Inf Technol Manage (2007) 8:253–262
DOI 10.1007/s10799-006-0005-3

An empirical study of the impact of team size on software development effort

Parag C. Pendharkar · James A. Rodger

Published online: 2 February 2007

Abstract In this paper, we investigate the impact of team size on the software development effort. Using field data of over 200 software projects from various industries, we empirically test the impact of team size and other variables—such as software size in function points, ICASE tool and programming language type—on software development effort. Our results indicate that software size in function points significantly impacts the software development effort. The two-way interactions between function points and use of ICASE tool, and function points and language type are significant as well. Additionally, the interactions between team size and programming language type, and team size and use of ICASE tool were all significant.

Keywords Software engineering · Cost estimation · CASE tools · Programming teams

1 Introduction

Software cost estimation is one of the challenging problems faced by IS managers [7, 28, 31]. A common approach to estimate software cost is to estimate software development effort in man-hours, and then multiply the software development effort by the average hourly rate [11, 12]. Predicting software development effort, however, is a daunting task as it requires understanding of various factors involved in software development process. A few researchers have proposed some causal models for predicting software development effort, and a popular software estimation model is the Wrigley and Dexter's [31] software effort estimation model. Wrigley and Dexter's [31] model consists of three independent variables that can be used to predict software development effort. These three independent variables are: system requirements size, personnel experience, and method and tools.

Team size as a variable has been used in several studies that have used non-parametric data mining techniques for predicting software development effort [9, 17, 20, 29], but most theoretical models for predicting software development effort do not consider team size as a predictor of software development effort [30, 31]. Given that the team size variable appears to impact software development effort in real-world empirical software engineering datasets, there is a need to extend existing theoretical software effort prediction models to include the team size variable.

In the current research, we use Wrigley and Dexter's [31] theoretical model for predicting software development effort and extended it to incorporate team size as a software development effort predictor. Then, using publicly available multi-organizational and multi-project data, we empirically evaluate the direct and interaction effects of the independent variables. Specifically, we empirically validate the impact of software size in function points, type of programming language

P. C. Pendharkar (✉)
Information Systems, School of Business Administration, Capital College, Pennsylvania State University, 777 W. Harrisburg Pike, Middletown, PA 17057, USA
e-mail: paragp@psu.edu

J. A. Rodger
MIS and Decision Sciences, Eberly College of Business & Information Technology, Indiana University of Pennsylvania, Indiana, PA 15705, USA
e-mail: jrodger@iup.edu

used, use of ICASE tools, and team size on software development effort. We study the interaction effects of software size (function points) with the type of language used; software size and ICASE tool; software size and team size; team size and type of programming language; and team size and ICASE tool. For the rest of our paper, we use the term software effort to refer to software development effort.

The rest of the article is organized as follows. First, using the relevant models from software engineering literature, we develop a set of hypotheses and identify the factors that may impact software effort. Second, we describe our data and empirical results. In the end, we provide a summary, limitations and implications of our research.

2 Relevant models and hypotheses

There are several software effort estimation models available in the software engineering literature. Among the popular software effort estimation models are Constructive Cost (COCOMO/COCOMO II) models, Software Lifecycle Management (SLIM) model, and the Select estimator [12, 13]. Most of these software effort estimating models are criticized for incorporating subjective estimates, and assuming normal data distributions and specific type of business application and programming language [12]. These subjective estimates, assumptions of parametric data distributions, and data related to one programming language and one business application limits the generalizability of software effort estimating models.

A few researchers have focused on developing models to understand the primary antecedents of software effort. Wrigley and Dexter [31] proposed a general model that causally predicts the software effort throughout the systems development life cycle. Wrigley and Dexter's [31] model consists of three independent variables: system requirements size, personnel experience, and method and tools. Wrigley and Dexter [31] argue that the independent variables in their model capture the concepts of problem space, labor and capital, respectively. Noting the direct effects of independent variables on the software effort, Wrigley and Dexter [31] write:

> ''The model suggests that these variables are independent; however, there may well be interaction effects between requirement types, personnel skills, and tools used. For example, more experienced personnel likely would be assigned to the more difficult projects. Because of the possibility of such interaction effects, the model presents only the main effects.''

