Integrate the GM(1,1) and Verhulst Models to Predict Software Stage-Effort

Yong Wang[1], Qinbao Song[1,4], Stephen MacDonell[2], Martin Shepperd[3], and Junyi Shen[1]
[1]Department of Computer Science and Technology, Xi'an Jiaotong University, China
wangyong@mailst.xjtu.edu.cn, qbsong@mail.xjtu.edu.cn, jyshen@mail.xjtu.edu.cn
[2]School of Computing and Mathematical Sciences,
Auckland University of Technology, New Zealand
stephen.macdonell@aut.ac.nz
[3]School of IS, Computing & Maths,
Brunel University, UK
martin.shepperd@brunel.ac.uk
[4]State Key Laboratory of Software Engineering,
Wuhan University, China

Abstract

BACKGROUND: Software effort prediction clearly plays a crucial role in software project management. PROBLEM: In keeping with more dynamic approaches to software development it is not sufficient to only predict the whole-project effort at an early stage. Rather, the project manager must also dynamically predict the effort of different stages or activities *during* the software development process. This can assist the project manager to re-estimate effort and adjust the project plan, thus avoiding effort or schedule overruns. METHOD: This paper presents a method for software physical time stage-effort prediction based on grey models GM(1,1) and Verhulst. This method establishes models dynamically according to particular types of stage-effort sequences and can adapt to particular development methodologies automatically by using a novel grey feedback mechanism. RESULT: We evaluate the proposed method with a large-scale real world software engineering data set and compare it with the linear regression method and the Kalman filter method, revealing that accuracy has been improved by at least 28% and 50%, respectively. CONCLUSION: The results indicate that the method can be effective and has considerable potential. We believe that stage predictions could be a useful complement to whole-project effort prediction methods.

1. Introduction

Accurate and unbiased software effort prediction is an important contributor to effective software project management [1-7]. Whole-project effort prediction is clearly important in terms of enabling software developers/managers to make a reasonable bid or form a plan of activities – consequently an extensive body of research has addressed this facet of project management, see for example [1, 4, 5, 8-11]. On the other hand, the capability to predict the effort required in different stages during the software development process is also important but rather less

studied [3, 10, 12-16]. Such a capability enables (or at least should enable) project managers to identify potential effort overrun risk *during the project* and to re-allocate resources when necessary. It has been estimated that around 75% of all projects overrun their schedules due to inaccurate effort prediction [5]. The development and use of software stage-effort prediction methods have the potential to ensure that predictions are revisited and revised on an ongoing basis, the result being that the accuracy of predictions should improve.

This is not to say, however, that the accurate prediction of software stage-effort is straightforward. The software development process can proceed in a sporadic manner despite the best laid plans. It is influenced by many uncertain and challenging-to-measure factors, such as individuals' levels of expertise, project difficulty and technical complexity [3, 14, 17, 18]. Moreover, software development is largely a continuous and cumulative process. The work of prior stages forms the basis of current and subsequent stages, and there is evidence to suggest that there are some inherent relationships between the effort of prior and subsequent stages [3, 10, 12, 13] – although the nature of those relationships is not yet clear. This lends motivation to the idea of using prior stage-effort to predict subsequent stage-effort, a prediction process that should be continuous and dynamic in line with contemporary approaches to software development.

The stage-effort prediction problem has been addressed using a variety of data analysis methods (see Section 2) – among them, linear regression analysis has been popular. In some cases, however, the relationships between different process stages was found to be not particularly strong, leading to some instances of large prediction error. In particular, predictions made in the early stages of a project tend to be more challenging – later-stage predictions can leverage the greater certainty that accrues with progress. Early-stage prediction must be performed in the context of data-starvation, a context that is not conducive to commonly employed statistical methods (such as regression and those based on time-series) and machine learning methods which usually require large data samples to determine statistical features of the series to build prediction models.

Grey System Theory (GST), a system engineering theory based on the uncertainty of small samples, was first proposed by Deng in 1982 [19]. In keeping with the notion of a black box representing a system whose internal workings are not visible, a system about which we only know *some* information (mechanism, relationship, structure, connotation and behavior data), is called a grey system. GST has a distinct advantage over the techniques described above i.e. it can enable the establishment of a prediction model using just a small amount of known data. In the context of data-starvation, GST is known to be effective and has been widely applied to address real world problems in the domains of energy management [20], mobile communication [21], instrument measurement [22], stock price analysis [23], and image processing [24, 25]. In addition, Song et al. [26] used GST to address the whole-project effort prediction problem in software engineering. They used Grey Relation Analysis (GRA) derived from GST to select more effective feature subsets and similar projects to support a prediction process. Their results showed that GRA

can resolve the effort prediction problem with high prediction accuracy. Encouraged by this successful work, we now explore the stage-effort prediction problem using GST in this research.

As we know, there are many project life cycle models used in the software development domain such as iterative cycles, traditional waterfall models and so forth. These in turn are split into different phases e.g. feasibility, high level design and so on. Each of these phases can have complex and on occasions ill-defined mappings to physical time units (such as week, month, and quarter). These time units have most generality and are clearly very important in project scheduling. For this reason, in this paper, we focus on software physical time unit which for clarity we refer to as stage-effort prediction. This is in contrast to prediction based on phases such as in MacDonell and Shepperd [12] that used a seven phase classification scheme.

We propose a method named GV (GM(1,1) and Verhulst). GV takes full advantage of two grey models of GST – GM(1,1) and Verhulst – to predict future stage-effort in light of the law derived from records of prior stage-effort, and it can adapt to particular development methodologies automatically by using a novel grey feedback (GFB) mechanism. We validate the method on a large scale software engineering data repository and obtain very promising results.

The remainder of this paper is organized as follows. In the next section we discuss related work on, and then present the basic concepts and the prediction principles of the Grey Models, followed by the introduction of the proposed stage-effort prediction method GV and an example. After that we describe the data sets and the experimental method we used. The results are then described and discussed, with a concluding discussion presented in the last section.

2. Related work

Surprisingly the empirical study of software stage-effort prediction during the development process has received limited attention. The authors are aware of only four studies that have investigated this issue empirically (one being a previous study involving two of the authors [12]). Kulkarni et al. [13] employed a form of transformation matrix, referred to as a mini-model, to predict effort for each life-cycle phase. By combining the set of distinct-phase mini-models into a single overall model, it is possible to determine the output measures from the final phase using input measures to the first phase. This method was applied to a military project by Kulkarni et al., and while it appeared to be potentially effective, prediction accuracy data was not presented. Ohlsson and Wohlin [24] used artifact-based proxies of project scope to augment existing within-project prediction and planning processes. They concluded that the approach has potential in making plans, and deviations from those plans, more visible, and that it can increase managers' confidence that predictions are of the right order. MacDonell and Shepperd [12] predicted life-cycle phase effort for 16 software projects using a simple linear regression method. They used prior-phase effort data to predict the effort needed for subsequent phases in each of the projects. The results showed that the method produced better predictions than those provided by the project managers. A study reported by Abrahamsson et al. [27] utilized

regression and neural network methods to generate iterative prediction and planning models suitable for projects developed using agile methods. They concluded that the approach improves prediction when compared to whole-project efforts and that such predictions are stable and convergent, attributes that are essential in terms of effective planning.

