IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITB12843

This paper appears in the book, *Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2* edited by Mehdi Khosrow-Pour © 2006, Idea Group Inc.

A Model for Measuring Team Size as a Surrogate for Software Development Effort

James A. Rodger, Indiana University of Pennsylvania, MIS & Decision Sciences, Eberly College of Business & Information Technology, Indiana, PA 15705, jrodger@iup.edu

ABSTRACT

In this paper we identify a set of factors that may be used to forecast the software effort as measured in man hours. Using field data on over 1,000 field software projects from various industries, and the multiple regression model, we empirically tested the impact of development type, functional point complexity, and language type, on the software development effort. Drawing on the literature, we deduced that the same independent variables (development type, functional point complexity, and language type) could be used to predict a surrogate measure of software effort by examining team size as a dependent variable. Our results indicate that a relationship exists between the application and use of I-CASE tools, the number of functional complexity points, the number of fourth generation language types employed and the software effort. In a similar manner, a relationship was found between the application and use of I-CASE tools, the number of functional complexity points, and the number of fourth generation language types employed and the number of people assigned to a team.

INFORMATION ARCHITECTURE AND SOFTWARE EFFORT MODEL

Organizations strive to develop information architectures, or high-level maps of the information requirements of an organization, in order to map information needs, relate them to specific business functions and document their interrelationships (Brancheau and Wetherby, 1986; Allen and Boynton, 1991). The information architecture requires enabling software products to provide customer satisfaction and to facilitate the functions necessary to support the organizational environment (Mudie and Schafer, 1985; Jones, 2001). Boehm and Basili (2000) state that "if an organization wants a good information-technology-based system, there must be a synergistic relationship between the IT components that comprise the delivered system and the software engineering elements that guide its definition, development, component selection integration and validation. However, it has been reported that at least one in four software projects ends in failure (Keil et al, 2000). Keil and Robey (2001) support these findings and report that "despite advances in software engineering, project failure remains a critical challenge for the software development community. The information architecture is further used to guide software development (Brancheau, Schuster and March, 1989). Advances in Software Engineering offer significant potential for the development of information architectures of contemporary organizations (Avgerou, 1987). Information system architectures have been viewed as a means to reduce costs and improve productivity (Wardle, 1984; Bailey and Basili, 1981; Angelis and Stamelos, 2000). Cost containment and productivity improvement are major concerns faced by information systems departments (Mahmood, M.A. Pettingell, K.J. and Shaskevich, 1996). One of the approaches to contain costs and improve productivity is to develop better software effort estimation techniques.

SOFTWARE EFFORT

Currently, there are three popular methods of estimating the software effort. These three models are the linear regression model, cost models (COCOMO and COCOMOII) and vector prediction models (Hastings and Sajeev, (2001). All three of the proposed models use lines of code as one of the independent variables. Lines of code have been proposed as both an input and output variable by several authors, (Bailey and Basil, 1981; Boehm, 1981; Putman, 1978; 1979; Wolverton, 1974; Wrigley and Dexter, 1991). Cusamano and Kemerer (1990) argue that Japanese programmers perform well in software development and average 2000 lines of code per month, with one-tenth the error defects, versus 300 lines per month for US programmers.

There are many factors that affect programmer effort during software application development (Thadhani, 1984). Many researchers have developed alternate models for estimating software effort (Wrigley and Dexter, 1991; Benbasat and Vessey, 1980; Boehm, 1981; Kemmerer, 1987; Deephouse, Mukhopadyay, Goldenson and Kellner, 1996). June and Lee (2001) have proposed a quasi-optimal case-selective neural network model of software estimation. Blackburn, Scudder and Van Wassenhove (1996) developed a model to improve the speed and productivity of software development. Finne, Witting and Petkov, (1993) proposed a model for estimating the software development effort utilizing case-based reasoning.