As mentioned before, several software effort estimation studies have considered team size as an independent variable for prediction of software effort. Among the researchers who have considered team size as a predictor of software effort are Blackburn et al. [9], Finnie et al. [17], and Smith et al. [29]. Most of these studies, using team size as an input, have used non-parametric machine learning models for forecasting software effort. A causal relationship between team size and software effort has not been well established yet. The team size variable represents the labor factor in Wrigley and Dexter's [31] model and we believe that Wrigley and Dexter's [31] software effort model can be extended to include team size as a predictor of software effort.

While we include the team size variable in Wrigley and Dexter's [31] model, we do not include personnel experience in our model. The exclusion of the personnel experience variable is due to lack of information on this variable in our data set. For the independent variables considered in our research, we study both the direct and the interaction effects.

2.1 System requirements size

The appropriate measure of system requirements size/software size has been debated in the literature. Among the popular approaches to measure software size are source lines of code (SLOC) and function points (FPs) methods [1]. There is no agreement among the researchers as to which approach provides better measurement of software size. FP approach is favored by several empirical studies [26], but is criticized as being labor intensive and incapable of lending itself to automation [26]. The SLOC approach provides simple easy to use and easy to understand metric, but it is criticized for not addressing the issue of language difficulty [26]. Some researchers recommend using both approaches: SLOC for measuring software size, and FP for measuring software complexity [4]. Subramanian and Zarnich [30] argue that for an integrated CASE (ICASE) environment, FP is a good estimator of software effort.

There are two major types of FPs: the international function point users group (IFPUG) function points; and the Mark II function points. The IFPUG FPs measure the functionality of an application and are independent of the technology being used [29]. The computation of the number of function points is based

on an algorithm that computes a weighted sum of inputs, outputs, and interfaces to other programs. FPs measure the following five software characteristics:

1. The number of external input types
2. The number of external output types
3. The number of master/logical file types
4. The number of inquiries
5. The number of interface file types

The final FP count is computed as a weighted sum of the aforementioned five software factors where the weights reflect the value of each factor to the customer [29].

The Mark II Function Points (MK II FP) approach was developed to overcome some of the difficulties found in the IFPUG function point approach [20]. The IFPUG function point approach was developed for transaction processing systems. The Mark II Function Point model views the system as a set of logical transactions. Each logical transaction consists of inputs, processes and outputs. For every transaction, the FPs are calculated. The FP in MK II FP is the measure of the information of the processing size factor. The total FPs of the system are the sum of FPs of its transactions. The advantage of MK II FP is that the measure of software size is based on logical transactions of the system.

Most studies note a high degree of positive correlation between FP and software effort for both non-CASE and ICASE projects written in 3GL, 4GL, and object-oriented programming languages [2, 14, 30]. We validate this relationship by proposing the following hypothesis:

Hypothesis 1: Software size in function points is positively associated with the software development effort.

2.2 Methods and tools

Programming methods and tools are known to have an impact on software effort [29, 31]. Programming methods consist of programming language and development methodology [29–31]. In this research, we consider programming language as a variable that might have an impact on software effort.

Programming, project management and design tools—hereafter called development tools—do have an impact on software effort. 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 [30]. Automated software development tools fall into three categories: programming support tools, design technique tools and project management tools [19]. One qualitative study suggests that the development tool type may have an impact on the software effort [15]. In our research, we consider ICASE tool as a variable that impacts software effort. We now develop hypotheses related to the impact of programming language and ICASE tools on software effort.

2.2.1 Programming languages

Programming languages are the primary methods for creating software. The basic challenge for business software builders is to build reliable software as quickly as possible. Fourth generation languages (4GLs) automate much of the work normally associated with developing software applications [27]. However, Blackburn et al. [10] reported that language type does not have an impact on software effort because some of the programming languages, such as C++, might be more complex than some of the other third generation languages (3GLs). This leads to the following hypothesis:

Hypothesis 2: The type of a programming language (4GL vs. 3 GL) will not have any impact on software development effort.

4GLs and recent object-oriented programming languages are not only complex, but these languages also require programmers to learn several new functionalities that might lead to increased software effort. For example, Microsoft Foundation Classes (MFC) in Visual C++ and JAVA Swing classes in Java programming provide several reusable classes that might be used to design graphical user interfaces efficiently. A programmer is required to understand prewritten classes and frameworks to reuse code. 3GL languages don't provide such extensive capabilities and relative easy to learn. The reused code increases the software size in function points, but FPs don't use a metric for code reuse. Thus, it is likely that the interaction of FPs and language type might have an impact on software effort. Given the paucity of the studies in this area, we propose the following hypothesis:

Hypothesis 3: The interaction between FP and 4GL programming language type is positively associated with the software development effort.

