USE OF CLUSTER ANALYSIS TO EVALUATE SOFTWARE ENGINEERING METHODOLOGIES*

Eric Chen and Marvin V. Zelkowitz
Department of Computer Science
University of Maryland
College Park, Maryland 20742

ABSTRACT

The development of quantitative measures to evaluate software development techniques is necessary if we are going to develop appropriate methodologies for software production. Data is collected by the Software Engineering Laboratory at NASA Goddard Space Flight Center on developing medium scale projects of up to ten man years effort. In this study, cluster analysis was used on this collected data and several measures are proposed. These measurements are objective, quantifiable, are the results of the methodology, and most important, seem relevant.

Introduction

Along with the development of numerous methodologies to aid in software development (e. g., structured programming, chief programmers, walkthroughs, code reading, etc.) is a growing awareness of a need to collect data to be able to quantify the effects of each new technique. Since this data is often collected after the fact, and is therefore often unobtainable and imprecise, at best, it can only be used to indicate possible trends and not specific effects of a given technique.

About five years ago, it was realized that data had to be collected as a project developed in order to better quantify a project's life cycle development. Although this imposed an additional burden on the project, it was believed that the cost was justified - both to give management more knowledge and control over the current project, and to allow the data to be analyzed later in order to determine the impact of the new techniques. Most of the recent work in this area has centered on what to collect - both in deciding what data is needed, and in overseeing the collection process to make sure that the data is collected (both manually and automatically) accurately.

This paper describes this process and a further development in this data collection trend. Now that sufficient data exists, tests are being developed to check the overall validity and value of the data. For example, if data is collected on two different projects, is there any bias in the way the two data sets are created? Can we apply the same measures on each and compare them? What techniques can be used on entire collections of data? Can we classify a project (or an attribute of the project) via measures defined on the data? These and related questions were behind the current study.

This paper introduces the concepts of cluster analysis, a well known technique in many of the social sciences, into the software development environment [Anderberg]. It shows that cluster analysis seems relevant, and the paper develops several measures that seem applicable in predicting methodologies in this environment. While the measures are based upon the data collected by the Software Engineering Laboratory, they appear to be generally applicable in a variety of settings.

At the NASA Goddard Space Flight Center in Greenbelt, Maryland, the Software Engineering Laboratory was organized in 1976 in conjunction with the University of Maryland and Computer Sciences Corporation. The purpose was to study software development within the NASA environment and develop techniques to improve software production [Basili - Zelkowitz]. Data are being collected from certain projects developed by NASA and are now under study. At present over 12,000 forms have been collected (figure 1) on some 30 projects,

* - Research supported in part by grant NSG-5123 from NASA Goddard Space Flight Center to the University of Maryland. Computer time provided in part by the Computer Science Center of the University of Maryland.

CH1627-9/81/0000/0117$00.75 © 1981 IEEE

Run Analysis Form	1834
Component Status Form	2669
Resource Summary	142
Change Report Form	4047
Component Summary Form	3003
General Project Summary	62
Maintenance Form	33

Figure 1. Forms collected by early 1980.

ranging in size from several thousand to over 100,000 Fortran source lines. Effort for each project varies from a few man months to about 10 man years, and most of the larger projects take about a year to complete. The programs generally provide attitude orbit information for unmanned spacecraft and operate on an IBM 360/95 computer; however, we view them simply as large application programs that include many characteristics of any software package, such as user interfaces, graphics, data base accesses, scientific computations and other characteristics.

Cluster Analysis

The information on software production is collected on a set of forms. Some forms are filled out on a regular basis. For example, the Component Status Report, filled out weekly by each programmer, gives the components of the system worked on that week, hours worked, and phase of development (e. g. design, code, test). With this data, a snapshot of the developing program can be computed, week by week. The Resource Summary gives the total hours spent by all personnel on the project for a given week.

Other forms are filled out when needed. Each computer runs results in an entry in the Computer Run Analysis form giving details of the run. Each change or correction of an error results in a Change Report Form being filled out giving the details of the change, its cause, and its effect. With this data, a complete history of a developing program can be maintained.