A variety of techniques, model inputs, and stage units were used in the studies just described. Compared with these studies, our method – GV – has the following differences or characteristics:

1) GV is capable of predicting project effort for physical time stages. The other studies depend on the use of the life cycle phase [12, 13, 15] or the iterative development cycle [27];

2) GV uses the inherent trend embedded in the prior stage-effort to predict. It does not need the historical data of outputs to build the model. The other studies focused on utilizing the relationships between input and output measures to build the models. MacDonell and Shepperd [12] examined the correlations between phases (e.g. between design and implementation); Ohlsson and Wohlin [15] mapped artifact-based proxies to effort; Abrahamsson et al. [27] used linear regression and neural network methods to express the relationships between inputs and outputs; Kulkarni et al. [13] also constructed a matrix-based model to reflect relationships between inputs and outputs;

3) GV requires only project effort itself as input. MacDonell and Shepperd [12] also used project effort as input. Abrahamsson et al. [27] used several estimated predictor variables and the effort of previous iterations as inputs. Ohlsson and Wohlin [15] used their artifact-based proxies as inputs, and Kulkarni et al. [13] used object measures (e.g. source lines of code, Ada packages, data flows).

In short, GV is a novel method to address the problem of stage-effort prediction in software; in principle, however, it would appear to have potential. We now provide an introduction to grey models and to the GV approach.

3. Grey Model

GST uses grey models to make predictions. In this section, we will introduce the basis concepts and two grey models used in this paper, respectively.

3.1. The basis

Generally, the non-negative raw data sequence has no obvious patterns in uncertain circumstances [19], it is difficult to find a proper curve to fit it; but after an accumulated generating operation (AGO), the generated sequence will became monotone increasing and reflect a strong exponential character. We call this the “accumulated generating grey exponential law”. So we can find an optimum exponential curve to simulate it. The AGO is defined as follows:

Let $X^{(0)}$ be the original non-negative sequence, the AGO sequence $X^{(1)}$ can be generated as:

$$x^{(1)}(k) = \sum_{i=1}^{k} x^{(0)}(i); \quad k = 1, 2, ..., n \tag{1}$$

For example, let $X^{(0)}$=(3, 5, 4, 7), then $X^{(1)}$=(3, 8, 12, 19). If the generating exponential character is not obvious, further AGOs may be applied. For returning the data to the original condition the inverse accumulated generating operation (IAGO) will be applied. So, AGO, and IAGO are a pair of inverse sequence operators. The operation of IAGO is defined as follows:

$$\begin{aligned} &x^{(0)}(1) = x^{(1)}(1) \\ &x^{(0)}(k+1) = x^{(1)}(k+1) - x^{(1)}(k);\ k = 1, 2, ..., n \end{aligned} \tag{2}$$

In GST, the curves used to fit AGO sequences can be represented by differential equations. We refer to these differential equations as Grey Models. There are many kinds of grey models available according to the different kinds of AGO sequences, such as GM(1,1), Verhulst, DGM, and so on. Among them, the GM(1,1) is commonly used to simulate exponential-type sequences, therefore it is suitable to describe any monotonic increasing procedure. Verhulst is suggested to simulate sequences with saturated trend. The typical curves of these two models [28] are illustrated in Fig. 1. Where k denotes the sequence element index and $x^{(0)}(k)$ denotes the sequence element value.

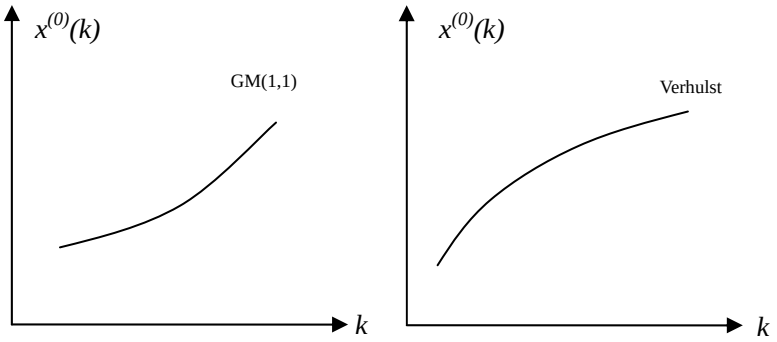

Fig. 1. Curves of model GM(1,1) and Verhulst

3.2. GM(1,1) Model

A form of single variable and first-order linear dynamic differential equation, the definition of the GM(1,1) model is:

$$\frac{dX^{(1)}}{dt} + aX^{(1)} = b \tag{3}$$

where $X^{(1)}$ is the AGO sequence of the original sequence, the parameters a and b are called the development coefficient and grey action quantity, respectively, which can be obtained from the following expression:

$$\hat{a} = \begin{bmatrix} a \\ b \end{bmatrix} = (B^T B)^{-1} B^T Y_N \tag{4}$$

where

$$B = \begin{bmatrix} -z^{(1)}(2) & 1 \\ -z^{(1)}(3) & 1 \\ \ldots & \ldots \\ -z^{(1)}(n) & 1 \end{bmatrix} \tag{5}$$

$$Y_N = \begin{bmatrix} x^{(0)}(2) \\ x^{(0)}(3) \\ \ldots \\ x^{(0)}(n) \end{bmatrix} \tag{6}$$

and

$$z^{(1)}(k) = \frac{x^{(1)}(k) + x^{(1)}(k-1)}{2}, \quad k = 2,3,\ldots,n \tag{7}$$

Then the sequence $X^{(1)}$ can be written as follows:

$$\hat{x}^{(1)}(k+1) = \left(x^{(0)}(1) - \frac{b}{a} \right) e^{-ak} + \frac{b}{a}, \quad k = 1,2,\ldots,n \tag{8}$$

Where $\hat{x}$ denotes the prediction of x. $\hat{x}^{(0)}(k+1)$ can be obtained by IAGO:

$$\hat{x}^{(0)}(k+1) = \hat{x}^{(1)}(k+1) - \hat{x}^{(1)}(k), \quad k = 1,2,\ldots,n \tag{9}$$

3.3. Verhulst Model

The Verhulst model is a form of single variable and second-order differential equation. The definition is:

$$\frac{dX^{(1)}}{dt} + aX^{(1)} = b(X^{(1)})^2 \tag{10}$$

where a and b can be obtained by:

$$\hat{a} = \begin{bmatrix} a \\ b \end{bmatrix} = (B^T B)^{-1} B^T Y_N \tag{11}$$

where

$$B = \begin{bmatrix} -z^{(1)}(2) & (z^{(1)}(2))^2 \\ -z^{(1)}(3) & (z^{(1)}(3))^2 \\ \ldots & \ldots \\ -z^{(1)}(n) & (z^{(1)}(n))^2 \end{bmatrix} \tag{12}$$

$$Y_N = \begin{bmatrix} x^{(0)}(2) \\ x^{(0)}(3) \\ \ldots \\ x^{(0)}(n) \end{bmatrix} \tag{13}$$

and

$$z^{(1)}(k) = \frac{x^{(1)}(k) + x^{(1)}(k-1)}{2}, \quad k = 2,3,\ldots,n \tag{14}$$

The solutions of equation (10) can be expressed as follows:

$$\hat{x}^{(1)}(k+1)=\frac{ax^{(1)}(0)}{bx^{(1)}(0)+(a-bx^{(1)}(0))e^{ak}},\quad k=1,2,...,n \tag{15}$$

Notice that in practice the saturated sequence already has some degree of exponential form, so the original sequence can be regarded as $X^{(1)}$ directly, that means, we need not apply AGO to $X^{(0)}$ to obtain $X^{(1)}$. And $X^{(0)}$ in Equ. (13) should be the IAGO sequence of the original sequence.

For the Verhulst model, the prediction values of the original sequence can be obtained from Equ. (15).

4. Analysis of software stage-effort sequences

The characteristics of software stage-effort sequences [1] are influential in determining the use of appropriate prediction methods. In this section, we introduce some general characteristics of software stage-effort sequences and demonstrate some preliminary analysis of subsequences[2] and sequence sets[3].

4.1. Overview

Software stage-effort sequences are records of the software development process. Investigation of stage-effort sequence data used in this study reveals that around 97% of all projects comprise fewer than 24 stages, where a stage represents one month. So software stage-effort sequences are finite and generally quite short in length. In addition, in many cases, software stage-effort sequences do not exhibit a regular shape over the duration of the project, such as the bell-shaped curve or similar, as we might imagine. In fact, they almost have no typical patterns. Fig. 2 provides an illustrative depiction of this scenario, using some of the software stage-effort data analyzed in this study.