2.2.2 ICASE tools

Integrated CASE (ICASE) tools are designed to provide support for all phases of the systems development life cycle [30]. The capabilities of ICASE tools include the following:

1. Graphical capabilities for modeling user requirements, and error and consistency checking
2. Prototyping and system simulation capabilities
3. Code generating capabilities
4. Code testing, code validation, and code reuse capabilities
5. Reengineering, reverse engineering, data dictionary and database interface capabilities
6. Management information acquisition, storing, managing and reporting capabilities

Subramanian and Zarnich [30] argue that ICASE tools by themselves do not always lower software effort. Most of the ICASE tools are complex to learn and sometimes increase software effort. The use of ICASE tools require extensive training to see significant software effort reduction. Blackburn et al. [10], speculating on the impact of CASE tools, mentioned the following.

> ''...that increasing project complexity and size are obscuring the advantages that CASE tools bring...''

Blackburn et al. [10] believed that the interaction of ICASE tools and project size might have an impact on software effort. For example, ICASE tools, due to their coordination capabilities, allow programmers to coordinate activities for large size projects leading to reduced software effort [10]. Thus, we have following two hypotheses:

Hypothesis 4: The use of ICASE tools will not have an impact on the software development effort.[1]

Hypothesis 5: The interaction between function points and use of ICASE tools is negatively associated with the software development effort.[2]

2.3 Team size

Team size, as a factor impacting software effort and productivity, has been used in several studies [6, 17, 21, 29]. While team size seems to play a role, its impact on software effort is not clearly established. Microsoft used a strategy of employing small teams of star developers and found that the strategy, when confronted with the market realities of marketing, developing, and maintaining large mass-market applications, does not work well [23]. Among the problems of using small teams at Microsoft were [16]:

1. The overlapping responsibilities of software developers resulted in ''reinvention of the wheel'' and biases in software testing.
2. Program manager's close connection with the market and developers made it difficult for him/her to make trade-off decisions. For example, some program managers were more concerned with the views of developers than average users.
3. Program managers did not give detailed specifications to small teams.
4. Small teams did not manage error backlog effectively.

Large team size increase software effort due to inefficiencies created by the problems of coordination and communication between the members of the team. However, larger team size represents better distribution of skills and software projects might benefit from larger team size [5]. The better distribution of skills of large teams typically improves software quality, but increases the overall software effort [16]. In an open source project GNOME, Koch and Schneider [24] observed that increasing the number of programmers working on a project leads to higher software effort. We write the impact of team size on software effort in the following hypothesis:

Hypothesis 6: Team size is positively associated with the software development effort.

Fried [18] mentions that the interaction between team size and type of programming language; and team size and use of CASE tools, might have an impact on software effort. For example, Fried [18] argues that the use of 4GLs has reduced software effort by vastly accelerating the turnaround of program testing and eliminating the complex communication requirements between programmers. By the same token, Fried [18] argues that CASE tools (that allow coordination) can reduce software effort by significantly reducing the number of interactions between team members. Fried [18] did not provide any empirical evidence for his claims. We test his claims by proposing the following hypotheses to test the interactions effects between team size and ICASE tools, and team size and programming languages.

Hypothesis 7: The interaction of team size and use of ICASE tools will have no effect on software development effort.

[1] Our data did not contain the team member skills. The team with better skills in using ICASE tools is likely to see a reduction in software development effort. We assume that our data contains team skills that are uniformly distributed and effort reduction by experienced programmers will be cancelled by the higher effort of inexperienced programmers leading to no overall change in the software development effort.

[2] Since complex large size projects typically have more experienced programmers and clear specifications, we assume that software size and use of ICASE tools will reduce effort.

Hypothesis 8: The interaction between team size and 4GL is negatively associated with the software development effort.

Fried [18] mentions the positive impact of team size on software quality. According to him, larger team sizes might lead to well-documented and quality systems. Biffl and Gutjahr [8], noting the interaction between software size and team size, note that project managers have to manage team size so that some members of the team can inspect the software for errors, and other members can do their usual work in software development. Thus, for larger projects, it is more likely that the projects team members will be divided into two different roles—development and inspection. Dual role of project personnel will lead to decreased software effort due to clear specifications and non-overlapping responsibilities [16]. This leads to the following hypothesis:

Hypothesis 9: The interaction between team size and software size (FP) is negatively associated with the software development effort.