As each form is entered in our data base, it becomes a vector of numbers. Thus each project creates a number of data sets, where each data set can be considered a multidimensional vector space with individual forms being points in that space. Obvious questions that arise from this view are which points are near one another, does the location in the space have any meaning, and do clusters of such points have any significance?

In order to answer such questions, cluster analysis is being applied to this data. The rest of this section will describe the assumptions that we have made and the algorithms that we have used for creating clusters. The remaining sections will descibe the various applications that we have applied cluster analysis to.

In order to cluster data, the following algorithm was used:

(1) Let x and y be two points (forms) in one data set and let dxy be the "distance" between points x and y. For this study we used the similarity between two vectors via the cosine function [Salton - Wong] as our distance function since the usual Euclideean distance measures did not seem relevant to components with different characteristics. Let xi and yi be the ith selectors (data values) of vectors (forms) x and y. Then

$$dxy = \frac{\sum(xi\ yi)}{\sqrt{\sum(xi)^2 \sum(yi)^2}}$$

dxy will have some value between 0 (if x and y are dissimilar) and 1 (if x and y are similar).

(2) Choose some threshold value B with B between 0 and 1. We assumed that for dxy<B, x is sufficiently dissimilar to y and therefore x is unrelated to y. If dxy≥B, x and y will be considered to be related. Later we will describe the various ways of choosing B.

(3) Compute the connectivity matrix C such that

$$Cxy = \begin{cases} \underline{true} \text{ if } dxy \geq B \\ \underline{false} \text{ if } dxy < B \end{cases}$$

Cxy=true means that nodes x and y are near one another and are considered to be connected. Since dxy=dyx, C is a symmetric matrix. We have now converted the distance matrix into a graph-structured connectivity matrix. Cxv=true means that there is an arc from node x to node y in some subgraph of all nodes. It is only necessary to compute the total subgraph of connected nodes in order to arrive at the cluster.

(4) The connected subgraph can be computed by computing the transitive closure C* of C:

$$C^* = C^1 + C^2 + C^3 + \ldots + C^n$$

where addition and multiplication refer to the logical operations of "or" and "and", respectively. In this case, C*xy = true if and only if x and y are in the same connected subgraph.

The set of subgraphs forms a disjoint set of clusters. Every point belongs to one and only one (possibly singleton) cluster.

This algorithm was used to cluster based upon subsets of the possible selectors for each vector. Various other criteria (e. g., an additional selector not used in clustering) were used to see if they were predictors of which cluster a given form would reside. If so, then this selector is a dependent variable, and relating back to the original goals of the research, would indicate a relationship between the methodology (as specified in the criteria) and the data collected on the forms.

Development Histories

Current software folklore states that better systems result if a greater emphasis is placed on design. Each such report gives its own correct formula (e. g. 40% design, 20% code, 40% test), but very little quantitative data exists for verifying such relationships. Most studies basically state that "we did it this way and the results look good." As an initial test of cluster analysis we decided to investigate this question. This would also be a relatively easy test on the merits of cluster analysis itself as a valid measuring tool in our environment.

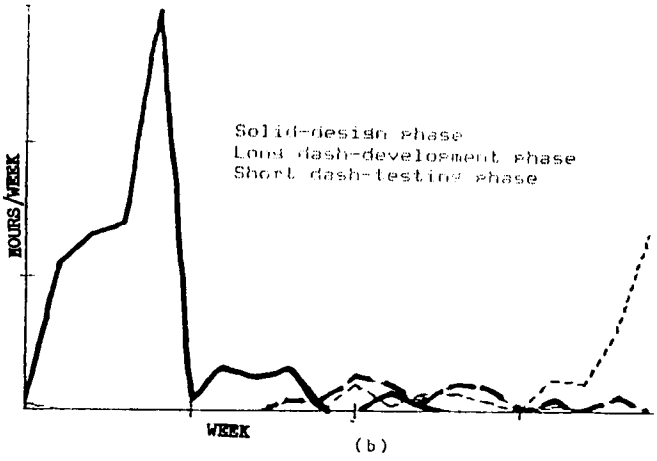
In our data base we collect the number of hours each programmer spends each week on each component. A component roughly translates into a Fortran subroutine . The stage of development worked on (design, code, test) is also indicated (figure 2). (The group that we are monitoring at NASA gets the specifications from another group and a third group takes over the software for its operational lifetime. Thus we are only evaluating the actual development process.) For this reason, the percentages that we develop later in this paper differ from more "classical" life cycle models, since we are mostly ignoring requirements, specification and operational phases.

