A Practical Way to Use Clustering and Context Knowledge for Software Project Planning

Jürgen Münch[a], Jens Heidrich[b], Alexandra Daskovska[b]
[a]*Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany*
[b]*University of Kaiserslautern, Kaiserslautern, Germany*
muench@iese.fhg.de, heidrich@informatik.uni-kl.de, daskovsk@mathematik.uni-kl.de

Abstract

The use of empirical data from past projects for project planning is gaining increasing importance in engineering-style software development. Since software development projects are unique, experience from these projects cannot be reused directly. Nevertheless, common patterns can often be found when comparing past projects. This information can then be used to better support project planning. The article sketches the SPRINT I technique for project planning and controlling. The approach is grounded on the usage and analysis of context-oriented cluster curves. The article focuses on two aspects: How to identify similar projects and build so-called clusters with typical data curves (such as effort distribution); and how to characterize these clusters by using context knowledge. This allows for assigning a new project to a cluster in order to obtain a prediction. Results from an evaluation with data from 25 projects show that the technique provides a practical way to increase the accuracy of software project planning by using empirical data.

1. Introduction

An essential task during software project planning is the identification of target values for key factors (such as effort, cost, schedule). This, for instance, allows for assigning budgets, allocating personnel, and controlling project performance. Appropriate target values can be used to detect abnormal project situations or even tendencies towards unexpected project behavior. This offers opportunities for reacting early and preventing plan deviations. The fundamental project planning activity for getting such target values is estimation. Precise estimates are essential for better project planning and control, and can lead to successful project delivery [2]. Underestimating projects (e.g., expenses, time) can have negative effects on the business reputation and performance of an organization [18]. The consequences of overestimated projects can be poor resource allocation or missed opportunities. Getting accurate estimation models for software development projects is a difficult task. One reason is that software development includes a lot of creative activities, which are typically non-repetitive. Moreover, the underlying processes are not unique and have to be adapted to project constraints and organizational constraints. Thus, estimation models need to be adaptable to specific project and organizational contexts (such as experience of the developers, tool environment, application domain). Early estimation methods emphasized project sizing, cost drivers, or expert judgment [6]. A well-known method of this type, proposed by Boehm [3], is the Constructive Cost Model (COCOMO). It provides equations that incorporate system size as the principal effort driver. Predicted development effort is then adjusted to accommodate the influence of 15 additional cost drivers. Percentage methods could be applied, for instance, to distribute the estimated total effort to phases. These values could be used as target values for software project controlling. Other examples of methods emphasizing project sizing, cost drivers, or expert judgment are SLIM [17], COPMO [6] or the 'reinvention' of COCOMO, namely COCOMO II. Unfortunately, most of these methods are very general. They describe the software process quite broadly.

Practical experience has shown that in many software-developing organizations, data from past projects is available, which could be used for predictions. The collection of this data, for instance, could have been motivated by a CMMI (Capability Maturity Model Integration) improvement effort. This article proposes a technique called SPRINT I, which uses such data from past projects for estimations. Furthermore, the technique gives guidance on how to generalize project-dependent data in order to make them comparable to each other. The technique is based on the cluster analysis introduced by Li and Zelkowitz in 1993 [12], and on the identification of trend changes introduced by Tesoriero and Zelkowitz in 1998 [19]. Cluster analysis is a means for finding groups in data that represent the same information model (e.g., similar effort distributions) [12]. A subsequent analysis of the corresponding context characteristics of the projects in one group supports the assignment of actual projects to similar past projects and helps to predict the future behavior more accurately.

The paper is organized as follows: Section 2 gives a high-level overview of the SPRINT I technique for planning and controlling software development projects. In

Section 3, a technique is shown of how to identify similar projects and build so-called clusters with typical data curves. Section 4 describes how to characterize these clusters by using context knowledge and how to assign actual projects to clusters. Section 5 sketches results from an evaluation of the technique. Section 6 surveys related work with a special focus on clustering approaches. Finally, Section 7 gives an outlook on future research directions in the field.

2. Overview

SPRINT I is a technique for planning and controlling software development projects [15]. The technique can be applied to several attributes of interest (e.g., effort or defect distribution). For clarity, in the following description only one attribute is considered. The technique can be sketched as follows: First, measurement data (i.e., past time series) from former projects are analyzed in order to build up characteristic curves (one for each project), which are comparable to each other. Afterwards, clusters of projects are identified. Based on the context of the project to be planned, the technique selects a suitable cluster and uses its cluster curve (mean of all characteristic curves within a cluster) for prediction. During the enactment of the project, the prediction is adapted with respect to actual project data. This leads to an empirical-based prediction and to flexibility for project and context changes.