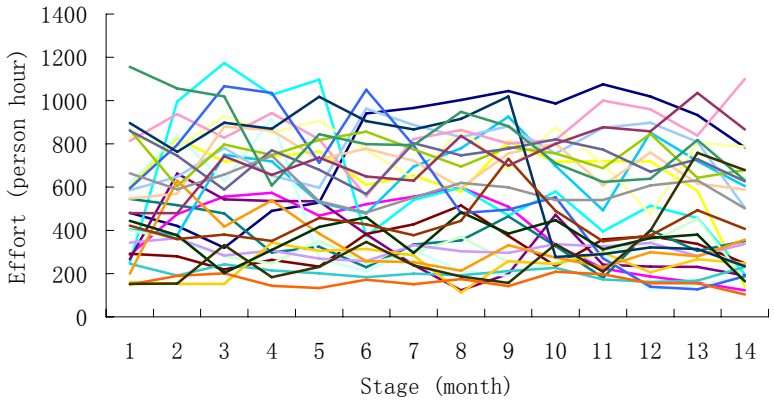

Fig. 2. Curves of some software stage-effort sequences

[1] Sequence: a series of physical time stage-effort values of a software project.
[2] Subsequence: a subset of a software stage-effort sequence for building prediction model.
[3] Sequence set: a collection of software stage-effort sequences.

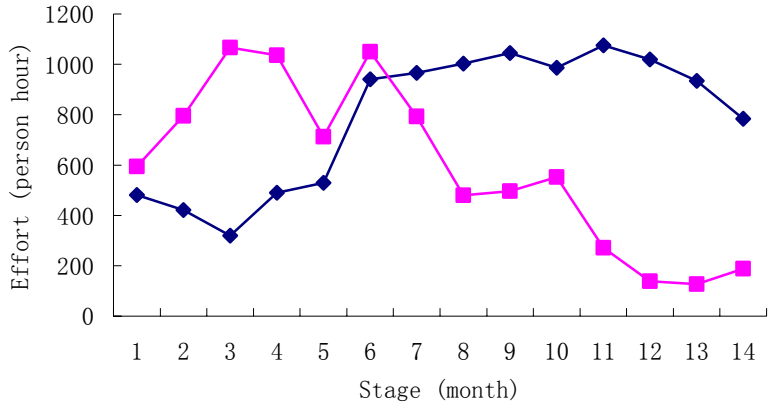

Fig. 3. Stage-effort curves of two projects

From Fig. 2 we note that at least some of the sequences are very irregular; however, because all of the sequences are plotted on this single graph it is difficult to identify any consistent patterns within or among them. For ease of observation and comment we pick out two curves at random from Fig. 2 and show them in Fig. 3. We find that the trend over the whole sequence changes frequently. Within segments of the two sequences, however, the trend demonstrates a degree of stability. In the absence of any consistent whole-sequence patterns, it may be that subsequences can be used to reflect changes in stage-effort and can form the basis for the building of accurate effort prediction models.

4.2. Subsequences

Usually more recent data points are more important in predicting the next data point, so we use recent stage-effort data points to compose each model subsequence. How many stage-effort observations should we use to build a prediction model? In the early stages of the software development process, the number of stage-effort values available is small. A grey model can establish a prediction model with small data samples, but usually more data is helpful. On the other hand, a model that employs many past observations is not always sensitive to data that is liable to frequent and/or significant change, or can incorporate obsolete values in terms of their relevance to future predictions. Based on these considerations we use subsequences consisting of the most recent three stage-effort data values.

We find that a subsequence with three elements can be classified into one of four classes according to its trend and shape. Fig. 4 portrays the four classes of subsequences with characteristics abstracted.

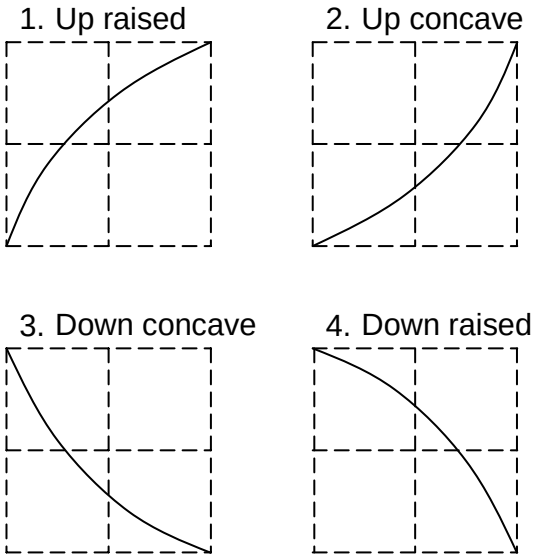

Fig. 4. The four classes of stage-effort subsequences

Considering the sequence representations in Fig. 4 in relation to the model types referred to in Fig. 1, we note that curves 1 and 2 are similar to the curves of Verhulst and GM(1,1), respectively. It is also evident that curves 1 and 3 and curves 2 and 4 are symmetrical pairs. If we flip the class 3 or 4 curve in an up/down direction, the curve adopts the form of curve 1 or 2, so they can be dealt with as per curve 1 or 2. So we can use GM(1,1) or Verhulst to build an appropriate prediction model dynamically according to the types of subsequences identified.

4.3. Sequence set

It is more likely that a common law may be found from a sequence set than from a single sequence. Appropriately selected and classified historical projects will be helpful in software effort prediction [4, 10, 26]. So we group the projects used in this study according to their development methodologies and hope to find typical patterns in each group. It is reasonable that projects complying with same development process model will have similar stage-effort sequence shape. Unfortunately, however, even within the same group, it may be difficult to find a "standard" shape (as noted in [12]).

Although prior research has failed to find clear laws by observation [29, 30], these works have shown that, using GM(1,1) or Verhulst grey models to make predictions using historical project data grouped by development methodology (or industry and so on), the mean prediction biases for each group are significantly different. These differences can be regarded as accumulate effects of the particular methodology applying on the predictions. On the other hand, it indicates the grey models don't differentiate between the methodologies. Adopting a system view, we can regard the stage-effort sequences as inputs, the grey models as a process system, and the biases as outputs. We can then use the mean biases as feedback to adjust the system, to obtain more optimal and consistent outputs. The feedback has relations with the development methodologies but its components are not completely clear, so we refer to it as "grey feedback" or GFB. Fig. 5 depicts the mechanism of GFB.

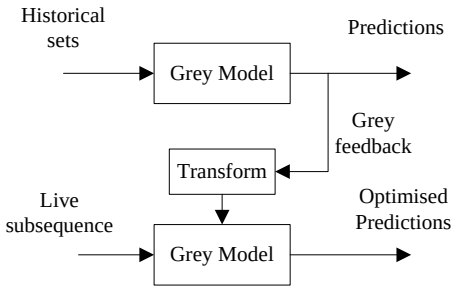

Fig. 5. The grey feedback mechanism.