Due to high computer costs, we limited ourselves to the 50 largest components (out of about 400) per project assuming that most of the effort on a project will be used to build the largest components. These will have a greater influence on the overall methodology than the others. The largest component needed about 400 hours to complete while the smallest of the 50 required about 25 hours.

Assuming a continuous curve, smoothing techniques were applied. In order to compare dissimilar components, the time axis was converted from weeks into deciles and the effort (vertical) axis was converted into per cent of total effort. Thus any two components were comparable (figure 3).

WEEK	DESIGN	CODE	TEST
1	33.0	0.0	0.0
2	40.0	0.0	0.0
3	42.0	0.0	0.0
4	89.0	0.0	0.0
5	1.0	0.0	0.0
6	10.0	0.0	0.0
7	7.0	0.0	0.0
8	10.0	2.0	0.0
9	0.0	2.0	0.0
10	0.0	8.0	6.0
11	4.0	6.0	1.0
12	1.0	2.0	4.0
13	0.0	6.0	4.0
14	0.0	5.0	2.0
15	0.0	0.0	0.0
16	1.0	1.5	7.0
17	0.0	0.0	6.0
18	0.0	3.0	18.0
19	0.0	0.0	40.0
TOTAL	238.0	35.5	88.0

(a)



(b)

Figure 2. Effort (in hours) to develop a typical component (by week)

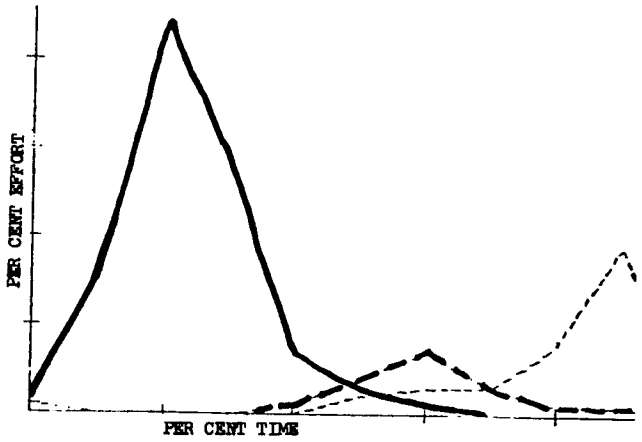


Figure 3. Smoothed and scaled data of figure 2.

Module Modifiability

In order to pick an appropriate B, various values were tested on five projects (figure 4). As B varied between 0 and 1, 4 of the five tested project had similar number of clustered components. Only project [redacted] differed and project [redacted] was the only one of the five that consisted mainly of reused modules from similar previous projects. Thus the number of clusters, relative to B, may be an invariant that can be used to measure the "newness" of the source code. Such a measure can be objectively applied to a given project to determine the degree to which previous source code has been modified for this new project.

We used a B that forced the largest cluster to be under 20 components in size. A smaller B caused many of the clusters to merge into one large cluster while any larger B caused cluster size to drop rapidly.