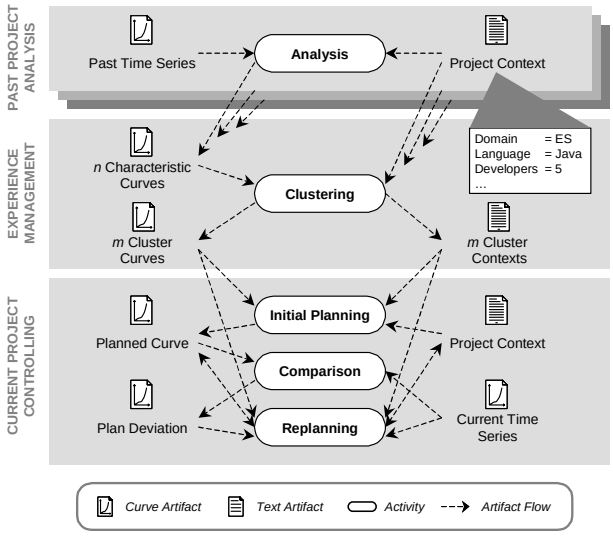

Figure 1. Artifact flow [1].

The technique can basically be separated into five steps (see Figure 1):

Analysis. The first step of the technique analyzes the time-series of the measured attribute per completed project in order to transform it into a so-called characteristic curve. Characteristic curves are comparable curves with respect to a specific attribute for a certain set of past projects. The reason for creating characteristic curves is the following: The original time series represent the actual data measured during the respective projects. Thus, the data from different projects might be on different levels of granularity. If the granularity of a project's time series is too fine, smoothing is required for identifying trends and creating the characteristic curve. If the granularity is too coarse-grained, building the characteristic curve requires some kind of interpolation. Furthermore, an adaptation of the scale unit might be necessary in order to get comparability (e.g., transformation of absolute values into percentage values). Additionally, characteristic curves should exclude one-time interferences that are not relevant for future projects. Several approaches can be used for creating characteristic curves, e. g. [19].

Clustering. For an attribute to be controlled (e.g., accumulated effort) clustering is used to identify groups of characteristic curves that belong together. These are, for instance, curves whose distance (e.g., measured by the Euclidian distance) is less than a certain threshold. All characteristic curves within one cluster are averaged to get a model (the so-called cluster curve), which stands for the whole cluster. This is described in more detail in Section 3. Afterwards, the context of a cluster is determined, which defines the scope of validity of the cluster curve. This is discussed in more detail in Section 4.

Initial Planning. During initial project planning, the relevant target values are estimated on the basis of cluster curves. At the beginning of the project, no actual data are available and therefore, a context characterization of the actual project is used to find a suitable cluster curve for the project attribute under consideration (see Figure 1).

Comparison. During the enactment of the actual project, the current values of an attribute are compared to the predicted values of the cluster curve. If a plan deviation occurs, the predicted values have to be adapted with respect to the new project situation. Therefore, the distance between the two curves has to be computed regularly. If the distance is above (or below) a most tolerable threshold, project management has to be informed in order to initiate dynamic replanning steps, and a new cluster curve has to be sought in order to make a new prediction.

Replanning. In case of significant plan deviation, the causes for the deviation have to be determined. We basically distinguish three different cases: The first one is that the experience we used to build up our prediction model was wrong. The second one is that the characteristics we assumed for our project were wrong (e.g., the experience of the developers was low instead of high). In this case we have to adapt the project context characterization. The third case is that problems occur in the project that lead to a change of the characteristics of the project (e.g., technology changed). In all three cases we can try to identify a new cluster curve within the set of computed clusters.

Basically, there are three ways to choose a suitable cluster for prediction: The first one is matching the contexts of the actual project and the cluster curves (like initial planning). The second possibility is to use the current data of a certain attribute, which has been measured during the enactment of the project up to the present, and match it with the cluster curves in order to find the best cluster curve for prediction. This is a dynamic assignment approach, which incorporates actual project behavior. The third option is to combine the static and dynamic approach to get a hybrid one. If the two possibilities lead to different clusters, a set of exception handling strategies can be applied and reasons be sought. Afterwards, the step comparison is iterated and uses the adapted prediction in order to further control the project.

The *prerequisite* for the application of the technique is that project data from past projects is available for the attribute to be predicted and controlled. Additionally, the context for each of these past projects (i.e., the boundary conditions such as organizational, personal and technical constraints) needs to be known and explicitly characterized.

Keeping these limitations in mind, we see the following *advantages* of the SPRINT I technique: The prediction and control of project progression is directly based on the experience in past projects, the accuracy of planned curves is increased by data- and context-driven selection of cluster curves (that is, the curves of one cluster are selected by the curve itself and context information is assigned to each curve, respectively), the approach is directly applicable without a number of reference applications, and the adaptation of planned curves takes place dynamically.