5. The proposed software stage-effort prediction method GV

5.1. General method

In general, GV uses subsequences to build the grey models, and uses sequence sets to obtain GFB (grey feedback). The grey models are then used to make predictions and GFB is acted on in terms of prediction adjustment, leading to the production of an optimal prediction. So GV leverages both local (within sequence) and global (sequence set) information to produce predictions. Assume that a live project has a stage-effort sequence, $\{x(1), x(2), ..., x(n)\}$, $(n \geq 3)$, GV can use the most recent three stage-effort values $x(n-2)$, $x(n-1)$, and $x(n)$ to predict the effort of stage $n+1$, $x(n+1)$. The prediction procedure of GV includes the following steps:

Step 1: Construct the model subsequence. Use the most recent three stage-effort data values $x(n-2)$, $x(n-1)$, and $x(n)$ as model subsequence;

Step 2: Smooth the subsequence. Use the modified moving average of order 3 to eliminate unwanted fluctuations of the subsequence (see subsection 5.2 for details);

Step 3: Determine the subsequence type according to the trend (up/down) and shape (raised/concave) (see subsection 5.3 for details);

Step 4: Establish a grey model dynamically according to the subsequence type and produce a preliminary prediction value $\hat{x}(n+1)$ (see subsection 5.4 for details);

In order to facilitate the following description, we represent Steps 1 to 4 as a function:

$$\hat{x}(n+1) = Grey\_Predict(x(n-2), x(n-1), x(n)),\ n \geq 3 \tag{16}$$

Where the input to the function is the subsequence, and the output is the prediction value of stage-effort for the next stage $n+1$.

Step 5: Adjust the prediction result using the Stage-effort Adjustment Coefficient (*SAC)* (see subsection 5.5 for details), then obtain the optimal prediction:

$$\hat{x}_{opt}(n+1) = \hat{x}(n+1) \times SAC \tag{17}$$

For ease of description, we denote the overall GV prediction procedure (from Steps 1 to 5) by a function:

$$\hat{x}_{opt}(n+1) = GV(x(n-2), x(n-1), x(n)),\ n \geq 3 \tag{18}$$

where $\hat{x}_{opt}(n+1)$ is the final effort prediction value of stage n+1. Fig. 6 portrays the details of the general prediction procedure of GV.

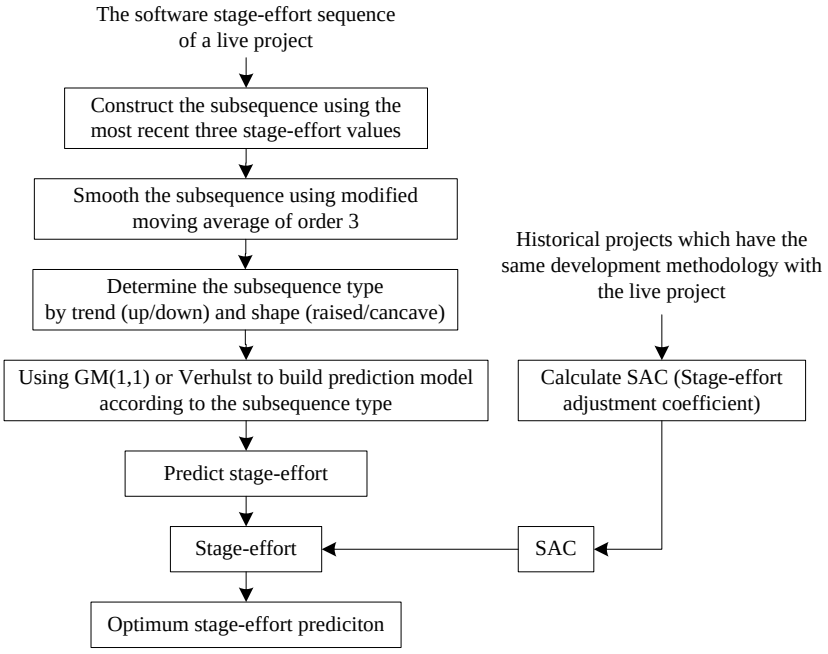

Fig. 6. Prediction procedure of GV

5.2. Modified moving average of order 3

Smooth transformation can reduce noise-based fluctuations in the sequences. A common transformation method is the weighted moving average of order k [31]:

$$\frac{\omega_1 y_1 + \omega_2 y_2 + ... + \omega_k y_k}{\omega_1 + \omega_2 + ... + \omega_k}, \frac{\omega_2 y_2 + \omega_3 y_3 + ... + \omega_{k+1} y_{k+1}}{\omega_2 + \omega_3 + ... + \omega_{k+1}}, \frac{\omega_3 y_3 + \omega_4 y_4 + ... + \omega_{k+2} y_{k+2}}{\omega_3 + \omega_4 + ... + \omega_{k+2}}, ... \tag{19}$$

where $Y = (y_1, y_2, y_3, \cdots)$ is a given sequence, and ω_i is the weight. After transformation k-2 head and tail data are discarded. This would likely raise problems in software stage-effort prediction as in general the stage-effort data of a software project is limited, particularly in the early stages of development. The above transformation makes small data samples even smaller, perhaps of insufficient size to construct a prediction model. Therefore we modify Equ. (19) by reserving the head and tail data. For example, the weighted moving average of order 3 used in this paper can be treated as:

$$\frac{3y_1 + y_2}{4}, ..., \frac{y_{k-1} + 2y_k + y_{k+1}}{4}, ..., \frac{y_{n-1} + 3y_n}{4} \tag{20}$$

Specifically, for a sequence of three elements (y_1, y_2, y_3), the transformed sequence is:

$$\frac{3y_1+y_2}{4}, \frac{y_1+2y_2+y_3}{4}, \frac{y_2+3y_3}{4} \tag{21}$$

The three elements can form a folding line. The basic shape of the folding line will not change after the transformation, and a straight line will still be a straight line after the transformation. Fig. 7 illustrates the original and transformed sequences where the *x*-axis denotes the indices of the elements and the *y*-axis denotes the values. The circles represent the original data and the squares represent the transformed values.

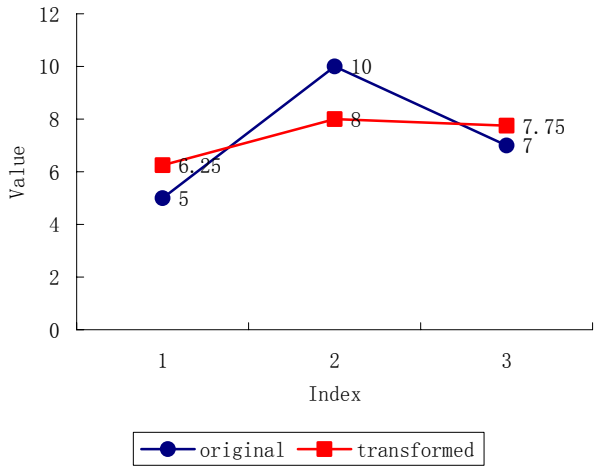

Fig. 7. Modified moving average of order 3

From Fig. 7 we can see that the transformed sequence is more regular than the original but that the original trend is preserved.

5.3. Determination of subsequence type

We assume the subsequence is composed of (y_1, y_2, y_3). The type determination procedure includes two stages. Fig. 8 shows in pseudocode the detailed determination procedure of subsequence type.

```
Procedure: Determine the stage-effort subsequence type
Input: subsequence, (y1, y2, y3)
Output: SEQUENCE_TYPE, subsequence type from 1 to 4
1)  if y3 > y1
2)      // up sequence
3)      if y2 >(y1+y3)/2                    // raised sequence
4)          SEQUENCE_TYPE = 1;
5)      else                                // concave sequence or straight line
6)          SEQUENCE_TYPE = 2;
7)      end if
8)  else
9)      // down or horizontal sequence
10)     if y2 <(y1+y3)/2                    // concave sequence
11)         SEQUENCE_TYPE = 3;
12)     else                                // raised sequence or straight line
13)         SEQUENCE_TYPE = 4;
```

```
14)        end if
15)  end if
```

Fig. 8. Determination of subsequence type procedure

First, classify the subsequence into up or down classes by comparing y_1 with y_3. If $y_3 > y_1$, we say that the subsequence has a total up trend, if $y_3 < y_1$, we say it has a total down trend, if $y_3 = y_1$ (horizontal), we regard it as a down trend (steps 1 and 8 respectively in Fig. 8).