3 Data and experiments

We obtained the data on 1,238 software projects from International Software Benchmarking Standards Group (ISBSG). The ISBSG (release 7) data are used by several companies for benchmarking software projects and are available in the public domain. The ISBSG procedures encourage software development teams to submit their project data to the repository in return for a free report that graphically benchmarks their projects against similarly profiled projects in the ISBSG repository [3]. The software project data are typically submitted by the software project manager, who completes a series of special ISBSG data validation forms to report the confidence he/she has in the information he/she provides. ISBSG has developed a special mutually exclusive data quality rating that reflects the quality of data related to any given project. Each project is assigned a data quality rating of A, B, or C to denote the following:

- A = The project data satisfies all the criteria for seemingly sound data.
- B = The project data appears fundamentally sound, but some data attributes might not be fundamentally sound.
- C = The project data has some fundamental shortcomings.

Companies participating in ISBSG benchmarking acquired project data in several different ways. FP data on the projects were acquired mostly by an automated process (about 40%) or obtained from the development tools (about 21%). The software effort data were mostly recorded (about 59%). In certain cases, the software effort data were derived from the actual project cost (about 13%). In many cases, the data acquisition procedures were missing or unknown.

The software projects in ISBSG release 7 data came from 20 different countries. The top three known contributing countries were the United States, Australia and Canada. Over 97% of the projects were completed between the years 1989 and 2001. Most of the projects (about 50%) were completed between the years 1999 and 2001. Approximately 57% of the projects had a data quality rating of A, 33% had a data quality rating of B and about 10% had a data quality rating of C. ISBSG endorses data quality rating of A & B as good data with high reliability. Figure 1 illustrates the industry type distribution for the 1,238 projects. Table 1 details the descriptive statistics of a few variables for the original 1,238 projects data.

The ISBSG data set included data on software size in FP, integrated CASE tools, programming languages, software effort and team size. Of the total 1,238 software projects, only 217 projects had complete data on all five independent and dependent variables. We used all 217 projects in our analysis. Table 2 illustrates the descriptive statistics of some of the variables used in

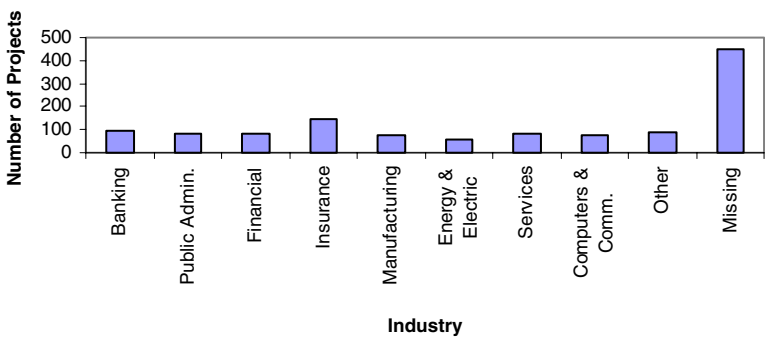

Fig. 1 Industry type distribution in the ISBSG release 7 project data

Table 1 Descriptive statistics of variables in ISBSG release 7 project data

Variable	Mean	Std. Dev.	Minimum	Maximum
Function points	642.59 counts	1,264.60	8	19,050
Software effort	8,414.80 man-hours	32,698.70	10	645,694
Team size	8.16 persons	20.16	1	468

Table 2 Descriptive statistics of variables

Variable	Mean	Std. Dev.	Minimum	Maximum
Function points	533.21 counts	615.17	11	4,932
Software effort	5,086.60 man-hours	11,386.30	17	138,883
Team size	6.97 persons	31.92	1	468

our analysis. The common methods for measuring function point are International Function Point Users Group (IFPUG) FPA and Mark II FPA. While most of the projects (57%) used IFPUG or Mark II FPA to record software size, about 32% of the projects did not mention FPA recording approach, and the remaining 11% used other methods. Comparing the mean, standard deviation, minimum and maximum values for FP in Tables 1 and 2 indicates that the 217 projects used in our analysis did not contain very large and very small size projects from the original 1,238 projects. Further, our sample had a lower variance for software size in FP than the original 1,238 projects.

We draw similar conclusions for the software effort. The team size variable in the data set is the maximum (peak) team size. A comparison of the team size descriptive statistics from Table 2 with the team size statistics from Table 1 reveals that team size in the sample of 217 projects has higher variance.