3. Creation of Software Project Clusters

One prerequisite of the SPRINT I approach is that groups of past projects following a similar behavior pattern can be identified. Clustering is a means to decide which projects can be considered as "similar" in this sense. As a first step, clustering requires the creation of so-called characteristic curves for the projects in order to make them comparable. If clustering is done for predicting work effort, for example, this requires comparable effort data from past projects and a common understanding of a phase model (i.e., a common understanding of activities such as requirement elicitation, design, coding, testing). Table 1 shows the required effort data where X_{ij}, $i = 1,\ldots, L$, $j = 1,\ldots, N$ is the accumulated work effort (in %) per development phase i of the software project j. Each project j is uniquely represented by a vector $X_j = (X_{1j}, \ldots, X_{Lj})$. If we want to control the accumulated project effort, the constraint $X_{1j} \leq \ldots \leq X_{Lj} = 100\%$ has to be valid.

Phases	Project 1	…	Project N
Phase 1	X_{11}	…	X_{1N}
...	...	…	...
Phase L	X_{L1}	…	X_{LN}

Table 1. Required effort data.

Next, for every characteristic curve of a project j, a corresponding context characterization has to be assigned, which represents the project environment that the curve originates from. A context description contains all factors with a proven or assumed impact on the attribute values (such as people factors, technology factors, organizational factors, process factors). This mean that for each project j, $j = 1,\ldots,N$, a vector $U_j = (U_{j1},\ldots, U_{jM})$ exists, where U_{ji} is a value of an context attribute i. In general, context attribute values can be simple strings (like the project's domain), whole numbers (like the number of developers) or real numbers (like a measure for design complexity), or even sets of these (like the application type: Internet, server, XML). For simplicity reasons we concentrate on the case where just one value is assigned to each attribute.

As a first result, we get a characteristic curve for each project j, $j = 1,\ldots, N$, and a corresponding context vector U_j (see Figure 2 and Table 2).

Figure 2. Characteristic curve of a sample project.

Attribute	Value
Team Size	14
Application Type	Internet Service
Programming Language	Fortran
Developer Experience	High
Platform	IBM
Business Area Type	Finance

Table 2. Context vector for a project.

For an attribute to be predicted (e.g., accumulated effort), clustering analysis is used to identify groups of characteristic curves that belong together. These are, for instance, curves whose distance is less than a certain threshold. There exist a number of various distance meas-

ures; however, the most straightforward (and probably the most commonly chosen) way of computing distances between objects in a multi-dimensional space is to use Euclidean distances. This is just the geometric distance in the multidimensional space. Using this measure implies that for every pair of vectors (X_l, X_j), $l, j = 1,\ldots, L$ the Euclidian distance between them is calculated as follows:

$$D_{lj} = Ed(X_l, X_j) = \sqrt{(X_{1l} - X_{1j})^2 + \ldots + (X_{Ll} - X_{Lj})^2}$$

The distances can be greatly influenced by differences in scale among the dimensions from which the distances are computed. For example, if one of the dimensions denotes a measured length in centimeters, and one then converts it to millimeters, the resulting Euclidean or squared Euclidean distances are affected, and thus, the results of cluster analysis may be very different. However, this kind of transformation is not being used in our technique, therefore Euclidian distance, as a measure of closeness, seems to be suitable. There exist other types of distance measures. The ones most often used are Manhattan distance and Chebychev distance.

Table 2 summarizes the Euclidian distances between the characteristic curves of the projects. The diagonal values are *0*, i.e., $D_{ij}=0$ for $i=j$.

Project	1		…		N
1	0	D_{12}	…	$D_{1(N-1)}$	D_{1N}
	D_{21}				
…	…		…		…
					$D_{(N-1)N}$
N	D_{N1}		…	$D_{N(N-1)}$	0

Table 2. Distances between characteristic curves.

Two vectors representing projects j and l are assumed to be similar and are in the same cluster if and only if $D_{lj} < \varepsilon$. By varying ε, we can adjust the size (and, respectively, the number) of clusters by specifying how close two vectors must be in order to fall in the same grouping.

If clustering with respect to several attributes is being made (e.g., for work effort and for reported errors per development phase), we need to perform clustering several times (once for each attribute). The number of clusters as well as the allocation of projects to clusters does not need to be identical in all the cases; as a result, the values of ε can differ (namely, we dynamically change the value of ε according to the number of projects we would like to obtain in one cluster).

Afterwards, the characteristic curves within one cluster are averaged to get a so-called *cluster curve*, which stands for the whole cluster. Thus, as an intermediate result, a set of clusters C_k, $k = 1,\ldots, s$ is obtained, where each cluster consists of a cluster curve and a set of context vectors U_i that were originally attached to the characteristic curves that formed the cluster curve. The question is now: What are typical context characteristics that describe the scope of validity of the cluster curve?

