defined?
- Where is this API used?
- Where is this variable assigned a value?
- I know this function writes data to the disk, but I forget exactly what its name is.
- Even if I spell it wrong, I still want to find IPersistentItemsChangedSink.
- Find all functions where Open() is called with Init().
- Show me all calls to this method, so I can refactor it by hand.

We index C, C++, C# and VBScript program source code using a modified compiler to extract and store lexical and syntactic metadata into a SQL Server 2005 or Windows Desktop Search database. The Codifier user interface, presented in Figure 1, enables software developers to search in this database for symbols found anywhere in the indexed source code (not just their definitions). Searches supported by metadata can be quite powerful. In additional to source code symbols, we can search for language-specific connectors (e.g. `foo::bar`, `foo.bar`,



COM

COM (50)

Lang	Kind	Symbol	Occurrence	Summary
	Namespace	`ns System.Runtime.InteropServices.Co...`	Use	`using System.Diagnostic`
	Class,Interface	`ComTypes`	Use	`/// <returns>See Platfo`
	Class,Interface	`Co`	Use	`/// <returns>See Platfo`
	Class,Interface	`Co`	Use	`/// <returns>See Platfo`
	Class,Interface	`Co`	Use	`/// <returns>See Platfo`
	Field,Parameter,LocalVar	`Co`	Use	`if(direction == System.`
	Field,Parameter,LocalVar	`ComTypes`	Use	`} else if(direction ==`
	Class,Interface	`ComTypes`	Use	`[PreserveSig ()] publi`
	Class,Interface	`ComTypes`	Use	`[PreserveSig ()] publi`
	Class,Interface	`ComTypes`	Use	`[PreserveSig ()] publi`
	Class	`COMException {}`	Use	`{ throw new COMExcepti`
	Class	`COMException {}`	Use	`srcByteCount); throw n`
	Class	`COMException {}`	Use	`hr = Marshal.GetHRForLa`
	Namespace	`ns System.Runtime.InteropServices.Co`	Use	`using System.Diagnostic`

```
namespace Microsoft
  class DataObject
    method SetData
      parameter formatIn
        class or interface ComTypes
```

Figure 1: Codifier search user interface showing a search for the symbol COM.

`foo->bar`), synonyms, homophones, abbreviations, concept keywords (e.g. searching for COM finds `COMString`, `ComException`, `ICOMPointer`), kind and usage of symbols (e.g. searching for definitions of methods named `WriteString` (kind:method usage:def WriteString), newly instantiated objects of class `IEnumString` (kind:class usage:use IEnumString), assignments to local variables named `firstTimeThroughLoop` (kind:localvar usage:assign firstTimeThroughLoop)), lexical scoping (e.g. searching for calls to method `Open()` in classes named `MemoryAccess`), and keywords for searching by programming language, source control information and file path.

Filtering, sorting and refinement capabilities are important for winnowing the thousands of answers resulting from a search over a large source code base. For example, Microsoft Windows 2003 Server contains several hundred million symbols in its source code – when searching for code, the "right" result may be one out of thousands, or may be *all* of them. Codifier provides support for filtering results based on the lexical, syntactic, and file path scope of the result. In addition, by presenting the results in a grid, with one row per symbol found, the UI enables sorting based on any of the metadata values. Refinement of queries is supported by metadata facet. A top 10 list of results is shown for each facet. When the user clicks on one of them, an additional filtering term is added to the query, which is then re-executed.

Codifier stores one symbol per row when using SQL Server 2005. Using as-of-yet unoptimized schema, each symbol's metadata is stored in about 2,300 bytes of space on disk, so even the largest bodies of source code we index fit into less than 500 GB. Indexing time is about 2 million symbols per hour. When using the less scalable Windows Desktop Search 3.0, Codifier stores all metadata for the symbols in each file in the inverted index, enabling metadata-based searches, but requiring reanalysis and extraction of metadata when each search result is retrieved. Indexing time with WDS is about 80,000 files per hour.

Other search engines such as Google Code Search, Krugle.com, and Koders.com have been targeted at program source code, but these index minimal metadata, and mostly function by restricting the scope of full-text search to source code files. Numerous IDEs such as Visual Studio and Eclipse support symbolic searches with full metadata support, but have limitations as well. Eclipse has a GUI-based interface to metadata specification which can be onerous to enter, and both IDEs limit searches to a single managed project at a time. The Source Insight IDE supports a larger search scope using heuristically-evaluated metadata, but does not support synonyms, homophones, concept keywords or lexical scoping in queries or results. Various research projects such as GENOA [4], SCRUPLE [7], TAWK [1], and a project by Clarke, Cox and Sim [3], have emphasized the back-end techniques of indexing code and left their front ends to technical pattern-matching languages. Strathcona [6] searches for code by example, alleviating the query language problem. Sourcerer [2] and Assieme [5] search for links within public code (and Assieme links in web-based descriptions) to enable programmers to learn how to use new APIs. We have made Codifier's query language straightforward, like a typical web search engine, and concentrate mainly on improving the usability of the UI for understanding and manipulating search results.

Codifier will be demoed at the workshop and comments and feedback will be gratefully appreciated.

[1] Atkinson, D. C. and Griswold, W. 2006. Effective pattern matching of source code using abstract syntax patterns. *Softw. Pract. Exper.* 36, 4 (Apr 2006), 413-447.

[2] Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P., and Lopes, C. 2006. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion To OOPSLA*. (Portland, Oregon, USA, Oct 22 - 26, 2006). ACM Press, 681-682.

[3] Clarke, C., Cox, A., and Sim, S. 1999. Searching program source code with a structured text retrieval system (poster abstract). In *Proceedings of SIGIR*. (Berkeley, California, United States, Aug 15 - 19, 1999). ACM Press, 307-308.

[4] Devanbu, P. T. 1992. GENOA: a customizable language- and front-end independent code analyzer. In *Proceedings of ICSE*. (Melbourne, Australia, May 11 - 15, 1992). ACM Press, 307-317.

[5] Hoffman, R., Fogarty, J., and Weld, D. Assieme: Finding and Leveraging Implicit References in a Web Search Interface for Programmers. To appear in *Proceedings of UIST*. (Newport, Rhode Island, Oct 7-10, 2007). ACM Press.

[6] Holmes, R., Walker, R. J., and Murphy, G. C. 2005. Strathcona example recommendation tool. In *Proceedings of ESEC*. (Lisbon, Portugal, Sept 05 - 09, 2005). ACM Press, 237-240.

[7] Paul, S. and Prakash, A. 1994. A Framework for Source Code Search Using Program Patterns. *IEEE Trans. Softw. Eng.* 20, 6 (Jun. 1994), 463-475.