A majority of the projects used in our analysis, about 52%, used a fourth generation programming language. ICASE tools were used in only about 9.2% of the projects. All other projects used upper CASE tools, no CASE tools or lower CASE tools. The ICASE tool distribution in the original data set is shown in Fig. 2. Only about 4% of the projects in the original ISBSG data set used ICASE tools.

Figure 3 illustrates project distribution by industry type. The data quality ratings for the 217 projects were about 52% A grade, about 30% B grade and about 18% C grade. Majority of projects, about 19%, were from banking industry. When comparing the industry distribution and data quality of the 217 projects with the original set of 1,238 projects, we see that the data quality distribution of the 217 projects is very similar to

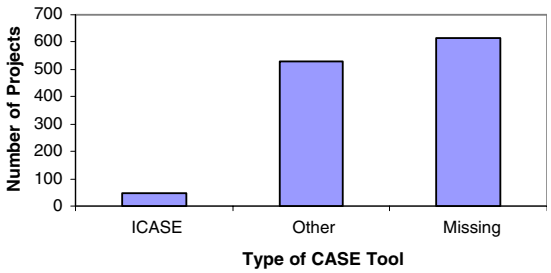

Fig. 2 Distribution of CASE tools in the ISBSG release 7 project data

the data quality distribution of the original 1,238 projects. On the other hand, the industry type distribution for the 217 is different from that of the 1,238 projects. We used the principle of relative entropy[3] for formally comparing the data quality and industry type distributions for the 217 projects with respect to the original data set of 1,238 projects [25]. According to the principle of relative entropy [25], given two discrete probability mass functions, say $\boldsymbol{p} = (p_1, p_2, \ldots, p_n)$ and $\boldsymbol{q} = (q_1, q_2, \ldots, q_n)$, the relative entropy of $\boldsymbol{q}$ with respect to $\boldsymbol{p}$ is defined by

$$\sum_{i=1}^{n} p_k \ln \frac{p_k}{q_k}.$$

When $\boldsymbol{p}$ and $\boldsymbol{q}$ are exactly same, the value of relative entropy is zero. Values close to zero are indicative of similar distribution [25]. If $\boldsymbol{p}$, for $n = 3$, denotes the data quality distribution for 217 projects used in data analysis, and if $\boldsymbol{q}$ denotes the data quality distribution for the original 1,238 projects, then $\boldsymbol{p}$ = (0.51, 0.31, 0.18) and $\boldsymbol{q}$ = (0.57, 0.33, 0.10). The elements of $\boldsymbol{p}$ and $\boldsymbol{q}$ are percentages of projects that belonged to data quality ratings A, B and C, respectively. Using the relative entropy formula, the relative entropies of the 217 projects with respect to the 1,238 projects for data quality and industry distribution were 0.03 and 0.41, respectively. The relative entropy numbers indicate that the data quality distribution between the 217 projects was more similar to that of the 1,238 projects than the industry distribution between the two sets of projects.

Several researchers have used previous versions of the ISBSG data set. For example, Angelis et al. [3] used the ISBSG release 6 data for generating a multi-organizational software cost estimation model. Jeffery et al. [22] used ISBSG release 5 multi-organizational

[3] We use entropy because these variables take discrete values (A, B, and C) and continuous value tests such as *t*-test are not suitable for testing difference in means for discrete variables.

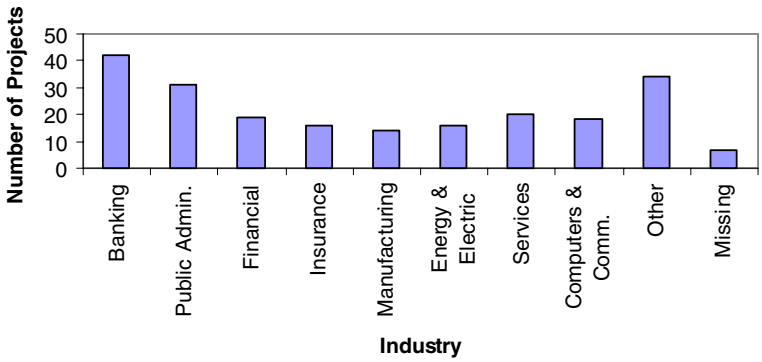

Fig. 3 Distribution of projects by industry type

data and company-specific data to compare two software development cost modeling techniques. Their results indicated that the use of ISBSG multi-organizational data provides significantly more accurate results than analogy-based estimates.