4. Using Context Knowledge to Characterize Project Clusters

After performing clustering, the set of clusters and corresponding cluster curves is obtained. Our main concern is the prediction and control of a new project. This requires the appropriate assignment of the new project to a certain cluster in order to use its cluster curve as prediction model.

At the very beginning of a new project (i.e., in the initial planning phase), a context characterization of the new project is used to identify an appropriate cluster. This requires as a prerequisite that a context characterization for each cluster is available. In the following, we describe how to come up with a context characterization for a cluster:

For instance, let attribute P_j = *"Programming language"* be a context attribute of the context vector. The possible values of this context attribute for a specific cluster can be:

$\{U_1^{(j} =$ *Fortran,* $U_2^{(j)} =$ *Java,* $U_3^{(j)} =$ *Pascal,* $U_4^{(j)} =$ *Java,* $U_5^{(j)} =$ *C++,* $U_6^{(j)} =$ *Other}*

In general, for each context attribute P_j, $j = 1,\ldots,M$, we have one table:

Attribute	Project 1	…	Project N
P_j	$U_i^{(j)}$	…	$U_i^{(j)}$

Table 3. Values of an attribute P_j for each of N projects

where $\{U_1^{(j} ,\ldots, U_n^{(j)}\}$ are the values that an attribute P_j can have.

Now, the question is: What is the typical value of this property for each cluster? This means: We have to find a value (or values) for each attribute P_j, $j = 1,\ldots,M$, which can be regarded as typical for this attribute in the cluster.

This can be done with the help of weights – the measure of how typical a certain value is for a context attribute. In other words, to each value of an attribute a number from the interval [-1; 1] (called "weight") is assigned. The larger the weight is, the more typical the corresponding value is considered to be.

Calculate the weight vector

$$\bar{w}^{(j)} = (w_1^{(j)}, \ldots, w_n^{(j)}), \text{ where } \bar{w}^{(j)} = \frac{\bar{v}^{(j)}}{\| \bar{v}^{(j)} \|},$$

$\|\ldots\|$ is a standard norm in $\mathbf{R}^n$ – space; that is,

$$\| \overline{v}^{(j)} \| = \sqrt{(v_1^{(j)})^2 + ... + (v_n^{(j)})^2}$$

for $\overline{v}^{(j)} = (v_1^{(j)}, ..., v_n^{(j)})$;

$$v_i^{(j)} = \frac{-\left(P_{db}(U_i^{(j)}) - P_{cl}(U_i^{(j)})\right)}{P_{db}(U_i^{(j)})}$$

(this means the ratio of presence of $U_i^{(j)}$ in the cluster related to the presence of $U_i^{(j)}$ in the database).

$P_{cl}(U_i^{(j)})$, $i = 1,...,n$ is the frequency of meeting $U_i^{(j)}$ (in %) in the cluster,

$P_{db}(U_i^{(j)})$, $i = 1,...,n$ is the frequency of meeting $U_i^{(j)}$ (in %) in the whole database.

The weight $w_i^{(j)}$ measures how significant the value for a cluster is. To understand the meaning of the weight, consider a simple example:

Suppose that we have a database of 100 software projects, 90 of which are written in *C++*, and 10 in *Java*. Moreover, consider a cluster consisting of 10 projects, 7 of which are written in *C++*, and 3 are written in *Java*. So, we will have the following vectors:
$(P_{cl}(U_1^{(j)}), P_{cl}(U_2^{(j)})) = (70\%, 30\%)$,
$(P_{db}(U_1^{(j)}), P_{db}(U_2^{(j)})) = (90\%, 10\%)$,
and $w_1^{(j)} = -0.11$, $w_2^{(j)} = 0.98$.

At first sight, the significant number of projects in the cluster is written in *C++*. But if we take a look at the distribution of values in the database, we will notice that 90% are written in *C++*, so it is normal to assume that one will have a similar distribution in the cluster. At the same time, the fact that the percentage of projects written in *Java* in the cluster is 3 times bigger than in the database indicates that it would be reasonable to consider *Java* as a characteristic value for the attribute *"Programming Language"*.

Of course, we are not going to neglect the fact that 70% of projects in the cluster are written in *C++*; and *C++* may as well be included in the list of typical values for an attribute *"Programming Language"* (of course, only as a "2nd typical value" for an attribute). The decision on how many typical values to choose will depend on particular situations and on one's specific needs.

The larger $w_i^{(j)}$ is, the bigger is the probability that we can take $U_i^{(j)}$ as a typical value for the property P_j. So, for each attribute P_j, $j = 1,..., M$ a set of couples $(U_1^{(j)}, w_1^{(j)}), ..., (U_n^{(j)}, w_n^{(j)})$ such that $w_1^{(j)} \geq ... \geq w_n^{(j)}$ will be obtained.

