# VoiceCode: an Innovative Speech Interface for Programming-by-Voice

**Alain Désilets**
National Research Council of Canada
Bldg M-50, Montreal Road
Ottawa, On, Canada
alain.desilets@nrc-cnrc.gc.ca

**David C. Fox**
Nuance
1 Wayside Road
Burlington, MA 01803
davidcfox@post.harvard.edu

**Stuart Norton**
School of Engineering
U. C. Santa Cruz
1156 High Street
Santa Cruz, CA 95060
stuart@soe.ucsc.edu

## Abstract
In this paper we describe VoiceCode, a system for programming-by-voice. With VoiceCode, programmers can dictate code in an easy to pronounce syntax, which the system translates to native syntax in the current programming language. We illustrate how this approach addresses most of the usability issues for programming-by-voice.

## Keywords
Programming-by-voice, speech interfaces.

## ACM Classification Keywords
H5.2. User Interfaces;


*CHI 2006*, April 22–27, 2006, Montréal, Québec, Canada.
ACM 1-59593-298-4/06/0004.

## Introduction
For the growing number of programmers suffering from Repetitive Strain Injury (RSI), programming-by-voice (i.e. speech input of programming language code) could be an attractive alternative to mouse and keyboard. Unfortunately, programming-by-voice with standard off-the-shelf Speech Recognition (SR) tools is not practical, mainly because programming languages were never meant to be spoken. For example, to dictate the simple C++ statement below:

```
if (currRecNum < maxOffSet)
{
    ^
}
```

One might have to say something like this:

" *if open-paren Charlie uniform Romeo Romeo cap Romeo echo Charlie cap-November uniform Mike less-than max begin-capitalize begin-no-space off set end-capitalize end-no-space close-paren new-line open-brace new-line new-line close-brace up-arrow tab-key"*

This simple example illustrates many of the usability issues of programming-by-voice described in [3]. These are: (i) code dictation (punctuation, indentation, and symbols), (ii) code navigation (local within screen and global project wide), (iii) error correction and (iv) mouse-free operation. Various tools have been

developed to address these issues ([1, 2, 3, 4, 5]) but to date, no single tool exists to address them all.

In this paper, we give an overview of VoiceCode [6] an OpenSource system for programming-by-voice that addresses all but one of those issues (for a more dynamic view of the system, see also the accompanying video by googling for *"VoiceCode CHI Demo Movie"*). Our basic approach is to allow the programmer to dictate code using a spoken syntax which is natural, concise as well as easy to utter and to learn. These natural utterances are then translated in real time to the more awkward native syntax of the programming language under use (ex: Python, C++). For example, to dictate the C++ statement from the **Introduction**, one would say something like this:

*"if current record number is less than max offset then"*

This is much shorter and less cognitively demanding to speak than the utterance shown in the **Introduction**. With VoiceCode, the programmer can also navigate and modify the code by uttering a continuous stream of natural spoken commands. In the above example, the word *"then"* moves the cursor to the beginning of the body of the if. Other forms of navigation are possible, which we will describe later in the paper.

VoiceCode interprets continuous utterances as a sequence of mostly independent **context-sensitive commands**. Commands may do a number of things such as: typing template code, moving the cursor, typing a symbol, deleting code, etc. The commands are context-sensitive in that the same spoken command may result in different actions depending on contextual information such as: code surrounding the cursor, language of the active source file, nature of the command that preceded it, etc.

In the rest of the paper, we show with examples how this approach addresses all but one of the usability issues listed earlier (namely, global navigation).

## Continuous dictation and navigation

With VoiceCode, programmers dictate continuously and can pause whenever it feels natural. In addition, they can intersperse continuous navigation commands within a dictation stream. For example the following continuous utterance (navigation command in bold):

*"clients array at index i **jump out** equals zero"*

Would result in this code:

```
clientsArray[i] = 0^;
```

Standard dictation systems do not allow such mixing of dictation and navigation commands because a phrase like *"jump out"* could mean either one of: "jump out of the brackets pair" or "type the literal string 'jump out'". Consequently most dictation interfaces require the user to disambiguate explicitly by speaking navigation commands as single isolated utterances. In the context of programming-by-voice where much navigation is needed, this is too cumbersome. Therefore in VoiceCode, we allow the programmer to mix commands and dictation utterances continuously.

## Effective use of code templates

VoiceCode addresses the issue of **punctuation** and **indentation** through the use of pre-punctuated and pre-indented code templates for common structures

(conditionals, for loops, class definitions, etc.). The programmer can then move around those templates using natural utterances like *"then"* and *"add arguments".* In the example from the **Introduction**, the word *"if"* was translated to the following template:

```
if (^)
{

}
```

and the word *"then"* moved the cursor to the blank line between the two braces. This combination of template and natural navigation commands allows the programmer to leave much unsaid (in this case, the braces, parentheses, all the new-lines, and the tab key). Note that the word *"if"* is highly ambiguous because it is often part of a symbol name (ex: `checkIfBusy`). We deal with this issue through context sensitivity. More precisely, the word *"if"* types template code only if it is uttered onto a blank line. Otherwise, it is considered to be part of a symbol.