We then classify the subsequence into raised or concave classes by comparing y_2 with the median of y_1 and y_3, i.e., $(y_1+ y_3)/2$. 1) For an up class subsequence, if $y_2 > (y_1+ y_3)/2$, we say the subsequence is raised, type is 1 (steps 3, 4); if $y_2 < (y_1+ y_3)/2$, we say it is concave, type is 2. When $y_2 = (y_1+ y_3)/2$, it is a straight line, we include it in the concave class (steps 5, 6). 2) For a down class subsequence (including $y_3 = y_1$), if $y_2 < (y_1+ y_3)/2$, we say the subsequence is concave, type is 3 (steps 10, 11), if $y_2 > (y_1+ y_3)/2$, we say it is raised, type is 4. When $y_2 = (y_1+ y_3)/2$, it is a straight line, we include it in the raised class (steps 12, 13).

5.4. Prediction model

GV establishes models dynamically according to the types of stage-effort subsequence. Considering the subsequence representations in Fig. 4 in relation to the model types referred to in Fig. 1, we note that curve 1 is up raised, that is, a saturated sequence, so a Verhulst model can be used to represent it. Curve 2 is up concave, that is, an exponential sequence, and so is theoretically suitable to be predicted by GM(1,1). Curves 3 and 4 are down sequences which cannot use GM(1,1) or Verhulst directly, because GM(1,1) and Verhulst require the sequences to have a total upward trend. So we must first convert these curves to up style sequences. In fact, as noted, it is clear that curves 1 and 3 and curves 2 and 4 are symmetrical pairs. If we flip the class 3 or 4 curve in an up/down direction, it adopts the form of curve 1 or 2. We can then use GM(1,1) or Verhulst to model and predict. After predicting, we need to flip again to ensure that the models match the original sequences. The detailed procedure is shown in Fig. 9. In summary, the classes and suitable models for various types of original data subsequences are listed in Table 1.

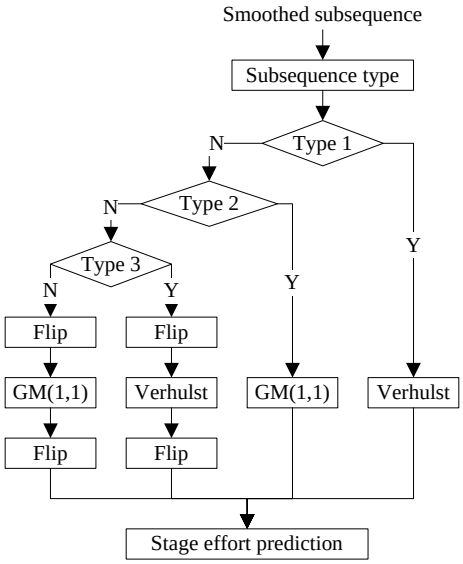

Fig. 9. The model and prediction procedure

Table 1 Types and treatments of stage-effort subsequences

Subsequence type	Class of original subsequence	Suitable prediction model
1	Saturated	Verhulst
2	Exponential	GM(1,1)
3	Saturated after up flipping	Verhulst
4	Exponential after up flipping	GM(1,1)

5.5. Obtaining SAC and GFB

SAC (Stage-effort Adjustment Coefficient) forms a bridge between the grey models and particular development methodologies. Using it, GV can fit the particular stage-effort sequences better and obtain more accurate results. In prior research [29, 30], a bias correction method has been used, but it tends to over-fit. In this study we make improvements based on the prior work and propose a novel adjustment coefficient *SAC* as follows:

$$SAC = \frac{e^{\frac{GFB}{N}}}{1-GFB} \tag{22}$$

where N is the length of a GV model subsequence, in this study, it is equal to three. *GFB* is intended here as a form of grey feedback pertaining to particular development methodologies (see subsection 4.3 for details). It can be estimated using the mean biases of predictions of historical stage-effort sequences.

To obtain *GFB*, we need a set of historical projects that comply with certain development methodologies. Each project has a stage-effort sequence and is regarded as a live project. First we use Equ. (16) to make predictions stage by stage for each project, then compute the bias of each prediction. This process iterates until all projects are processed. Finally we average all the biases and assign the result to *GFB*.

The procedure for obtaining *GFB* is summarized in Fig. 10.

Function: GetGFB, obtain the *GFB* (grey feedback) from a set of software stage-effort sequences.
Input: a set of stage-effort sequences.
Output: *GFB*

1) $j = 1$;
2) **for** each stage-effort sequence $\{x(1), x(2), ..., x(n)\}$, $(n \geq 3)$
3) **for** each i from 3 to n-1
4) $\hat{x}(i+1)$ = Grey_Predict($x(i-2)$, $x(i-1)$, $x(i)$); //see Equ. (16)
5) $Bias(j) = \frac{x(i+1) - \hat{x}(i+1)}{x(i+1)}$;
6) $j = j + 1$;
7) **end for**
8) **end for**
9) $GFB = \frac{1}{j-1} \sum_{k=1}^{j-1} Bias(k)$;

Fig. 10. The GFB obtaining function

5.6. An example

We now demonstrate the complete prediction procedure for a 'live' project using the GV method. The initial condition is: a live project that has been in progress for 3 stages (three months), stage-effort values (in person-hours) of 260, 101, 450; and *SAC* = 0.76. The prediction procedure of the effort needed to complete the fourth stage (the fourth month) is as follows.

Step 1: Construct the model subsequence using the three most recent stage-effort values. The subsequence is $X^{(0)} = (x^{(0)}(1), x^{(0)}(2), x^{(0)}(3)) = (260, 101, 450)$;

Step 2: Smooth the subsequence. Use Equ. (21) to transform the subsequence into: $X^{(0)} = (x^{(0)}(1), x^{(0)}(2), x^{(0)}(3)) = (220, 228, 363)$;

Step 3: Determine the subsequence type. According to subsection 5.3, because $x^{(0)}(1) < x^{(0)}(3)$ and $x^{(0)}(2) < \frac{x^{(0)}(1) + x^{(0)}(3)}{2}$, so $X^{(0)}$ is an up concave sequence, belongs to type 2, and according to Table 1, the suitable prediction model is GM(1,1);

Step 4: Build the prediction model. According to the building procedure of GM(1,1) (see subsection 3.2 for details), first apply AGO on $X^{(0)}$, obtain the AGO sequence: $X^{(1)} = (x^{(1)}(1), x^{(1)}(2), x^{(1)}(3)) = (220, 448, 811)$. Then we can get $Z^{(1)}$, B, and Y_N:

$$Z^{(1)} = \begin{bmatrix} z^{(1)}(2) \\ z^{(1)}(3) \end{bmatrix} = \begin{bmatrix} \dfrac{x^{(1)}(2) - x^{(1)}(1)}{2} \\ \dfrac{x^{(1)}(3) - x^{(1)}(2)}{2} \end{bmatrix} = \begin{bmatrix} 114 \\ 181.5 \end{bmatrix},$$

$$Y_N = \begin{bmatrix} x^{(0)}(2) \\ x^{(0)}(3) \end{bmatrix} = \begin{bmatrix} 228 \\ 363 \end{bmatrix},$$

$$B = \begin{bmatrix} -z^{(1)}(2) & 1 \\ -z^{(1)}(3) & 1 \end{bmatrix} = \begin{bmatrix} -114 & 1 \\ -181.5 & 1 \end{bmatrix},$$

the arguments *a* and *b* can be obtained by Equ. (4):

$$\hat{a} = \begin{bmatrix} a \\ b \end{bmatrix} = (B^T B)^{-1} B^T Y_N = \begin{bmatrix} \text{-0.4569} \\ 75.4112 \end{bmatrix}.$$

So we can get the prediction model of $X^{(1)}$ by Equ. (8):

$$\hat{x}^{(1)}(k+1) = \left(x^{(0)}(1) - \frac{b}{a} \right) e^{-ak} + \frac{b}{a} = 385.0497 e^{0.4569k} - 165.0497 .$$

Let *k*=1, 2, 3, we can get $\hat{X}^{(1)} = (\hat{x}^{(1)}(2), \hat{x}^{(1)}(3), \hat{x}^{(1)}(4)) = (443, 795, 1351)$. Note that $\hat{x}^{(1)}(4)$ corresponds to stage 4 (the stage for which we are predicting).

