BCC: Enhancing Code Completion for Better API Usability

David M. Pletcher[†] and Daqing Hou[‡]
Computer Science[†]/Electrical & Computer Engineering[‡], Clarkson University, Potsdam, NY 13699
{pletchdm, dhou}@clarkson.edu

Abstract

Nowadays, programmers spend much of their workday dealing with code libraries and frameworks that are bloated with APIs. One common way of interacting with APIs is through Code Completion inside the code editor. By default, Code Completion presents in a scrollable list, in alphabetical order, all accessible members available in the type of a receiver expression and its supertypes. This default behavior for Code Completion should (and can) be further improved because (1) not all public methods are APIs and presenting non-API public members to a programmer is misleading, (2) certain APIs are meant to be accessible only in some limited contexts but not others, and (3) the alphabetical order separates otherwise logically related APIs, making it hard to see their connection. BCC (Better Code Completion) addresses these problems by enhancing Code Completion so that programmers can control how specific API elements should be sorted, filtered, and grouped.

1. Introduction

The IDE (Integrated Development Environment) is an important contributor to programmer productivity. Drawing from features such as automated refactorings and code-assist operations, modern IDEs have certainly changed the way software is developed. IDEs are particularly critical in helping programmers deal with the proliferation of APIs in libraries and frameworks. For example, in a keynote speech [1], Charles Petzold estimated that "[in the] .NET Framework 2.0[,] [t]abulating only MSCORLIB.DLL and those assemblies that begin with word System, [there are] over 5,000 public classes that include over 45,000 public methods and 15,000 public properties, not counting those methods and properties that are inherited and not overridden." Programmers need tools and features that can help them manage these APIs.

One such feature that has become standard in IDEs is Code Completion. This functionality displays a list class members that can be accessed from or invoked upon a specified "receiver" object instance. An example in Java from the Eclipse IDE is shown in Figure 1a. In Figure 1a, a local variable of type java.awt.Graphics2D named g has had Code Completion invoked upon it by typing "g.". Typically, hitting "." after either an identifier, expression, or the "this" or "super" keywords invokes the code-completion process. During this process, Code Completion in Eclipse computes a list of "completion proposals", which are either method invocations or field accesses that could be used to complete the current Java expression. If the user continues to type after the initial ".", the list of proposals is automatically filtered such that only those whose names begin with the typed-in letters are displayed. As shown in Figure 1a, the completion proposals are listed one per line, in order of member name, showing the member name and argument names and types (if applicable), return type (or declared type for a field), and the first type on the path from the receiver type to java.lang.Object where the member was declared in the type hierarchy.

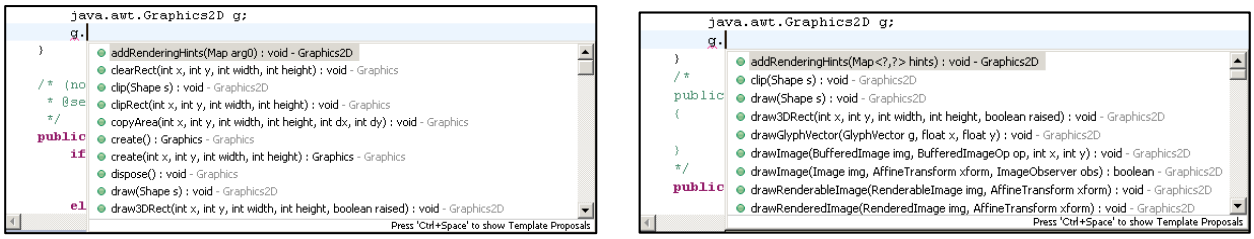
Code Completion regularly saves IDE users time when they work with difficult-to-remember or unfamiliar parts of APIs. However, Code Completion as currently implemented in IDEs like Eclipse can still be improved in several ways to deliver a better view of APIs to programmers. "Better Code Completion" (BCC) is a research prototype that demonstrates the feasibility of several such improvements. Specifically, BCC allows programmers to specify rules to control the presentation of APIs during Code Completion. [2] Specifically, BCC can be customized to filter out proposals that are not intended for programmers to see, to show a proposal only in certain specified contexts, and to provide a more intuitive ordering and grouping to the list of completion proposals. BCC is developed as an Eclipse plugin.