In our example, for attribute P_j = *"Programming language,"* write out:

$$\begin{pmatrix} P_{cl}(U_1) \\ . \\ . \\ . \\ P_{cl}(U_6) \end{pmatrix} = \begin{pmatrix} 42.9\% \\ 14.3\% \\ 42.9\% \\ 0\% \\ 0\% \\ 0\% \end{pmatrix} \text{ and } \begin{pmatrix} P_{db}(U_1) \\ . \\ . \\ . \\ P_{db}(U_6) \end{pmatrix} = \begin{pmatrix} 35.3\% \\ 5.9\% \\ 41.2\% \\ 5.9\% \\ 5.9\% \\ 5.9\% \end{pmatrix}.$$

Write out the vector $(w_1, ..., w_6) = (0.627, 0.098, 0.018, -0.446, -0.446, -.446)$.

So, with attribute P_j = *"Programming Language"* the following couples can be associated:

((Fortran, 0.627), (Java, 0.098), (Pascal, 0.018), (Java, -0.446), (C++, -0.446), (Other, -0.446)).

It seems to be reasonable to pick as a characteristic value $U_i^{(j)}$ such that $w_i^{(j)}$ is maximal. But in practice the values of $w_i^{(j)}$ differ insignificantly and we have to consider all weights within a certain distance. Thus we fix an admissible ε, so that if $w_i^{(j)} > \varepsilon$, $U_i^{(j)}$ can be regarded as a typical value for a property P_j.

For example, if we have the following couples:
$(U_1^{(j)}, 0.059), (U_2^{(j)}, 0.057), (U_3^{(j)}, -0.001), (U_4^{(j)}, -.035)$,
we can agree to take $\varepsilon = 0$, so that $U_1^{(j)}, U_2^{(j)}$ will be considered as characteristic values. If $w_i^{(j)} < 0$ for all $i = 1,..., n$, we can take $U_1^{(j)}$ $(w_1^{(j)} \geq ... \geq w_n^{(j)})$ as a characteristic value.

But, of course, we can vary ε according to our needs; in particular, we can choose ε with respect to the number of values we would like to consider for a certain property. The smaller we choose ε, the larger the quantity of values we have to consider.

It should be mentioned that we have to choose the same ε for all the properties. So, for each attribute P_j, $j = 1,..., M$ set $U_i^{(j)}$ as a characteristic value if and only if $w_i^{(j)} \geq 0$. If $w_i^{(j)} < 0$ for any i, then set $U_1^{(j)}$ as a characteristic value for an attribute P_j. $(U_i^{(j)}, w_i^{(j)})$ is called characteristic couple if $U_i^{(j)}$ is a characteristic value for attribute P_j. Finally, for each attribute P_j, $j = 1,...,M$, a characteristic value $U_i^{(j)}$ is defined.

For each cluster C_k, $k = 1,..., s$ assign the context vector $\underline{U}_k = (\underline{U}_k^1, ..., \underline{U}_k^M)$, where $\underline{U}_k^j$, $j = 1,..., M$ is a set of characteristic couples for an attribute P_j.

As an example, fix $\varepsilon = 0$, then *Fortran*, *Java* and *Pascal* can be regarded as characteristic values for the attribute *"Programming Language"*.

After repeating the above mentioned procedure for finding characteristic values for each attribute P_j we will get the following result (see Table 4):

During the initial planning of a new project, the context characterization must be used in order to find a suitable cluster curve for the project attribute under consideration (see Figure 1). The rest of this section describes how to find a suitable cluster for a new project (see Figure 3).

Attribute	Cluster 1	
Team Size	21	$\underline{U}_1^1$
Application Type	(Wireless Service, 0.059)	$\underline{U}_1^2$
Programming Language	(Fortran, 0.627) (Java, 0.098) (Pascal, 0.018)	$\underline{U}_1^3$
Developer Experience	(High, 0.06)	$\underline{U}_1^4$
Platform	(IBM, 0.018)	$\underline{U}_1^5$
Business Area Type	(Management, 0.06) (Finance, 0.018)	$\underline{U}_1^6$

Table 4. Context vector for a cluster.

Assume that a new project S_*, represented by a context vector $\underline{U}_* = (U_*^1, \ldots, U_*^M)$, where U_*^j, $j = 1,\ldots, M$ is the value of an attribute P_j, is given (see Table 5). The goal is to identify to which cluster the project S_* belongs. After assignment of the project to a certain cluster, the cluster curve will be associated with the project and used for future prediction and controlling.

