Chapter 4
From Natural Language to Programming Language

Xiao Liu
Pennsylvania State University, USA

Dinghao Wu
Pennsylvania State University, USA

ABSTRACT
Programming remains a dark art for beginners or even professional programmers. Experience indicates that one of the first barriers for learning a new programming language is the rigid and unnatural syntax and semantics. After analysis of research on the language features used by non-programmers in describing problem solving, the authors propose a new program synthesis framework, dialog-based programming, which interprets natural language descriptions into computer programs without forcing the input formats. In this chapter, they describe three case studies that demonstrate the functionalities of this program synthesis framework and show how natural language alleviates challenges for novice programmers to conduct software development, scripting, and verification.

INTRODUCTION
Programming languages are formal languages with precise instructions for different software development purposes such as software implementation and verification. Due to its conciseness, the absence of redundancy causes less ambiguity in describing problems but on the other hand, reduces the expressiveness. Since the early days of automatic computing, researchers have considered the shortcomings

DOI: 10.4018/978-1-5225-5969-6.ch004
that programming requires to accommodate the precision with the adoption of formal symbolism (Myers, Pane, & Ko, 2004). They have been exploring techniques that could help untrained and lightly trained users to write programming code in a more natural way, and natural programming is then proposed (Biermann, 1983; Pollock, Vijay-Shanker, Hill, Sridhara, & Shepherd, 2013).

Natural language, on the contrary, is excessive but in its expressiveness lacks precision (Biermann, Ballard, & Sigmon, 1983). Describing problems in natural language gives a considerable freedom in clarifying requirements closer to practice, but specifications will contain ambiguities which are fatal to problem-solving. The errors result from two perspectives: structural errors and descriptive errors. Structural errors are caused by language designs. For instance, “then” is used for describing sequential events but is considered only as the “consequence” construct in those programming languages (Pane et al., 2001). Descriptive errors are those brought by participants in specific problem descriptions which contain errors and ambiguities as well.

To achieve a balance between programming languages that contain rigid symbolism and syntaxes and natural language that contains ambiguities. We discuss the question of what is natural to end-users by reviewing a few papers on the language features in non-programmers’ descriptions to problem-solving. On top of the central finding on these features, we proposed a general framework for understanding natural language descriptions and automatically synthesizing programs for different software engineering purposes. With the implications of the proposed general framework, we take a closer look at different scenarios and conduct case studies on synthesizing a few domain specific languages. At the last, we discuss the potential limitations of the current framework and propose future works, before drawing a few conclusions.

**BACKGROUND**

Natural Programming, according to the definition from Brad Myer is “working on making programming languages, APIs, and environments easier to learn, more effective, and less error-prone”. To achieve the goal, researchers have conducted studies on various methods to make the programming process more natural. But what is natural to end-users? A few terms, including closeness of mapping (Green & Petre, 1996) and cognitive dimension (Bonar & Soloway, 1985) were created to evaluate the learnability of a programming environment or its language syntax. The closer a programming method is to the problem world, the easier the solution can be composed.

Natural language programming is one of the significant directions being discussed that creates an easier way for people to compose the solutions for a programming task.