Chapter 87
Effort, Time, and Staffing in Continually Evolving Open-Source Projects

Liguo Yu
Indiana University – South Bend, USA

ABSTRACT

Scheduling and staffing are important management activities in software projects. In closed-source software development, the relationships among development effort, time, and staffing have been well established and validated: the development effort determines the development time and the best number of developers that should be allocated to the project. However, there has been no similar research reported in open-source projects. In this chapter, the authors study the development effort, development time, and staffing in an open-source project, the Linux kernel project. Specifically, they investigate the power law relations among development effort, development time, and the number of active developers in the Linux kernel project. The authors find the power law relations differ from one branch to another branch in the Linux kernel project, which suggests different kinds of management and development styles might exist in different branches of the Linux kernel project. The empirical knowledge of software development effort obtained in this study could help project management and cost control in both open-source communities and closed-source industries.

1. INTRODUCTION

In software projects, scheduling and staffing are important management activities. Over scheduling and over staffing can result in the addition of development cost. Under scheduling and under staffing could result in delays of product delivery. Therefore, in software projects, scheduling and staffing are determined based on the estimated development effort. Several formulas have been proposed and validated to show the ideal relationships among development effort, development time, and the number of developers that should be allocated to a project (Walston & Felix, 1977; Putnam, 1978; Boehm, 1981).

However, most of the published work in this area is performed on closed-source projects. Little work has been done to study the relationship among development effort, development time, and the number of active developers in open-

DOI: 10.4018/978-1-4666-7230-7.ch087

Copyright © 2015, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

source projects (Koch & Schneider, 2002; Koch 2008). Although open-source projects are loosely organized and managed, it is worth of studying their management styles to understand how open-source projects self-organize to form their own laws about development effort, development time, and staffing in order to help project management and cost control in both open-source communities and closed-source industries.

The remainder of this chapter is organized as follows. Section 2 reviews related work in software effort estimation. Section 3 presents the background knowledge of this study. Section 4 describes the data source and the data representation. Section 5 presents the analysis and the results. Conclusions appear in Section 6.

2. LITERATURE REVIEW

Software effort estimation is to predict the manpower required to develop or maintain a software product. Effort estimation is the basis for cost estimation, time scheduling, and staff allocation. Extensive research has been performed in this area (Albrecht & Gaffney, 1983; Jeffery & Low, 1990). Basically, there are two types of effort estimation method: expert judgment (Parkinson, 1957) and algorithmic models (Donelson, 1976). In algorithmic models, COCOMO II is considered the most successful approach (Boehm et al., 2000). Since the introduction of these models, a lot of following work has been performed in this area.

Some studies are reported to compare the performance of different effort estimation models. For example, Jorgensen (1995) compared different software maintenance effort prediction models developed using regression analysis, neural networks, and pattern recognition. He found the most accurate estimations were achieved through applying multiple regression and pattern recognition in the prediction models. Jeffery, Ruhe, and Wieczorek (2000) compared the development cost estimation differences of models using ordinary least-squares regression and analogy-based estimation. Menzies, Chen, Hihn, and Lum (2006) applied heuristic rejection rules to comparatively assess effort predictions generated from different models.

Some studies are performed to improve the accuracies of effort estimation models. For example, Chulani, Boehm, and Steece (1999) proposed using Bayesian approach to calibrate and improve cost estimation models, such as COCOMO II. Idri, Kjiri, and Abran (2000) suggested improving COCOMO model with fuzzy logic. Reddy and Raju (2009) recommended using Gaussian Membership Function to determine the cost drivers in order to improve the prediction accuracy. Huang, Ho, Ren and Capretz (2007) proposed improving the COCOMO effort estimation mode using neuro-fuzzy approach.

Building cross-project effort estimation models has been the major line of study in this area. Kitchenham and Mendes (2009) investigated the comparative effort prediction models. Caivano, Lanubile and Visaggio (2001) found that effort estimation models are process-dependent and accordingly cannot be reused for other processes. Maxwell, Wassenhove, and Dutta (1999) suggested that software companies should develop their own cost estimation models based on their own experience in order to generate accurate effort predictions. Menzies, Port, Chen, Hihn, and Sherry (2005) found that effort estimation models should be calibrated to local data using incremental holdout studies and predictions based on the within-company model were not significantly more accurate than those based on the cross-company model.

Software maintenance and evolution is one of the most important phases in software life cycle. Building maintenance effort estimation model is accordingly an important task for software engineering researchers. Sneed (2004) presented an effort model for software maintenance and evolution based on separations of fixed and variable cost. In a different study, Sneed (2005) described

20 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/chapter/effort-time-and-staffing-in-continually-evolving-open-source-projects/120996

Related Content

Repositories with Public Data about Software Development
Jesus M. Gonzalez-Barahona, Daniel Izquierdo-Cortazarand Megan Squire (2010). *International Journal of Open Source Software and Processes (pp. 1-13).*
www.irma-international.org/article/repositories-public-data-software-development/44968

Dynamic Key-Based and Context-Aware Internet of Things Authentication Approach
Mihir Mehtaand Kajal Patel (2022). *International Journal of Open Source Software and Processes (pp. 1-15).*
www.irma-international.org/article/dynamic-key-based-and-context-aware-internet-of-things-authentication-approach/310939

A Multi-Step Process Towards Integrating Free and Open Source Software in Engineering Education
K.G. Srinivasa, Ganesh Chandra Dekaand Krishnaraj P.M. (2021). *Research Anthology on Usage and Development of Open Source Software (pp. 389-397).*
www.irma-international.org/chapter/a-multi-step-process-towards-integrating-free-and-open-source-software-in-engineering-education/286584

Optimized Test Case Generation for Object Oriented Systems Using Weka Open Source Software
Rajvir Singh, Anita Singhrovaand Rajesh Bhatia (2021). *Research Anthology on Usage and Development of Open Source Software (pp. 596-618).*
www.irma-international.org/chapter/optimized-test-case-generation-for-object-oriented-systems-using-weka-open-source-software/286595

Long-Term Analysis of the Development of the Open ACS Community Framework
Michael Aram, Stefan Kochand Gustaf Neumann (2017). *Open Source Solutions for Knowledge Management and Technological Ecosystems (pp. 111-145).*
www.irma-international.org/chapter/long-term-analysis-of-the-development-of-the-open-acs-community-framework/168981