To establish how "good" the new project will fit for each cluster C_k, $k = 1,\ldots, s$ calculate the function of "goodness of fitting of project S_* to cluster C_k.":

$$G_{S*}(C_k) = \sum_{j=1}^{M} P_k(U_*^j) \text{ , where}$$

$$P_k(U_*^j) = \begin{cases} w_i^{(j)} & \text{, if exists } U_i^{(j)} \in \underline{U}_k^j \text{ s.th. } U_*^j = U_i^{(j)} \\ 0 \text{ , else} & \end{cases}$$

In other words, for a cluster C_k we compare the values U_*^j of the new project with all characteristic values from a set of characteristic couples $\underline{U}_k^j$ (see Figure 3). This is done for each attribute P_j. As a result, the function simply adds the weights of all those characteristic values, which appears to be equal to U_*^j.

Find $\max_{k=1,\ldots s} \{G_{S*}(C_k)\} = G_{S*}(C_i)$

It is assumed that the new project S_* will belong to the cluster C_k, where the function of "goodness of fitting of project S_* to cluster C_k" takes the maximal value.

After the function of "goodness of fitting of project S_* to cluster C_k" is calculated for each cluster C_k, $k = 1,\ldots, s$, one can tell to which cluster the project will belong and, as a result, which pattern (determined by cluster curve) the behavior of the new project will probably follow. However, not only the cluster with maximal value of function should be considered. However, clusters with high values (although not maximal) of the function should not be excluded from consideration.

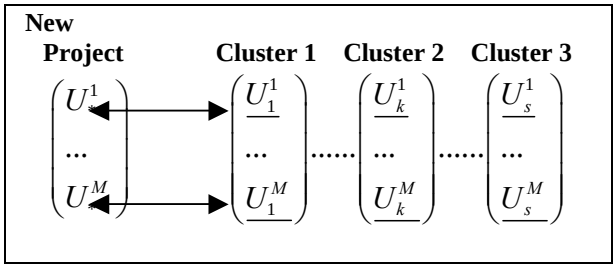

Figure 3. Comparison of contexts.

For instance, take the test project (Table 5) and calculate $G_P(C_i)$ for $i = 1,\ldots, 5$. The values of the function are as follows: $G_P(C_1) = 0.44$; $G_P(C2) = 0.76$; $G_P(C3) = 1.94$; $G_P(C_4) = 1.78$; $G_P(C5) = 1.77$.

This means that the test project belongs to cluster number *3*. Nevertheless, clusters number *4* and *5* should be considered as possible candidates, since the values of $G_P(C_3)$ and $G_P(C_4)$ do not differ a lot from value $G_P(C_3) = 1.94$.

Attribute	Value
Team Size	18
Application Type	Wireless Service
Programming Language	JAVA
Developer Experience	High
Platform	IBM
Business Area Type	Finance

Table 5. Context vector for a test project.

Therefore, it may already happen in the early phase of project development that the project will follow the pattern of cluster *4* or *5* instead of cluster *3*.

Generally speaking, at the very beginning we can assign the new project to a certain cluster by using only its context characterization (since at the very beginning no actual data of the time series of a project is available). After the project has been initiated, not only context-, but also distance-allocation can be used for identifying the closest cluster. If the characteristic curve of the new project (calculated to the actual point in time) deviates significantly from the corresponding cluster curve at some moment, the project may be reassigned to another cluster. That is, context allocation itself is quite a reliable means for prediction; however, combination with distance allocation allows us to make much more precise forecasts.

5. Evaluation

The technique has been initially evaluated using real work effort data from 25 software development projects. Clustering has been performed on 17 randomly selected projects using 10 context parameters. 5 clusters could be identified. The determination of characteristic cluster con-

texts was based on majority decision and averaging (in case of numerical context characteristics). 4 projects were used for testing the SPRINT I technique; these projects were not used for clustering. According to the SPRINT I technique, the 4 test projects were assigned to corresponding clusters (i.e., with each test project, a cluster curve was associated). This has been done using *only* the context characterization of these projects (it was assumed that the distribution of work effort is unknown). Figure 5 illustrates the results obtained; here one can see whether the actual data of the test project is similar to the cluster curve that is recommended for prediction by the technique. The application of the technique on the 4 test projects showed that the context-based selection of cluster curves led to curves with small average deviations between predicted and actual values.

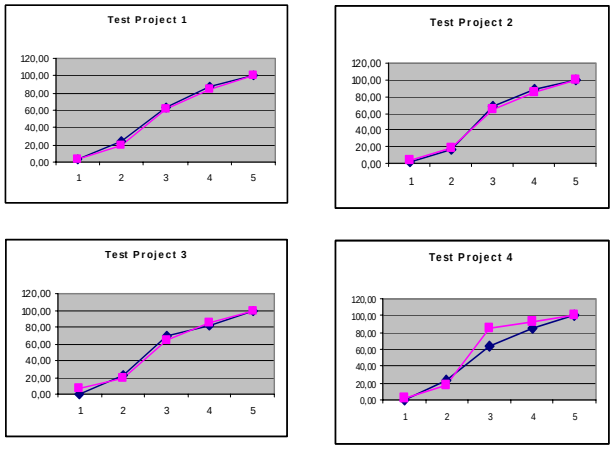

Figure 5. Actual data (gray line) and corresponding cluster curves (black line).