Step 5: Make the preliminary prediction. According to Equ. (9), we can get the preliminary prediction of stage 4: $\hat{x}^{(0)}(4) = \hat{x}^{(1)}(4) - \hat{x}^{(1)}(3)) = 1351 - 795 = 556$.

Step 6: Apply *SAC* to the preliminary prediction to obtain the optimum prediction. The final prediction value is: $556 \times SAC = 556 \times 0.76 = 422$.

So the prediction effort for the fourth stage of this project is 422 person-hours. After one further month, we obtain the real effort value of the fourth month, then the fifth stage-effort value can be predicted by the known effort of stages 2, 3, and 4. The procedure is executed like this until the project completes overall. Note that *SAC* is obtained from historical projects, but if no historical projects are available, let *SAC* = 1.

6. Experiments and analysis

6.1. Data sources

The data used in this research are drawn from a large scale software engineering data repository[4]. The data contains information relating to around 1900 projects undertaken between 2000 and 2004. The projects come from around 30 countries or areas. The main contributing countries are: the United States, Australia, Sweden, Canada, New Zealand, and the United Kingdom. The application domains range from aerospace, financial, manufacturing, medicine, traffic, and business. The project data in the repository are composed of project properties and effort data. The static

[4] We regret that the data set is subject to a non-disclosure agreement due to its commercially sensitive nature.

properties, such as development methodology, industry, development team, project type, experience levels and so on, are collected once at the beginning of each project. The databases, platforms, CASE tools used and so on are recorded as changes in each occur. Each project is described in terms of a set of concurrent activities which comply with a certain methodology. The effort for each live development activity is recorded monthly. To obtain a general view, we employ the monthly effort of a project (instead of its composite low-level activities) as the figure of interest in each stage in the experiments of this study. The former is the sum of the latter for each month.

As stated previously, GV can adjust itself to suit particular development methodologies, so we organized the projects into four data sets drawn from the underlying data set according to development methodology. Tables 2 and 3 summarize the real raw data stored in the repository and show that the number of projects per methodology ranges from 45 to 864, the monthly effort ranges from 0 to 69,233 person hours. Therefore, the data sets are large (by software engineering standards) and methodologically diverse. It has long been acknowledged that the absence of systematic historical data is a significant obstacle in the software effort prediction domain [17, 32], with experiments and evaluations having to rely on small data samples [6]. The data sets used in this study enable us to evaluate and calibrate the proposed method with data from large numbers and various kinds of projects, and should therefore lead to more robust results and reliable conclusions.

Table 2 Distribution of projects

dataset	Description	Count
1	Combined Dev+ Life Cycle Modules	864
2	Application Development Overlapping Waterfall	567
3	SSP – Software Engineering and Release	390
4	Application Implementation	45

Table 3 Descriptive statistics of project stage-effort (in person hours per month)

dataset	Mean	Std. Deviation	Minimum	Maximum	Median
1	729.17	1701.96	0	37765.50	331.25
2	941.03	3604.13	0	69233.00	312.50
3	437.74	737.15	0	7892.50	194.00
4	649.38	1378.09	0	19845.00	260.00

6.2. Experimental methods

6.2.1. General method

The purpose of the experiments is to evaluate the prediction performance of GV. First, we preprocess the four data sets (see subsection 6.2.2 for details), then for each of the four data sets, we systematically extract 5 pairs of the training and test data sets, and obtain a total of 20-pair training-test data sets. We obtain *GFB* and *SAC* on each training set and evaluate GV on the test set (see subsection 6.2.3 for details). Finally we use a linear regression method and the Kalman Filter method (see subsection 6.2.4

for details) as benchmarks to compare against the performance of GV.

6.2.2. Data Preprocessing

Inspection of the data sets reveals that there is a lot of noisy and inconsistent data. We preprocess the data as follows:

Delete the extreme[5] cost data values from the stage-effort sequences. These values are too small or too big when compared with the other effort values. The stages with such extreme values are abnormal stages and could not be predicted using a generally useful model. For instance, the minimum value of stage-effort is zero and this is considered an extreme value (see Table 3).

6.2.3. Validation method

Cross-validation is a method for estimating generalization error based on "resampling" [33]. We use a 5-fold cross-validation strategy as the validation approach. In 5-fold cross-validation, the dataset D is randomly partitioned into 5 mutually exclusive subsets, D_1, D_2, ..., D_5, each of approximately equal size. The inducer is trained and tested five times. Each time $t \in \{1,2,\mathrm{L}\ ,5\}$, the subset D_t is reserved as the test set, and the remaining subsets $D \ominus D_t$[6] are used as training set.

We obtain *SAC* from the training set (see subsection 5.5 for details) and evaluate GV on the test set. When evaluating, we regard the projects in the test set as live projects, i.e., for each project with stage-effort sequence $\{x(1), x(2), ..., x(n)\},\ (n \geq 3)$, we use $\{x(k-2), x(k-1), x(k)\},\ \text{where } 3 \leq k \leq n\text{-}1$, as the input subsequence, and use Equ. (18) to obtain the prediction value $\hat{x}_{opt}(n+1)$. For a project with *n* stages, we can make *n*-3 predictions. For each prediction, we compare the prediction value with the actual value to obtain evaluation results. In this study, we use the Bias, MMRE, and MdMRE as evaluation measures.

The Bias establishes whether models are biased and tend to over or under prediction. The Bias is defined as follows:

$$Bias_i = \frac{\varepsilon_i - \hat{\varepsilon}_i}{\varepsilon_i} \tag{23}$$

where $\hat{\varepsilon}$ is the prediction value of actual effort ε.

The Magnitude of Relative Error (MRE) is another common criterion for evaluating software effort prediction methods. For a prediction *i*, the corresponding MRE_i is defined as follows:

$$MRE_i = \left| Bias_i \right| \tag{24}$$

By averaging MRE_i over multiple predictions *n*, Mean MRE (MMRE) is

[5] If a value is very large or small compared with its adjacent values in the sequence, then we call it an extreme.

[6] The notation $\mathcal{D} \ominus \mathcal{D}_t$ means set $\mathcal{D}$ minus set $\mathcal{D}_t$.

obtained:

$$MMRE = \frac{1}{n}\sum_{i=1}^{n} MRE_i \tag{25}$$

MMRE is the most frequently used criterion for evaluating software effort prediction methods. However it is known to be sensitive to individual predictions with excessively large MREs. We therefore also use the median of MREs for the *n* predictions (MdMRE), which is less sensitive to extreme values, as another measure. For both MMRE and MdMRE, a higher value means lower prediction accuracy.

6.2.4. Benchmark methods

MacDonell and Shepperd employed a simple linear regression (LR) method in [12], and because there is no other work that can be compared, we use LR as a benchmark. However, as LR often suffers from a leverage effect of outliers [17] we complement it with a further approach. The Kalman filter method (KF) [34] is a well-known series prediction method that can deal with noise and outliers robustly [35], so we also use the Kalman filter as another benchmark method and determine its arguments from the sequences in the training set (the training and test sets KF used are same as those used with GV, see subsection 6.2.3 for details). For fairness, we use the same data sets and subsequences for the three approaches.

As many statistical techniques that deal with the prediction of time series data require large samples (i.e., long sequences), so they are not suitable to be benchmarks here. For example, effective fitting of Box-Jenkins models (often called ARIMA models) [36] typically requires at least 50 observations [37]. This is too many for most software project stage-effort sequences and cannot be satisfied here.