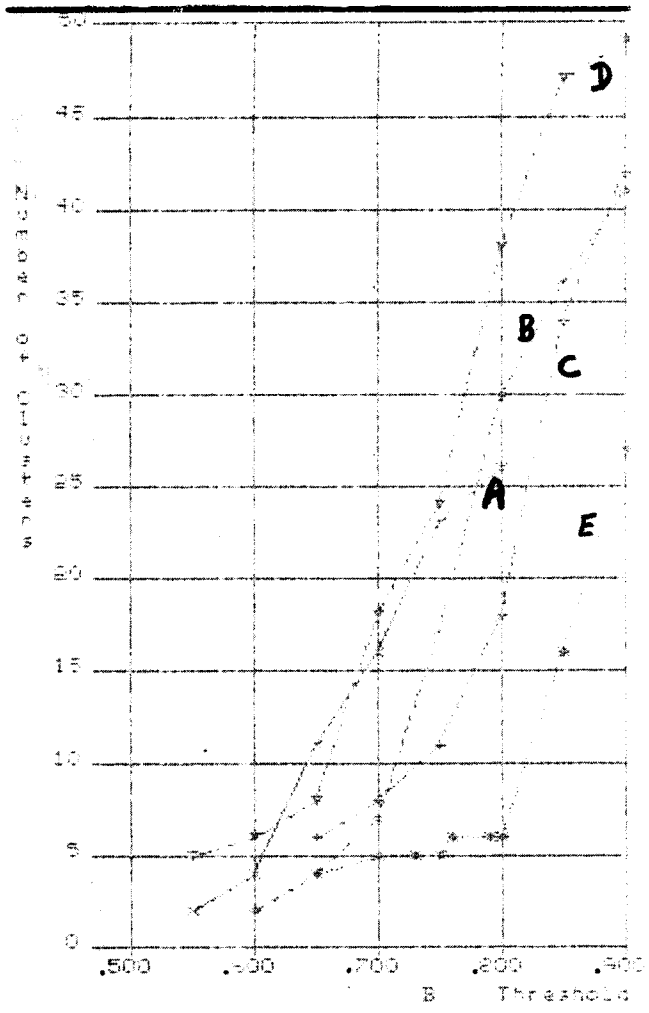


Figure 4. Number of clusters as B varies (for projects A, B, C, D, and E)

Project A

UNCHANGED	CHANGED
6	2
0	5
0	2
0	3
0	10
0	2

Unchanged: 10

Project B

UNCHANGED	CHANGED
2	5
1	5
1	2
1	1
1	1
0	2
0	2
0	4

Unchanged: 10

Project C

UNCHANGED	CHANGED
3	1
2	0
2	0
1	1
0	2
0	2

Unchanged: 32

Project D

UNCHANGED	CHANGED
3	0
2	0
1	1
0	2
0	2
0	2
0	2
0	5

Unchanged: 12

Project E

UNCHANGED	CHANGED
3	1
1	15
0	3
0	3
0	2

Unchanged: 8

Figure 5. Clusters. Each line represents one cluster of changed and unchanged components

Module Correctness

Each component's development history was now reduced to 30 values (3 at each 10 percentile), and these 30 values were used to cluster the 50 components in each project. As an independent variable we considered whether a component had been modified via a change or error. This would be a measure of how effective the process had been. Once unit testing is completed, a component is added to the project's library. If it ever changes after that date (due to further testing), then a change report form is submitted. We simply looked for change report forms that had been filled out for the 50 components under study.

Approximately 80% (about 40) of the 50 components for each project are eventually altered (figure 5). However, in 4 out of 5 projects, all of the unaltered components seem to merge into a few clusters that contain few (if any) altered components. Thus the shape of the development history curve seems to be an indicator of component reliability (as measured by the absense of any changes during testing). The physical significance of each "error free" cluster is now under study.

Projects A B, D E

UNCHANGED	CHANGED
4	1
4	0
2	2
1	9
1	1
1	1
1	6
1	2
0	2
0	2
0	2
0	2
0	2
0	2
0	2
0	2
0	2
0	2
0	2
0	3
0	3
0	3
0	3

Figure 6. Clustering 4 projects

The reliability of this conclusion was enhanced somewhat by merging all four projects and clustering the 200 resulting components. In this case the error free clusters did seem to merge (figure 6).

Phase effort

We were now ready to test one of our original hypotheses - per cent of effort in each phase. Project D had to be eliminated since the data collection began when this project was mostly complete with design.