After context assignment of each one of the 4 test projects to a cluster, the actual data and the cluster curve of corresponding clusters were compared to perform the evaluation (i.e., the test projects were assigned to a certain cluster by using a distance measure). Afterwards, we compared whether each project belongs to the same cluster according to both context assignment and distance assignment. For 3 of the 4 test projects, the results obtained after the context assignment were equivalent to the results obtained after distance assignment. One project (i.e., test project 4) was allocated to different clusters after performing both types of allocation; the distance between actual data of the project and cluster curve exceeded the tolerance limit (in our case $\varepsilon = 20$) of two curves too much to be considered similar. Possible causes for this deviation could be: (1) the assignment was based only on the context characterization, but not on actual data; (2) the quantity of data was small (just 17 projects); (3) the characterization of the context could have been done better; (4) the averaging of numeric data was done rather roughly. Another possible reason could be the type of distance measure (namely, Euclidian distance) chosen by us for comparing actual data with assigned cluster curves; however, different types of distance measures were applied, and the Euclidian distance showed the best results (by means of compliance with the context-oriented assignment). We expect that the promising results can be further improved if one adds the actual data for determining the prediction curve.

6. Related Work

For over 15 years the software engineering community has been studying various models for predicting software development projects. In the 1980s, *parametric models* were widely used. Some of the main conclusions were that these models perform poorly when applied in an uncalibrated manner to other environments [6][9]. In the 1990s, *nonparametric modeling techniques* based on the machine learning algorithms, such as optimized set reduction [1], artificial neural networks [8], CART regression trees [22] and analogy-based estimation [20], appeared. In general, these methods make weak assumptions about the data, and some of them create models that can easily be interpreted and can accommodate complex relationships and interactions between cost drivers.

Clustering techniques have already been used for predicting software development projects [12]. Clustering techniques themselves can be roughly classified as follows [7]: *Hierarchical techniques*. The clusters themselves are classified into groups, the process being repeated at different stages to form a tree: nearest neighbor method, farthest neighbor method, centroid cluster analysis [21], and the automatic interaction detector method [13]. The main idea of hierarchical techniques is trying to find the most efficient step, in some defined sense, at each level of subdivision (divisive techniques) or synthesis (agglomerating techniques) of the population. So, a major problem for the investigator is to decide at which stage of the analysis he wishes to stop (i.e., to choose the optimal number of clusters). A general disadvantage is that these techniques do not allow for reallocation of entities that may have been poorly classified at an early stage of analysis (i.e., there is no possibility of correcting a poor initial partition). *Optimization-partitioning techniques*. Clusters are formed by optimization of a so-called “clustering criterion”. The clusters are pair-wise exclusive, as a result forming a partition of the set of entities [10]. The difference to hierarchical techniques is that the optimization partitioning techniques allow the reallocation of the entities, thus permitting poor initial partitions to be corrected at a later stage. Most of the methods also assume that the number of groups is determined *a priori* by the researcher. But there

exist other methods that do allow for changing the number during the course of analysis. One of the major problems is choosing an appropriate clustering criterion, because the use of a particular criterion implies various assumptions about the structure of the data. *Density or mode-seeking techniques*. Here, clusters are formed by looking for regions that contain a relatively dense concentration of entities: the Taxmap method [4], the Cartet count method, and mode analysis [23]. This technique tends to include entities into existing clusters rather than to initiate new clusters. Density or mode-seeking techniques suffer from the problem of sub-optimal solutions, since there might be more than one solution. *Clumping techniques*. Here, the clusters or clumps can overlap [16]. These techniques can be used in some cases (e.g., language studies), where an overlap between classes is allowed (e.g., words tend to have several meanings). *Others.* There exist some methods that do not fall clearly into any of the four previous groups: the Q-Factor analysis [5], and the latent structure analysis [11].

The SPRINT approach uses an optimization partitioning technique for clustering, namely *K-Means*. This method produces disjoint clusters and allows for restructuring of clusters due to experience evolution. This accommodates the typical changes of project contexts over time and supports continual software improvement paradigms such as QIP (Quality Improvement Paradigm). Different optimization partitioning techniques were tried out on real data, among which the K-Means method showed the best results. Nevertheless, using K-Means is just a proposal; an empirical evaluation of several techniques needs to be done in the future.