Maximum Team Size

Assembling the right people for the software engineering project determines the team structure (Thomsett, 1994). Hammer and Champy (1994) have discussed "flattening the pyramid of the organizational structure", and the same idea can be applied to software project teams. Faraj and Sproull (2000) investigated the importance of expertise coordination in 69 software development teams. Expertise coordination involves "knowing where expertise is located, knowing where expertise is needed, and bringing needed expertise to bear." They found that "expertise coordination shows a strong relationship with team performance that remains significant over and above team input characteristics, presence of expertise, and administrative coordination. Ang and Slaughter (2001) suggest that here is a difference in work attitudes, behaviors and performance between outside contractors and permanent software development professionals. Howard (2001) points out that software engineers may have the right technical skills for a project but may not have the right personality characteristics.

While reusability is often correlated with alleviating the need to remodel and retest a large amount of software, it is equally important as a people issue (Unhelkar, 2003). People who have experience with reuse find it easier to create reusable designs and code because it leverages expertise (Lim, 1994). Further, rewards are often given at the completion of a major milestone in functional development, while in object-oriented development, reusability requires the need to think ahead beyond the immediate problem Unhelkar and Mamdapur, 1995). Peer recognition for large-scale reuse and recognition from the organization when the

project is delivered on time motivates members of the team to reuse designs and codes written by other members of the team (Meyer, 1995; Thomsett, 1990).

FUNCTIONAL POINT COMPLEXITY

The number of function points is based on an algorithm that computes a weighted sum of inputs, outputs, and interfaces to other programs (Yourdon, 1993). Function points have been proposed as input variables for the software estimation model (DeMarco, 1978, 1982; Albrecht, 1979; 1984; Albrectht and Gaffney, 1983; Desharnais, 1988; Jones, 1986; Rubin, 1983; 1985; Symons; 1988). Kemerer and Porter (1992) conducted an empirical study for improving the reliability of function point measurement.

INTEGRATED DEVELOPMENT TOOLS

CASE technology tends to make automating tools flexible and easy to use. Development tools have been used to improve analyst and programmer productivity, improve software quality and reduce maintenance, and increase management control over the software development process. Automated software development tools fall into three categories, programming support tools, design technique tools and project management tools (Gregory and Wojtkowski, 1990). There is qualitative data available that supports the development tool type as having an impact on the software effort and productivity (Cusamano and Kemerer, 1990). Kim (1983) emphasizes that Toshiba's Software Factory produces 2870 instructions per programmer month in 1981 due to the use of integrated tool sets. Other researchers support these claims (Zelkowitz et al. 1984; U.S. Commerce, 1984; Johnson, 1985; Gamota and Frieman, 1988).

LANGUAGE TYPE

Programming languages are the primary tools for creating software. The basic challenge for business software builders is to build reliable software as quickly as possible. Fourth generation languages automate much of the work normally associated with developing software applications (Mimno, 1985). Several researchers have demonstrated a relationship between function points and lines of code in familiar languages (Yourdon, 1993; Jones 1991). It is generally accepted that the lines-of-code metric creates an obvious bias against high-level languages. Johnson (2000) states that a controversy exists about Object-Oriented Systems Development (OOSD) and that while experts claim that the advantages of OOSD make it superior to conventional systems development, others point out the disadvantages of OOSD and question whether it will ever be the dominant approach to software development. Sircar et al. (2001) demonstrated a gap between OO and the structured approach at the analysis and design stages of the SDLC. They suggest that training should focus on modeling issues rather than syntax and that this will help developers to bridge the gap between OO and structured approaches.

TECHNIQUE

The approaches and tools used to develop the software component of information system architectures are often key factors in their ultimate success (Paddock, 1986). These approaches can be divided into several categories. The traditional approach is based on the System Development Life Cycle (SDLC). These approaches can be further subdivided into Descriptive and Normative categories (Colter, 1984; Berrisford and Wetherbe, 1979). Howard (2002) points out that RAD was promised as a giant leap in software developer productivity and as a technique for cutting through traditional software project delays. However, few organizations have reported significant benefits attributed to RAD techniques. Highsmith and Cockburn (2001) advocate the application of agile software development approaches such as Extreme Programming, Crystal methods, Lean Development, Scrum, Adaptive software Development, that more closely reflects today's business and technology environment.