A few variables that are used in our study overlap the variables used in previous studies. These variables are function points, software effort and team size. The cost models used by Jeffery et al. [22] and Angelis et al. [3] used function points and team size as independent variables and software effort as a dependent variable, among others. Our study enhances the Jeffery et al. [22] and Angelis et al. [3] studies by including more variables (development tools and development language) and several interaction effects. The data set used in our study—release 7—has several additional projects.

We use Ordinary Least Squares (OLS) analysis to test all the hypotheses. Since our data have some variance in data quality, which reflects errors in measurement of variables, we consider data quality (DataQ) as an interaction factor in our OLS. ISBSG considers data quality labels A & B data as high reliability data. However, data quality label C data is generally considered low quality data. For our OLS, we consider data quality variable taking binary values of high quality (A&B labels) and low quality (label C) data. Table 3 illustrates the results of the overall model fit. The results indicate that the overall model fit was satisfactory. The *F*-value is 52.49, and the model fit was significant. The *R*-square for the model was 0.784. This indicates that model-independent variables explain about 78.4% of the variance in the dependent variable.

Table 3 The overall model fit statistics

Source	DF	Sum of squares	Mean square	*F*-value
Model	14	21,966,311,312	1,569,022,236.5	52.49**
Error	202	6,037,683,848	29,889,524.0	
Corrected total	216	28,003,995,160		

** 1% significance; *R*-square = 0.784

Table 4 illustrates the coefficients and significances of factors, covariates and interactions effects. The results provide support for hypothesis one, two and three. Further, the results indicate a very weak support for hypothesis four. Hypothesis four was supported at 1% and 5% level of statistical significance. However, at a 10% level of statistical significance hypothesis four was rejected. At this level of statistical significance, the negative coefficient in the results indicates that the use of ICASE tool will actually lower the software effort.

Hypothesis five was supported by the results indicating that the interaction between FP and ICASE tools lowers software effort. No support was found for hypothesis six indicating that increasing team size does not lead to higher software effort. The results did not provide any support for hypothesis seven indicating that the interaction between team size and the use of ICASE tools will increase the software effort. The results supported hypothesis eight and did not support hypothesis nine, which indicated that interaction between team size and 4GL reduces software effort and interaction between team size and FP does not lower software effort.

Table 4 Model parameter estimates and their significance

Parameter	Coefficient	Std. error	| *t* |-value	Significance
Team	–611.27	371.82	1.644	0.102
ICASE	–7680.82	4,115.36	1.866	0.063***
Language	427.61	2,147.42	0.199	0.842
FP	12.656	1.857	6.814	0.000**
Team*ICASE	730.45	284.45	2.57	0.011*
Team*Language	–600.48	187.85	3.197	0.002**
Team*FP	–0.034	0.112	0.300	0.764
Language*FP	10.196	1.635	6.235	0.000**
ICASE*FP	–7.307	1.72	4.249	0.000**
DataQ*ICASE	366.89	1,981.41	0.185	0.853
DataQ*Language	678.46	2,026.23	0.335	0.738
DataQ*FP	–3.837	1.802	2.13	0.034*
DataQ*Team	605.89	167.31	3.621	0.000**
Intercept	6,525.93	4,510.63	1.447	0.150

* 5% significance; ** 1% significance; *** 10% significance

We tested whether the interaction of data quality rating and different independent variables had an impact on the software effort. The interactions of data quality and ICASE tool, and data quality and language type were not significant. Since ICASE tools and language type are categorical variables, it is less likely to have measurement errors for these variables. The interaction effects of data quality and FP, and data quality and team size were significant.

Since team size measurement errors were more likely in our dataset, the results of hypotheses related to team size should be considered preliminary and care should be exercised in using the results of team size hypotheses.

4 Summary and limitations

We have investigated the factors impacting the software effort. Using the existing literature, we identified several variables that might impact the software effort. Further, using a data set of over 200 projects, we empirically tested the impact of several factors on the software effort. The results of our experiments indicate that higher software size in FPs lead to higher software effort. We did not observe any significant relationship between ICASE tools and software effort. Further, we observed that the interaction of ICASE tools and function points decreases the software effort.

ICASE tools have been known to have significant impact on software effort [30]. In our case, weak significant relationship was observed for the impact of ICASE tools on reduction of software effort. Our data on ICASE tool was biased as over 90% of our data set did not contain ICASE tools, and the limited number of ICASE tools might have jeopardized the statistical significance.