2. BCC: Better Code Completion

Unlike the default Code Completion in Eclipse, BCC allows programmers to customize the sorting, filtering, and grouping of code completion proposals.

1 http://www.charlespetzold.com/etc/DoesVisualStudioRotTheMind.html

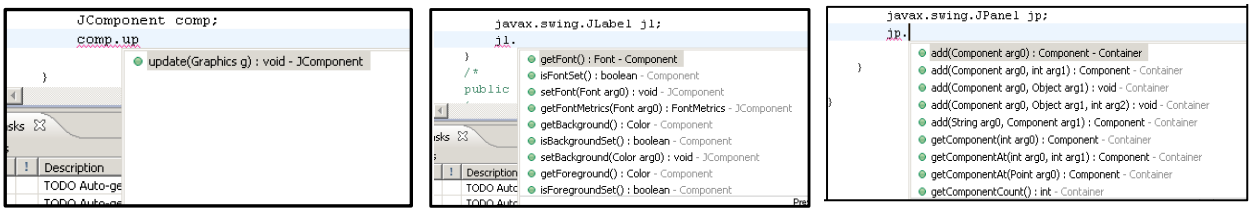
2 Currently, rules are specified with a simple grammar and stored in a text file. Due to space limit, the details will be described elsewhere.



(a) Code Completion in Eclipse 3.4 sorts all proposals available from supertypes alphabetically by name.

(b) BCC sorts proposals by position of declaration type in type hierarchy and then alphabetically by name within type.

Figure 1: BCC sorts proposals differently from the default Eclipse Code Completion.



(a) BCC filters out public method updateUI().

(b) BCC filters out public method Container.add() for JLabel.

(c) BCC keeps add() available for JPanel.

Figure 2: BCC private filters.

BCC sorts the list of completion proposals in the order from the declared type to the root in the type hierarchy (java.lang.Object), then alphabetically within each type. An example of this behavior is shown in Figure 1b, where methods from Graphics2D are shown at the top of the list, unlike the default Code Completion shown in Figure 1a, which mixes up methods from Graphics and Graphics2D.

BCC also allows for filtering out completion proposals based upon user-defined context-sensitive filters. Three types of filters can be applied.

- *Private Filter.* The class javax.swing.JComponent contains the public method updateUI(). Although public, this method is not intended to be directly accessed by client code. It is made public because the Swing artifact(s) invoking this method are located outside the package javax.swing. BCC can be configured to filter out such non-API public methods (Figure 2a).
- *Private Filter With Receiver Exceptions.* Single inheritance in Java has been shown to generally lead to cleaner design, but it can have negative consequences too. One example is that class javax.swing.JComponent inherits from java.awt.Container. The designers probably made this decision to avoid the code duplication that would result from having each "true container" class redefine the exact same Container methods. However, this means that non-container classes like JLabel also inherit from Container, even though JLabel is probably never intended to be a container. This filter makes methods inaccessible (Figure 2b) except upon certain subclasses (e.g., JPanel in Figure 2c).
- *Subclass Only.* The public paint() method in class java.awt.Component should generally be called by an instance's own repaint() method, or from within a subclass of Component that wishes to extend paint(). With BCC, paint() can be made accessible only from within subclasses of Component.

Finally, BCC allows programmers to specify which methods should belong to the same group and, thus, be displayed together in the Code Completion pane. For example, the add()/getComponent()/remove() methods in java.awt.Container can be grouped together logically as they control how child components are added or removed from a container. In addition, a group can span multiple classes in the same type hierarchy.

3. Status and Future Work

We are currently working on validating BCC by applying it to real-world libraries like AWT/Swing. Another future work is to test BCC with real users to better understand its potentials in helping programmers.

Because BCC implementation involves direct modification to the non-public parts of the JDT plugin in Eclipse, BCC cannot be simply downloaded and installed as a plugin. But BCC can be made available from us upon request.