Chapter 14
Improving the Performance of Neuro-Fuzzy Function Point Backfiring Model with Additional Environmental Factors

Justin Wong
University of Western Ontario, Canada

Danny Ho
NFA Estimation Inc., Canada

Luiz Fernando Capretz
University of Western Ontario, Canada

ABSTRACT
Backfiring is a technique used for estimating the size of source code based on function points and programming. In this study, additional software environmental parameters such as Function Point count standard, development environment, problem domain, and size are applied to the Neuro-Fuzzy Function Point Backfiring (NFFPB) model. The neural network and fuzzy logic designs are introduced for both models. Both estimation models are compared against the same data source of software projects. It is found that the original NFFPB model outperforms the extended model. The results are investigated, and it is explained why the extended model performed worse.

DOI: 10.4018/978-1-4666-4785-5.ch014
INTRODUCTION

Software Estimation Background

Software estimation is important in software development. Project estimation is used to determine the necessary time and budget of projects. Many estimation techniques have been developed for software estimation such as: Constructive Cost Model (COCOMO), Putnam’s Software Lifecycle Management (SLIM), and Function Point (Stutzke, 2005). Source lines of code (SLOC) and function points are two popular metrics used today.

SLOC metric is used to measure the amount of source code in software. SLOC can also be used to determine the cost and effort needed to develop a software application. SLOC can be referred to as physical SLOC or logical SLOC. Physical SLOC is the number of lines in an application. Problems with physical SLOC are that formatting and style affect the size of an application. Logical SLOC, on the other hand, is unaffected by formatting and style because it measures the number of statements. The SLOC metric still has shortcomings. It is sensitive to programming language and technology (Galorath & Evans, 2006). Despite these problems, SLOC is still a popular metric used in the software industry today.

Function point is a unit of measurement for determining the functional size of an information system introduced by Albrecht in the 1970s (Albrecht & Gaffney, 1983). In the 1990s, the popularity of function point grew. It became a major tool in software sizing. Function points are used today for sizing software in both industry and academia. As the usage of function points grew, the International Function Point User Group (IF-PUG) was formed (International Function Point Users Group, 2007).

Function point analysis is a process of classifying major system components as ‘simple’, ‘average’, or ‘complex’. Unadjusted Function Points (UFP) is a measure obtained from identifying system components. The function components are Internal Logic Files (ILF), External Interface Files (EIF), External Inputs (EI), External Outputs (EO), and External Inquiries (EQ). The resulting function point count or Adjusted Function Points (AFP) is obtained by multiplying the UFP by the Value Adjustment Factor (VAF). There are 14 General System Characteristics (GSC) that defines VAF (International Function Point Users Group, 2005).

BACKFIRING TECHNIQUE

Backfiring is a technique used to size source code by converting function points to logical SLOC statements (Jones, 1995). Backfiring can be accomplished by multiplying the function point with the conversion ratios to obtain the SLOC. Similarly, backfiring can also be used for calculating function points by dividing the SLOC by the conversion ratios (Stutzke, 2005). Jones classified that the conversion ratios are defined based on the number of statements required to implement one function point based on the programming language (Jones, 1995). Based on these classifications, “high-level language” is defined as having less than 50 source lines-of-code per function point (SLOC/FP), while “low-level language” has over 100 SLOC/FP (Jones, 1995). Software Productivity Research (SPR) annually publishes the conversion ratios for many programming languages (Software Productivity Research Incorporated, 2006). Table 1 illustrates SPR’s programming language’s SLOC/FP and language levels.

The equation for converting function point to SLOC is defined in Equation 1. In Equation 1, FP is the function point input and conversion is the SLOC/FP. The SLOC equation can be rearranged to find the number of function points if the SLOC is known (Jones, 1995).