The type of programming language did not have an impact on software effort. The descriptive statistics of the data indicate that about 51% of the projects were developed in 4GL languages, and 49% of the projects were developed in 3GL programming languages. Thus, we believe that our data was not very biased for any particular generation of programming languages. The insignificance of programming language on software effort could be due to several reasons. The first reason may be that the programmers' experience in programming language might play a role in determining software effort as a few languages are more difficult to learn than others. Second, the complexity of a language, such as code and design reuse, may compensate for any other advantages that it might offer and negate any significant impact on software effort.

We found that the interaction between 4GL and software size in FP leads to increase in software effort. We observed very interesting results in regards to team size. We noticed that an increase in team size does not increase the software effort. When team size increases, it is likely that both the team expertise and the communication requirements increase and no significant software effort improvements are seen. The interaction between team size and language type decreases the software effort. The 4GL languages and larger team sizes have a lower software effort than 3GL languages and smaller team sizes. The interaction between team size and ICASE tools was significant indicating that increase in team size and use of ICASE tools increases the software effort.

We found that the data quality and its interactions with team size and FP were significant. Lower quality data on team size tended to increase software effort and lower quality data on FP tended to decrease the software effort. Thus, data quality appears to impact the results of our study. In addition to the data quality, there are a few other limitations of our study. We were restricted in terms of variables available in our data sets. For example, we did not have any information on team expertise and project schedules. In the lack of team expertise information, we made a restrictive assumption that the team skills are uniformly distributed. The team expertise information, if available, may have allowed us to better assess the impact of team size on software effort. Project schedules and pressures associated with tight schedules are known to impact software effort [28]. Since our data set did not contain information of project schedules, we assumed that project schedules do not impact software effort. We believe that this is also a restrictive assumption and may have limited the generalizability of our study.

5 Implications and future work

Software development costs can be predicted and managed effectively. In our study, we developed a theoretical model that can allow researchers and practitioners to understand the factors that impact software development costs. The results of our study indicate that project managers can effectively manage software product development costs by controlling three factors. First, they can choose an appropriate programming language for a software project size. Another choice would be to select appropriate ICASE tools to improve productivity and coordination of the programmers. Finally, they can appropriately select a software development team size.

Given three primary choices to manage software development costs, the results of our study indicate that the use of ICASE tools with large teams should be avoided particularly if a software project manager does not have sufficient information on whether programmers have sufficient prior experience in using ICASE tools. The use of ICASE tools, however, does reduce the software development cost for large size projects. Since more often than not large size projects may contain large team sizes, the biggest challenge for software project managers in these projects would be to assess the impact of ICASE tools on software development cost. Based on our preliminary results, it appears that if software programmers do not have sufficient ICASE tool experience, it is best not to use ICASE tools for large size software projects. A similar contradictory effect was observed for the use of 4GL in large size projects. The use of 4GL in large size projects increases the software effort, but the use of 4GL with large team sizes decreases the software effort.

Given that large size projects are complex and may contain large team sizes, the team member expertise in the use of ICASE tools and 4GL programming language may be the most important determinant of software development cost. One of the limitations of our study was that we did not have any information on the team member expertise on the prior use of ICASE tools and 4GL programming language experience. We believe that this was a primary limitation of our study and we suggest that future studies should capture this variable for appropriate prediction of software development cost. Among other limitations of our study were the data bias of non-CASE and ICASE tools (about 90:10) and measurement errors in terms of data quality. Thus, our model needs to be validated by acquiring more reliable unbiased data with information on team member expertise.