6.3. Experimental results

We conduct the experiments using GV (the GV method), KF (the Kalman filter method) and LR (the simple linear regression method) on the 4 data sets. Table 4 to Table 6 report the accuracy of the respective methods in terms of MMRE, MdMRE, Bias, and the improvements of GV upon KF and LR.

Table 4 MMRE of GV, LR and KF and GV's improvement upon KF and LR with different data sets

Dataset	MMRE (%)			GV's MMRE improvement (%)	
	GV	KF	LR	upon KF	upon LR
1	62.60	80.35	111.29	28.35	77.78
2	61.44	84.98	131.18	38.31	113.51
3	61.52	95.69	134.64	55.54	118.86
4	54.48	77.95	81.63	43.08	49.83

Table 5 MdMRE of GV, LR and KF and GV's improvement upon KF and LR with different data sets

Dataset	MdMRE (%)			GV's MdMRE improvement (%)	
	GV	KF	LR	upon KF	upon LR

1	46.96	59.02	73.87	25.68	57.30
2	44.17	53.89	74.59	22.01	68.87
3	49.81	67.59	85.43	35.70	71.51
4	42.04	57.01	64.31	35.61	52.97

Table 6 Bias of GV, LR and KF and GV's improvement upon KF and LR with different data sets

Dataset	Bias (%)			GV's Bias improvement (%)	
	GV	KF	LR	upon KF	upon LR
1	-0.76	-25.73	-29.04	3285.53	3721.05
2	0.18	-35.83	-34.49	19805.56	19061.11
3	3.22	-40.42	-50.31	1155.28	1462.42
4	5.85	-19.73	-20.98	237.26	258.63

From Table 4 to Table 6 we observe that the MMREs, MdMREs and Biases of GV are superior to those of KF and LR for all four data sets. Further, GV's prediction accuracies across all data sets exhibit smaller differences. This indicates that GV has better consistency and stability on different development methodologies. The small bias values also imply that GV can cope with the differences in development methodologies to obtain optimal results. We also observe that compared with KF and LR, GV's MMREs are lower by at least 28% and 50%, respectively, MdMREs are lower by at least 22% and 53%, respectively, and GV shows very good performance with respect to bias, with values that are lower by at least 237% and 259% than KF and LR, respectively.

In statistics, percentiles are used to describe characteristics of distributions. Here, we used percentiles to explore the distribution of the absolute residuals of prediction accuracy for the three methods with four data sets. Table 7 contains the results. From it we can find that, GV has less values than KF and LR for the 5th, 10th, 25th, 50th, 75th, 90th, and 95th percentiles with all four data sets except for the 75th percentile of dataset 4 LR has a less value. This reveals that GV outperformed both KF and LR.

Table 7 Percentiles of absolute residuals of prediction accuracy for GV, KF and LR with all data sets

dataset	Method	Percentiles						
		5	10	25	50	75	90	95
1	GV	9.4	20.7	57.7	159.1	363.8	764.8	1160.2
	KF	13.0	24.5	64.7	168.7	409.5	858.4	1302.9
	LR	12.8	27.9	75.4	182.4	404.1	804.2	1182.3
2	GV	8.2	18.6	52.7	153.2	375.6	788.8	1462.0
	KF	12.0	24.4	69.4	178.9	413.3	877.8	1465.8
	LR	12.8	28.2	74.0	186.9	417.4	840.3	1480.4
3	GV	4.2	10.4	34.6	94.4	228.7	519.3	801.0
	KF	6.7	17.9	45.6	121.7	286.2	607.5	934.3
	LR	7.1	16.3	47.8	128.6	292.8	584.0	901.9
4	GV	5.3	11.0	34.7	100.4	270.0	540.6	688.0

	KF	8.5	14.3	37.4	104.8	315.6	588.0	861.6
	LR	5.6	12.5	40.5	106.5	265.6	599.9	914.0

Although we have made some observations based on the data presented in the tables above, to rigorously compare the differences between prediction accuracy between the three methods we need to perform statistical significance testing. The MMREs of the three methods do not follow a normal distribution, so we use 1-tailed Wilcoxon Matched-Pairs Signed Ranks tests to examine if there exist significant improvements on the MMREs of GV over the other two methods. Table 8 gives the testing results.

From Table 8 we observe that all *p*-values are less than 0.001. This means that the MMREs of GV are significantly lower (and therefore better) than those of KF and LR.

Table 8 1-tailed Wilcoxon Matched-Pairs Signed Ranks test of MMREs of GV vs KF and GV vs LR on all data sets

Dataset	*p*-value	
	GV vs KF	GV vs LR
1	.000	.000
2	.000	.000
3	.000	.000
4	.000	.000

To summarize, in this study GV outperforms KF and LR on all four data sets and demonstrates considerable potential. The reasons lie in the following:

1) The AGO (accumulated generating operation) procedures of grey models make the original stage-effort sequences more regular, so, easier to fit;

2) The subsequence type recognition procedures makes GV flexible, which means GV can capture the various changing trends and further take full advantage of two grey models;

3) The capability of using *GFB* (grey feedback) from historical projects means GV can utilize global domain knowledge to pilot local predictions along correct directions, avoiding reliance on just small local data sets and maximizing the capability of the method.

7. Conclusions

Dynamic software project stage-effort prediction facilitates the evaluation of potential effort problems, and could provide early warning information thus ensuring that projects are completed within (possibly adjusted) schedules and budgets. In this paper we have proposed a novel approach of using Grey Models of Grey System Theory to address the software stage-effort prediction problem during the development process. The proposed method can predict the future stage-effort using the effort of three most recent continuous stages and can suit particular development methodologies by using a novel grey feedback mechanism.

Our experiments have been conducted on a large scale software engineering data

repository split into four data sets based on development methodology. Because there is no other work that can be compared, we employed the Kalman filter method (KF) and the linear regression method (LR) as benchmarks. The results show that the proposed GV method outperforms KF and LR in terms of MMRE, MdMRE and Bias for all data sets used.

Finally we pose the question of generalization. The data sets are drawn from a repository of medium to large projects from an international software house. They cover a range of application areas such as commerce, government information systems, defense and retail. Of course different organizations may run projects and collect data in different ways. Therefore, it would be interesting to see replication of this study in different environments. Nevertheless this is an encouraging result and shows that the method has considerable potential.

References

[1] M. Jørgensen and M. Shepperd, "A systematic review of software development cost estimation studies," IEEE Transaction on Software Engineering, vol. 33, no. 1, pp. 33-53, 2007.

[2] M. Shepperd, "Software project economics: a roadmap," in Future of Software Engineering (FOSE 2007), Minneapolis, MN, 2007, pp. 304-315.

[3] B.W. Boehm, Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[4] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," IEEE Transactions on Software Engineering, vol. 23, no. 11, pp. 736-743, 1997.

[5] K. Moløkken and M. Jørgensen, "A review of surveys on software effort estimation," in Proceedings of the 2003 International Symposium on Empirical Software Engineering (ISESE 2003), Rome, Italy, 2003, pp. 223-230.

[6] L.C. Briand, T. Langley, and I. Wieczorek, "A replicated assessment and comparison of common software cost modeling techniques," in The 22th International Conference on Software Engineering (ICSE 2000), Limerick, Ireland, 2000, pp. 377-386.

[7] L.C. Briand, K.E. Emam, D. Surmann, I. Wieczorek, and K.D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," in The 21st International Conference on Software Engineering (ICSE 1999), Los Angeles, CA, 1999, pp. 313-323.

[8] M. Jørgensen, "How much does a vacation cost? or what is a software cost estimate?," ACM SIGSOFT Software Engineering Notes, vol. 28, no. 6, pp. 5-5, 2003.