For project A, the cluster with 6 out of 8 unchanged components turned out to have: Design: 64.1%, Code: 14.4%, and Test: 21.4%, while the five clusters with errors broke down as follows:

DESIGN	CODE	TEST
4.5	53.3	42.1
0	78.7	21.2
7.3	51.7	40.8
0	34.1	65.8
0	35.4	64.5

which clearly shows the value of good design efforts.

The data for projects C and E have similar results:

PROJECT C

UNCHANGED CLUSTERS			CHANGED CLUSTERS		
DES	CODE	TEST	DES	CODE	TEST
83.6	16.3	0	37.3	40.2	22.3
50.9	45.0	3.9	21.4	50.0	28.5
81.7	3.4	14.8			
100.0*	0	0			

***- This shows that while we believe that the data is accurate, some errors must exist.**

PROJECT E

UNCHANGED CLUSTERS			CHANGED CLUSTERS		
DES	CODE	TEST	DES	CODE	TEST
97.5	2.4	0	7.3	83.5	9.0+
			25.2	67.5	7.2
			1.5	82.6	15.8
			24.0	75.9	0

further strengthening this result.

For project B, where clustering was not as effective, the breakdown was as follows. For clusters with at least one unchanged component:

DESIGN	CODE	TEST
22.0	66.2	11.3
27.6	52.8	19.3
26.0	39.7	34.2
94.9	5.0	0

and for clusters with changed components:

DESIGN	CODE	TEST
78.9	15.3	5.6
6.2	66.7	25.9
22.4	24.1	53.4

and do not seem to have much significance. **Project B, interesting enough, was the one project that had the hardest time meeting its objectives.**

In order to put these numbers in perspective, for the NASA environment, the per cent design, code and test effort was computed. If the data is displayed in the conventional manner using official milestone dates for each phase (figure 7a), then design accounts for about one quarter of the effort, and code for about one half. However, if the actual phase effort is computed independent of the date the task is performed, then the percentages change significantly. Design increases about 10% and coding drops about 5%. Thus in the NASA environment, simply using milestone dates results in:

(1) Underestimating the actual design effort, and

(2) Overestimating the actual code effort.

One other aspect of this data can be seen by comparing the per cent of a task that was performed after its official milestone date (or before in the case of testing) (figure 7b). Note that a consistent 23%-25% of the design occurred during the coding phase and up to half of the testing occurred before the official test phase. Since module unit testing was considered to be part of the development phase (for figure 7), this seems significant. In addition, since project B was the most behind schedule, the 38% of design that occurred during coding might indicate a too early design milestone which

+- The cluster with one unchanged and 15 changed components was considered a changed cluster for this chart.

caused other problems later. Thus using milestone dates for phase determination must be viewed with caution.

Error Histories

A second test of cluster analysis was performed by analyzing the change report form, mentioned previously. Unlike the previous study on component development histories, where each data point represented 30 related attributes (per cent effort), the change report form consists of approximately 50 items that seek to identify a source program error, its cause, effects, and effort used find and correct the error. Figure 8 lists the selectors we used to cluster each form.

It was assumed that each set of responses on one form indicated a technique used to debug a system. Therefore the set of forms could be as a measure of the methodology used. It was assumed that different projects using different methodologies would have different clusters of change report forms.

In one run the programmer was the independent variable (i. e., not used in cluster analysis). This was to determine any bias in the way different programmers filled out the forms. However, to the .05 significance level, all programmers were uniformly distributed in all clusters. The conclusion therefore seems to be that in our environment, all programmers are doing essentially the same job. This would indicate that there is no real chief programmer/programmer dichotomy in the tasks we measured. This agrees with the subjective conclusions about these projects.

PROJECT	BY DATE			BY PHASE		
A	22.7	49.6	27.6	30.7	44.7	24.5
B	22.2	68.2	9.5	34.1	45.6	20.2
C	27.4	61.6	11.0	36.8	48.7	14.5
E	30.2	52.3	17.4	42.0	50.4	7.6

(a) Per cent design, code and test by milestone date and actual task

	%DESIGN DURING CODE	%CODE DURING TEST	%TEST DURING DESIGN & CODE
A	23	27	49
B	38	4	67
C	25	8	56
E	25	21	24

(b) Per cent effort during another phase

(Data collection began after the design phase of project D, so it is omitted here.)