References

1. A. J. Albrecht, Measuring application development productivity, in: *Proceedings of the IBM Applications Development Symposium*, GUIDE/SHARE (1979), pp. 83–92.
2. A. J. Albrecht and J. E. Gaffney, Software function, source lines of code, and development effort prediction: A software science validation, IEEE Transaction on Software Engineering 6 (1983) 639–647.
3. L. Angelis, I. Stamelos and M. Morisio, Building a software cost estimation model based on categorical data, in: *Proceedings of Seventh International Software Metrics Symposium*, London, UK (2001), pp. 4–15.
4. R. D. Banker, S. M. Datar and C. F. Kemerer, A model to evaluate variables impacting the productivity of software maintenance projects, Management Science 37(1) (1991) 1–18.
5. R. D. Banker and C. F. Kemerer, Scale economies in new software development, IEEE Transactions on Software Engineering 15(10) (1989) 1199–1205.
6. R. Banker and S. Slaughter, A field study of scale economies in software maintenance, Management Science 43(12) (1997) 1709–1725.
7. F. Bergernon and J. Y. St-Arnaud, Estimation of information systems development efforts, Information and Management 22 (1992) 239–254.
8. S. Biffl and W. Gutjahr, Influence of team size and defect detection technique on inspection effectiveness, in: *Proceedings of Seventh International Software Metrics Symposium*, London, UK (2001), pp. 63–75.
9. J. Blackburn, G. Scudder, L. Van Wassenhove and C. Hill, Time based software development, Integrated Manufacturing Systems 7(2) (1996) 35–45.
10. J. D. Blackburn, G. D. Scudder and L. N. Van Wassenhove, Improving speed and productivity of software development: A global survey of software developers, IEEE Transactions on Software Engineering 22(12) (1996) 875–885.
11. B. W. Boehm, Understanding and controlling software costs, IEEE Transactions on Software Engineering 14(10) (1988) 1462–1477.
12. B. W. Boehm, C. Abts and S. Chulani, Software development cost estimation approaches: A survey, Annals of Software Engineering 10(1) (2000) 177–205.
13. B. W. Boehm, B. Clar, C. Horowitz, C. Westland, R. Madachy and R. Selby, Cost models for future software life cycle processes: COCOMO 2.0.0, Annals of Software Engineering 1(1) (1995) 1–30.
14. L. C. Briand and J. Wüst, Modeling development effort in object-oriented systems using design properties, IEEE Transactions on Software Engineering 27(11) (2001) 963–986.
15. M. A. Cusumano and C. E. Kemerer, A quantitative analysis of U.S. and Japanese practice and performance in software development, Management Science 16(11) (1990) 1384–1406.
16. M. A. Cusumano and R. W. Selby, *Microsoft Secrets* (New York, NY: The Free Press, 1995).
17. G. R. Finnie, G. E. Witting and J. M. Dersharnais, Estimation software development effort with case-based reasoning, in: *The 2nd International Conference on Case-Based Reasoning* (ICCBR-97), Providence, RI (1997), pp. 13–32.
18. L. Fried, Team size and productivity in systems development, Journal of Information Systems Management 8(3) (1991) 27–41.
19. W. Gregory and W. Wojtkowski, *Applications Software Programming with Fourth-Generation Languages* (Boston: Boyd and Fraser, 1990).
20. F. J. Heemstra, Software cost estimation models, in: *Proceedings of the 5th Jerusalem Conference on Information Technology (Cat. No.90TH0326-9), Next Decade in Information Technology* (1990), pp. 286–297.
21. T. E. Hastings and A. S. Sajeev, A vector-based approach to software size measurement and effort estimation, IEEE Transactions on Software Engineering 27(4) (2001) 337–350.
22. R. Jeffery, M. Ruhe and I. Wieczorek, A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data, Information and Software Technology 42 (2000) 1009–1016.
23. C. F. Kemerer, Progress, obstacles, and opportunities in software engineering economics, Communications of the ACM 41(8) (1998) 63–66.
24. S. Koch and G. Schneider, Effort, cooperation and coordination in an open source software project: GNOME, Information Systems Journal 12 (2002) 27–42.

25. A. Leon-Garcia, *Probability and Random Processes for Electrical Engineering* (Reading, Massachusetts: Addison-Wesley Publishing Company, 1994).
26. M. A. Mahmood, K. J. Pettingell and A. I. Shaskevich, Measuring productivity of software projects: A data envelope analysis approach, Decision Sciences 27(1) (1996) 57–80.
27. P. Mimno, Power to the users, Computerworld (April 8, 1985) 11–28.
28. T. L. Roberts, M. L. Gibson, K. T. Field and K. Rainer, Factors that impact implementing a system development methodology, IEEE Transactions on Software Engineering 24(8) (1998) 640–649.
29. R. K. Smith, J. F. Hale and A. S. Parrish, An empirical study using task assignment patterns to improve the accuracy of software effort estimation, IEEE Transactions on Software Engineering 27(3) (2001) 264–271.
30. G. H. Subramanian and G. E. Zarnich, An examination of some software development effort and productivity determinants in ICASE tool projects, Journal of Management Information Systems 12(14) (1996) 143–160.
31. C. D. Wrigley and A. S. Dexter, A model of measuring information system size, MIS Quarterly 15(2) (1991) 245–257.