HYPOTHESES

H01: There is a relationship between the number of functional point complexities and the software effort.

Ho2: There is a relationship between the number of fourth generation languages employed and the software effort.

Ho3: There is a relationship between the number of integrated development tool types (I-CASE tools) employed and the software effort.

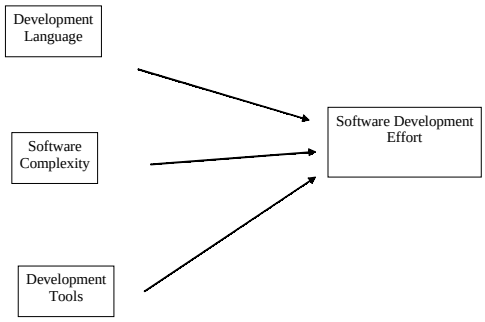
H04: There is a relationship between the number of functional point complexities and the software team size.

Ho5: There is a relationship between the number of fourth generation languages employed and the software team size.

Ho6: There is a relationship between the number of integrated development tool types (I-CASE tools) employed and the software team size.

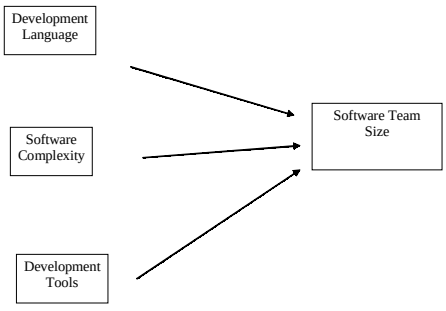
SOFTWARE EFFORT MODEL

Figure 1. Software Effort Model



SOFTWARE TEAM SIZE

Figure 2. Software Team Size Model



RESULT ANALYSIS OF SOFTWARE EFFORT MODEL

The applicability of the Software Effort Model can be determined by an analysis of the field study of 434 software development projects. Multiple regression analysis reveals the effects of integrated development tool type, functional point complexity, and language type on the software effort. (Table 1).

The results indicate that overall project evaluations are consistent with the Software Effort Model. The F value was 153.75 and the model was significant at the p=0.001 level of significance. The R-square for the model was .517. This indicates that model-independent variables explain 526% of the variance in the dependent variable.

Table 1. Software Effort One-Way ANOVA Table for Regression Analysis

DEGREES OF FREEDOM	SOURCE	SUM OF SQ.	MEAN SQ.	F VALUE	P>F
3	Regression	2.37E+10	7891343846	153.75	.000*
431	Residual	2.21E+10	51324693.70		
434	Total	4.58E+10			

** significant at p=.001 **Dependent Variable: Summary Work Effort*

Table 2. Software Team Size One-Way ANOVA Table for Regression Analysis

DEGREES OF FREEDOM	SOURCE	SUM OF SQ.	MEAN SQ.	F VALUE	P>F
3	Regression	11564.20	3854.73	3.94	.009*
213	Residual	2085564.99	979.18		
625	Total	220129.18			

** significant at p=.05 **Dependent Variable: Maximum Team Size*

SOFTWARE TEAM SIZE MODEL

The applicability of the Software Team Size Model can be determined by an analysis of the field study of 216 software development projects. Multiple regression analysis reveals the effects of integrated development tool type, functional point complexity, and language type on the software team size. (Table 2).

The results indicate that overall project evaluations are consistent with the Software Team Size Model. The F value was 3.94 and the model was significant at the p=0.05 level of significance. The R-square for the model was 0.053. This indicates that model-independent variables explain 5.3% of the variance in the dependent variable.

Table 1 data reveals the suitability of the Software Effort Model, Table 2 reveals the suitability of the Software Team Size Model. Based on the data shown in both tables and according to field study projects; integrated development tool type, functional point complexity, and language type, are the major factors that affect both summary work effort and maximum team size.