### Natural dictation of symbols

VoiceCode addresses **symbol dictation** by allowing the programmer to utter them as a sequence of English words, without worrying about abbreviations and formatting. In the **Introduction** example, the phrase "*current record number*" was automatically translated to `currRecNum`. When translating an utterance, if VoiceCode encounters a phrase that does not correspond to a known command, it assumes it to be the spoken form of a symbol. It then looks in a dictionary of known symbols (compiled from standard libraries and source files in the current project) to see if any of them is a likely abbreviation for the spoken form that was uttered. If it finds one, it simply types it in, and otherwise, it formats the spoken form to create a new symbol. When creating a new symbol, it uses the surrounding code context to decide how to do it. For example, utterance *"current record number"* could be formatted as `CurrentRecordNumber` or `currentRecordNumber`, depending on whether a class or a method name was expected there. This is another example of context-sensitivity.

### Flexible wording

To facilitate learning of the spoken syntax, VoiceCode supports different ways of saying the same thing. This allows programmers to use the form that comes most naturally to them. For example, all the following utterances would result in the same code *"class bubble sorter inherits from sorting strategy"*, *"new class bubble sorter with superclass sorting strategy*", *"define class bubble sorter subclass of sorting strategy"*.

### Transferability across programming languages

In addition, the bulk of the spoken syntax can be used across different programming languages. This too facilitates learning since it allows programmers to learn the syntax once and use it across languages. For example, if the programmer says: *"define class bubble sorter"* into a Python file, VoiceCode will type code that defines a class in the Python syntax. If the same utterance is spoken into a C++ file, VoiceCode will type code that defines a class in the C++ syntax. This is another example of context-sensitivity.

### Local Navigation

VoiceCode addresses **local within-screen navigation**, by supporting a variety of strategies: (i) Template

Navigation, (ii) Navigation by Punctuation and (iii) Navigation by Pseudo Code. We already provided examples of **Template Navigation** (ex: *"then"*, *"add arguments"*). **Navigation by Punctuation** involves moving the cursor before or after a particular punctuation mark (ex: *"next comma"*, *"after paren"*, *"out of brace", "jump out"*). **Navigation by Pseudo Code** involves moving the cursor before or after a particular snippet of code. This is done by uttering a word like *"before"* or *"after"* followed by an utterance that could be used to dictate the desired snippet of code (ex: *"after clients array at index zero"*). Navigation commands can be repeated using subsequent commands, in the same or reverse direction (ex: *"again 2 times", "previous one"*). Such utterances are considered repetition commands, only if the previous command allows repetition (of which navigation commands are particular examples). If not, they are taken to be part of a symbol. This is another example of context sensitivity. Combined together, these 3 strategies provide much better support for navigating code than standard commands like *"move up/down/left/right"*. Moreover, they are effective enough to support **mouse-free operation**.

## Error correction

VoiceCode supports two types of **error correction**: (i) Not What I Said and (ii) Not What I meant. **Not What I Said** is the standard type of correction and addresses situations where VoiceCode did not correctly recognize the words that were spoken. **Not What I Meant** is non-standard and addresses situations where VoiceCode did recognize the spoken words correctly, but translated them to the wrong code. For example, if *"current record number"* was translated to symbol `current_record_nb` when the user actually meant symbol `currRecNum`. For both types of correction, VoiceCode is able to correct the content of the source files based on the user's correction, and also to adapt in order to avoid the same mistake in the future.

## Conclusion

We believe VoiceCode is one of the best (if not the best) and most complete tools currently available for programming-by-voice. It represents an innovative application of speech interfaces to a domain that is not well serviced by standard dictation systems. To achieve this, we had to use several non-standard techniques: mixing dictation and commands in a continuous speech stream, context-sensitive commands, Not What I Meant correction, and automatic generation of abbreviations. Some of those techniques may have applications to speech interfaces outside of the programming domain.

## References

[1] Arnold, S. et al. Programming by Voice, VocalPrograming. In *Proc. of the ACM Conference on Assistive Technologies*. Nov 2000.

[2] Begel, A. Programming By Voice: A Domain-specific Application of Speech Recognition*. AVIOS Speech Technology Symposium–SpeechTek West* (2005)

[3] Désilets, A. VoiceGrip: A Tool for Programming-by-Voice. *Int. J. of Speech Technology 4, 2* (2001), 103-116.

[4] Price D. et al. NaturalJava : A Natural Language interface for Programming in Java. In *Proc. of the Int. conf. on Intelligent User Interface*. January 2000.

[5] Snell, L. An Investigation Into Programming by Voice and Development of a toolkit for Writing Voce-Controlled Applications. *M Eng. Report. Imperial college of Science, Technology and Medicine, London*. June, 2000.

[6] The VoiceCode project. http://voicecode.iit.nrc.ca/.