Figure 7. Project task breakdown by date and phase

Dates (time error found, fixed)
Type of error
Time to make and fix change
Causes of error
Tools to find error
Was error related to other errors
Time to locate error
Clerical error

Figure 8. Sample data used in change report

Methodology Signatures

The set of clusters for an entire project define the basic methodology for developing a software project. We call this set of clusters its methodology signature. Two similar projects using similar techniques should have similar signatures. That is, they should find each type of error in approximately the same ratio using similar techniques for the discovery.

To test this we clustered the change report forms of several projects, then we combined the forms for two of them and clustered the merged set. Each cluster in the merged set represented two clusters - one from each set. We counted the number of components in each cluster and graphed these (figure 9). Note that large clusters tended to merge with large clusters and small clusters with small clusters. The merged set of clusters had a correlation coefficient of .32 with respect to the clusters that make up the set.

This leads to an interesting methodology measure. First cluster two of the sets of change forms and then look at the clusters formed by clustering the combined set of forms. If they have similar patterns and similar clusters merge together, then they indicate similar development structure and

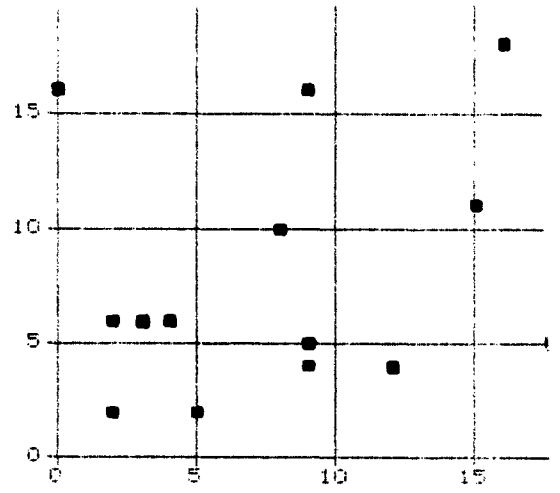


Figure 9. Number of components from project A (horizontal) and project B (vertical) in joint clusters.

probably similar methodologies. If not, then further study is indicated. Either the methodologies differ, or the class of errors found differ for some reason.

An interesting idea (although only speculation at this time) would be to generate a set of benchmark projects each representing a different methodology. An unknown project could then be clustered with each, and the one for which the merged graph generates the highest correlation would represent the unknown methodology. If this turns out to be true, then this technique would represent a quantitative method to measure a software methodology.

Conclusions

Cluster analysis has been applied on data collected by the Software Engineering Laboratory on several projects. The preliminary results should that the technique is effective in determining characteristics about the projects and the underlying methodology used in their development. Several measures seem interesting and are now under study:

(1) The threshold value in determining connectedness of the underlying graph structure (called B in this report) seems to have significance and seems to be a measure of the "reusability" of the existing source code in a new project.

(2) The development history is an indication of probable program reliability.

(3) The methodology signature developed from analyzing the change report forms looks like an effective measure of the techniques used in developing projects.

(4) More complex measures of distance between points are being considered. The current one has the virtue of being easy to program but has the disadvantage that long threadlike "snakes" of points will be in the same cluster, rather than some central "centroid" with only points near than centroid being in the cluster.

The entire software development methodology area is often filled with vague statements, folklore, and lack of quantifiable data. It is hoped that techniques such as described here can be used to give this important topic a more quantifiable, exact and scientific footing.

Acknowledgement

We would like to acknowledge the contribution of Mr. Warren Miller in computing many of the values that are described herein.

References

[Anderberg] Anderberg M. A., Cluster Analysis for applications, Academic Press, New York. 1973.

[Basili - Zelkowitz] Basili V and M Zelkowitz, Resource estimation for medium scale projects, Third International Conference on Software Engineering, Atlanta Georgia, 1978.

[Salton - Wong] Salton G and A Wong, Generation and search of clustered files, ACM Transactions on Database Systems 3, No. 4, 1978, pp. 321-346.