7. Future Work

The development of the technique led to several open research questions, such as: How to dynamically assign a cluster to an actual project based on both context characteristics and actual project data? How to maintain cluster sets (e.g., performing corrections, adding curves)? Which similarity measures are appropriate for comparing context information? How to build clusters based on both quantitative data and context knowledge? What are factors that influence project performance? The latter can be answered with empirical software engineering methods. The use of dynamic simulation modeling techniques (such as System Dynamics) for prediction could be a good complementation of the technique in case that not enough empirical data is available for clustering. Automation of the SPRINT I technique can be improved, e.g., by integrating it into a so-called *software project control center (SPCC)* [2]. Additionally, further evaluation of the technique is necessary and the practical feasibility has to be discussed.

Acknowledgements. We would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of the article. This work was partly funded by the Federal Ministry of Education and Research (BMBF) as part of the project SEV and the Deutsche Forschungsgemeinschaft as part of the Special Research Project SFB 501.

References

[1] L. C. Briand, V. R. Basili, W. M. Thomas. A Pattern Recognition Approach for Software Engineering Data Analysis. *IEEE Transactions on Software Engineering,* 18(11), pp. 931-942, November 1992.

[2] L. C. Briand, I. Wieczorek. Resource Estimation in Software Engineering. *Encyclopedia of Software Engineering*, pp. 1160-1194, 2001.

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

[4] J. W. Carmichael, P. H. A. Sneath. Taxometric Maps. *Systematic Zoology,* 18(4), pp. 402-415, December 1969.

[5] R. B. Cattell. *Factor analysis.* New York, Harper, 1952.

[6] S. D. Conte, H. E. Dunsmore, V. Y. Shen. *Software Engineering Metric and Models*. Benjamin/Cummins, Menlo Park, CA, 1986.

[7] B. Everitt. *Cluster Analysis*. Heinemann Educational Books Ltd, 1974.

[8] G. R. Finnie, G. E. Wittig, J. M. Desharnais. A Comparison of Software Effort Estimation Techniques: Using Funstion Points with Neural Networks, Case Based Reasoning and Regression Models. *Journal of Systems and Software,* 39(3), pp. 281-289, December 1997.

[9] C. F. Kemerer. An Empirical Validation of Software Cost Estimation Models. *Communications of the ACM*, 30(5), pp. 417-429, May 1987.

[10] R. W. Kennard, L. A. Stone. Computer Aided Design of Experiments. *Technometrics*, 1l, pp. 137-148, 1969.

[11] P. F. Lazarsfeld, N. W. Henry. *Latent Structure Analysis.* New York, Houghton Mifflin, 1968.

[12] N. R. Li, M. V. Zelkowitz. An Information Model for Use in Software Management Estimation and Prediction. In *Proceedings of the 2nd International Conference on Information and Knowledge Management*, pp. 481-489, 1993.

[13] J. N. Morgan, J. A. Sonquist. Problems in the Analysis of Survey Data, and a Proposal. *Journal of the American Statistical Association*, 58, pp. 415-434, 1963.

[14] J. Münch, J. Heidrich. Software Project Control Centers: Concepts and Approaches. Accepted for publication in *Int. Journal of Systems and Software.*

[15] J. Münch, J. Heidrich. Using Cluster Curves to Control Software Development Projects. Short Paper, research demonstration and poster. In *Proc. of the 1st International Symposium on Empirical Software Engineering (ISESE 2002),* volume II, Nara, Japan, October 3-4, 2002.

[16] A. F. Parker-Rhodes, D. Jackson. Current Approaches to Classification and Clump-finding at Cambridge Language Research Unit. *Computer J.,* 10(1), pp. 29-37, May 1967.

[17] L. H. Putnam. A General Empirical Solution to the Macro Software Sizing and Estimating Problem. *IEEE Transactions on Software Engineering*, 4(4), pp. 345-361, 1978.

[18] L. H. Putnam, W. Myers. *Measures for Excellence, Reliable Software on Time, Within Budget*. Yourdon Press, Englewood Cliffs, NJ, 1992.

[19] R. Tesoriero, M. V. Zelkowitz. A Model of Noisy Software Engineering Data (Status Report). In *International Conference on Software Engineering*, pp. 461-464, 1998.

[20] M. Shepperd, C. Schofield. Estimating Software Project Effort Using Analogies. *IEEE Transactions on Software Engineering,* 23(12), pp. 736-743, 1997.

[21] R. R. Sokal, C. D. Michener. A Statistical Method for Evaluating Systematic Relationships. *University of Kansas Science Bulletin*, 38, pp. 1409–1438, 1958.

[22] K. Srinivasan, D. Fisher. Machine Learning Approaches to Estimating Software Development Effort. *IEEE Transactions on Software Engineering,* 21(2), 1995.

[23] D. Wishart. Mode Analysis: A Generalisation of Nearest Neighbour Which Reduces Chaining Effects. In *Numerical Taxonomy*, ed. A.J. Cole, London, Academic Press, 1969.