[9] A. Idri, T.M. Khoshgoftaar, and A. Abran, "Can neural networks be easily interpreted in software cost estimation?," in Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on, 2002, pp. 1162-1167.

[10] B.W. Boehm, Clark, Horowitz, Brown, Reifer, Chulani, R. Madachy, and B. Steece, Software Cost Estimation with Cocomo II. New York: Prentice Hall, 2000.

[11] G.R. Finnie and G.E. Wittig, "AI tools for software development effort estimation," in Proceedings of the 1996 International Conference on Software Engineering: Education and Practice (SE:EP 1996), Dunedin, New Zealand, 1996, pp. 346-353.

[12] S. MacDonell and M. Shepperd, "Using prior-phase effort records for re-estimation during software projects," in The Ninth IEEE International Software Metrics Symposium (METRICS 2003), Sydney, Australia, 2003, pp. 73-86

[13] A. Kulkarni, J.B. Greenspan, D.A. Kriegman, J.J. Logan, and T.D. Roth, "A generic technique for

developing a software sizing and effort estimation model," in Proceedings of The 12th International Computer Software and Applications Conference (COMPSAC 1988), 1988, pp. 151-161.

[14] N.E. Fenton and S.L. Pfleeger, Software Metrics, A Rigorous and Practical Approach, 2nd ed. Boston: PWS Publishing Company, 1997.

[15] M.C. Ohlsson and C. Wohlin, "An empirical study of effort estimation during project execution," in The Sixth International IEEE Software Metrics Symposium, Boca Raton, FL, USA, 1999, pp. 91-98.

[16] A. Rainer and M. Shepperd, "Re-planning for a successful project schedule," in The Sixth IEEE International Software Metrics Symposium (METRICS 1999), Boca Raton, FL, 1999, pp. 72-81.

[17] G. Liebchen, B. Twala, M. Shepperd, M. Cartwright, and M. Stephens, "Filtering, robust filtering, polishing: techniques for addressing quality in software data," in The First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, Spain, 2007, pp. 99-106.

[18] M. Jørgensen and S. Grimstad, "Avoiding irrelevant and misleading information when estimating development effort," IEEE Software, vol. 25, no. 3, pp. 78-83, 2008.

[19] J. Deng, "Control problems of grey systems," Systems & Control Letters, vol. 1, no. 5, pp. 288-294, 1982.

[20] T.H.M. El-Fouly, E.F. El-Saadany, and M.M.A. Salama, "Grey predictor for wind energy conversion systems output power prediction," IEEE Transaction on Power Systems, vol. 21, no. 3, pp. 1450-1452, 2006.

[21] S.L. Su, Y.C. Su, and J.F. Huang, "Grey-based power control for DS-CDMA cellular mobile systems," IEEE Transactions on Vehicular Technology, vol. 49, no. 6, pp. 2081-2088, 2000.

[22] K. Lin and B. Liu, "A gray system modeling approach to the prediction of calibration intervals," IEEE Transactions on Instrumentation and Measurement, vol. 54, no. 1, pp. 297-304, 2005.

[23] Y.F. Wang, "On-demand forecasting of stock prices using a real-time predictor," IEEE Transactions on Knowledge and Data Engineering, vol. 15, no. 4, pp. 1033-1037, 2003.

[24] R.C. Luo, T.M. Chen, and K.L. Su, "Target tracking using a hierarchical grey-fuzzy motion decision-making method," IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, vol. 31, no. 3, pp. 179-186, 2001.

[25] C. Jau-Ling and C. Pei-Yin, "An efficient gray search algorithm for the estimation of motion vectors," IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 31, no. 2, pp. 242-248, 2001.

[26] Q. Song, M.J. Shepperd, and C. Mair, "Using grey relational analysis to predict software effort with small data sets," in The 11th IEEE International Software Metrics Symposium (METRICS 2005), Rome, Italy, 2005.

[27] P. Abrahamsson, R. Moser, W. Pedrycz, A.A.S.A. Sillitti, and G.A.S.G. Succi, "Effort prediction in iterative software development processes -- Incremental versus global prediction models," in The First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007), Madrid, Spain, 2007, pp. 344-353.

[28] S. Liu, Y. Dang, and Z. Fang, The Grey System Theory and Application (in Chinese). Beijing: Science Press, 2004.

[29] Y. Wang, Q. Song, and J. Shen, "Grey learning based software stage-effort estimation," in Proceedings of the Sixth International Conference on Machine Learning and Cybernetics (ICMLC

2007), Hongkong, 2007, pp. 1470-1475.
[30] Y. Wang, Q. Song, and J. Shen, "Grey prediction based software stage-effort estimation," Wuhan University Journal of Natural Sciences, vol. 13, no. 5, pp. 927-931, 2007.
[31] J.W. Han and M. Kamber, Data Mining: Concepts and Techniques. San Fransisco, CA: Morgan Kaufmann, 2001.
[32] M. Shepperd and M. Cartwright, "Predicting with sparse data," IEEE Transactions on Software Engineering, vol. 27, no. 11, pp. 987-998, 2001.
[33] J. Shao and D. Tu, The Jackknife and Bootstrap. New York: Springer-Verlag, 1995.
[34] R.E. Kalman, "A new approach to linear filtering and prediction problems," Transaction of the ASME - Journal of Basic Engineering, vol. 82, no. 1, pp. 35-45, 1960.
[35] H. Weiming, T. Tieniu, W. Liang, and S. Maybank, "A survey on visual surveillance of object motion and behaviors," IEEE Transaction on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 34, no. 3, pp. 334-352, 2004.
[36] P. Brockwell and R. Davis, Introduction to Time Series and Forecasting, 2nd ed. New York: Springer-Verlag, 2002.
[37] C. Chatfield, The Analysis of Time Series, 5th ed. New York: Chapman & Hall, 1996.

Yong Wang received a M.S. Degree in Computer Science from the Xi'an Jiaotong University, Xi'an, China, in 2003. He is a Ph.D. candidate in Computer Science at the Xi'an Jiaotong University. His research interests include software cost estimation and software engineering data quality. He previously worked for a research institude in Qingdao, China, and has more than 10 years experience in software development and project management.

Qinbao Song received a PhD in computer science from the Xi'an Jiaotong University, China in 2001. He is professor of software technology in the department of computer science & technology at the Xi'an Jiaotong University and deputy director of the department of computer science & technology at the Xi'an Jiaotong University. He has published more than 70 referred papers in the area of machine learning and software engineering. He is a Board Member of the Open Software Engineering Journal. His research interests include machine learning, empirical software engineering, and trustworthy software.

Stephen MacDonell is Professor of Software Engineering and Director of the Software Engineering Research Laboratory (SERL) at the Auckland University of Technology (AUT) in New Zealand. Stephen undertakes research in software metrics and measurement, project planning, estimation and management, software forensics, and the application of empirical

analysis methods to software engineering data sets. He is a Member of the IEEE Computer Society and the ACM, and serves on the Editorial Board of Information and Software Technology.

Martin Shepperd received a PhD in computer science from the Open University in 1991 for his work in measurement theory and its application to empirical software engineering. He is professor of software technology at Brunel University, London, UK and director of the Brunel Software Engineering Research Centre (B-SERC). He has published more than 100 refereed papers and three books in the areas of software engineering and machine learning. He was editor-in-chief of the journal Information & Software Technology (1992-2007) and was Associate Editor of IEEE Transactions on Software Engineering (2000-4). Presently he is an Associate Editor of the journal Empirical Software Engineering. He also previously worked for a number of years as a software developer for a major bank.

Junyi Shen graduated from the Department of Applied Mathematics at the Xi’an Jiaotong University, China in 1962. He is professor of software technology in the department of computer science & technology at the Xi'an Jiaotong University, Xi'an, China. He has published more than 100 refereed papers and three books in the areas of database, data mining, and software engineering.