Visualization and Analysis of Software Engineering Data Using Self-Organizing Maps

Stephen G. MacDonell
SERL, School of Computing and Mathematical Sciences, Auckland University of Technology, Private Bag 92006, Auckland 1142, New Zealand
stephen.macdonell@aut.ac.nz

Abstract

There is no question that accuracy is an important requirement of classification and prediction models used in software engineering management. It is, however, just one of a number of attributes that contribute to a model being 'useful'. Understandably much research has been undertaken with the objective of maximizing model accuracy, but this has often occurred with little regard for these other model attributes, which might include cost-effectiveness, credibility and, for want of a better term, meaningfulness. The research described in this paper addresses both model accuracy and meaningfulness as conveyed by self-organizing maps (SOMs). SOMs are neural-network based representations of data distributions that provide two-dimensional depictions of multi-dimensional relationships. As such they can enable developers and project managers (and researchers) to visualize often complex interactions among and between software measurement data. We illustrate the effectiveness of SOMs here by building on two previous empirical studies. Not only are the maps able to portray graphically the distributions of variables and their interrelationships, they also prove to be effective in terms of classification and prediction accuracy. As a result we believe that they could be a useful supplementary tool for researchers and managers concerned with understanding, modeling and controlling complex software projects.

1. INTRODUCTION

Empirical software engineering research has led to the development of a large number of models that have been said to be or shown to be useful in project management. Many have been predictive models, estimating artifact size, development effort and phase/project duration in particular. Others have used classification methods to address issues such as task risk, test coverage and artifact reliability. In many instances the objective of the research has been to arrive at models that are as accurate as possible. This is understandable – while other model attributes are possibly desirable, accuracy is generally considered to be of primary importance in terms of managing projects most effectively. A *singular* focus on accuracy may not be beneficial in a broader sense, however, particularly if organizational learning is considered valuable alongside accuracy. Other attributes then assume some level of importance, attributes such as the transparency of the model-building process, model robustness with respect to inputs, generalizability over novel data, data range coverage, freedom from fixed structure, and model understandability, objectivity, parsimony, and meaningfulness.

Model-building methods vary in the degree to which they facilitate the delivery of these attributes. For instance, algorithmic approaches such as function point analysis (FPA) may result in accurate models for systems of a certain type or from a specific domain but they may not generalize well. Statistical techniques such as regression, and machine learning methods including perceptron-based artificial neural networks, may similarly produce models that are accurate, but they can also be difficult to interpret. In this paper we consider another neural network approach – the Kohonen self-organizing map (SOM) [8] – as a means of building software project management models that are both accurate and meaningful. Whereas to date SOMs have been used largely as classification tools, we here extend this work to perform data prediction.

Next we discuss previous work addressing the visualization of software engineering (SE) data before introducing SOMs as a means of doing so. We then describe two empirical analyses, each employing a data set published previously in the software metrics literature. In each case we compare the outputs of the SOM-based analysis with the more typical statistical approach employed in the previous work. We then conclude the paper with a brief discussion of the outcomes and implications of our research.

2. VISUALIZING SE DATA USING SOM CLUSTERS

A well-constructed image can convey more or different information to that available from simple text or tabular depictions. Prior research investigating the visualization of software engineering (SE) data has tended to focus on this as an almost separate aim to that of achieving accurate predictive or classification models (e.g. see [3]). Several papers, however, present a combined approach to visualization and analysis of data across domains including software engineering. Lee *et al*. [9,10] discuss the use of colour-coded starfield plots (adapted from scatter plots) to consider conversion patterns in website ecommerce metrics. Noble [12,13] presents a screen overlay-based approach to help developers assess the usability of their user interfaces, employing metrics that represent task concordance, layout uniformity, and coherence. Most relevant here is the work of Irwin and Churcher [6] who developed a formally defined grammar for software metrics, with a focus on aspects of object-oriented (OO) structure. One of the benefits of such a grammar is the ability to objectively and automatically parse the constructs, resulting in data that can be visualized in a form configurable by the user. Irwin and Churcher [6] use a 3D virtual representation to illustrate the number and strength of connections among software components.

In research concerned with developing models for classification and prediction, simple visualisations are dominant. The more commonly used methods are scatter and regression plots, histograms and frequency distributions, normal probability plots, and box and whisker plots. Less commonly used depictions include Kiviat diagrams, employed by Fioravanti and Nesi [4] to visualize OO metrics, and input-output dependence graphs (IODGs), used by Kang and Bieman [7] to depict aspects of module cohesion. The latter authors contend that using visual representations of structure enhances decision-making on software reorganization. Visual analysis can in turn be aided by tabular data [1], showing the outcomes of tests of independence, correlation, variable significance and so on.

There is some merit in using simple depictions. Most are easy to build and such an approach ensures that the focus is on the models and their accuracy rather than on how they are depicted. However, the more simple methods by their nature may not be effective in conveying the complex relationships that exist when there are multiple independent variables, which are perhaps in themselves related. In these circumstances model interpretation becomes more difficult.

Kohonen's self-organizing maps (SOMs) were developed to address these sorts of challenges [8]. They offer a means of data clustering and visualization that projects multiple dimensions into 2D (or perhaps 3D) space based on object similarity while retaining the original topology of the data. The structure of a SOM is often characterized as a sheet-like array of cells (or nodes), each of which is tuned according to patterns that occur in the input space. The map learns in an unsupervised manner; the network is not trained (as is the case in backpropagation-based networks) and no 'correct' learning outcomes are specified. Rather, input vectors are compared with those already in the map and clusters are created or augmented to minimize the distance between similar vectors (and conversely to maximize the distance between different vectors). As in other clustering approaches, the measure of distance can be one of many. Because of its simplicity, Euclidian distance is most commonly employed [17].

As an unsupervised method, some user input is required to produce a relevant and appropriately structured map. The user must decide on the number of nodes (or cells) in the map to give the desired level of granularity, and the number of clusters that makes analytical sense. Choice of the number of clusters is in the first instance automated by the implementation of an objective function that maximizes similarity within clusters and distinction between clusters. The user can also specify the learning rate, the extent to which neuron connections are adjusted as clusters are formed. Finally the user must also specify a stopping condition – normally either the number of learning iterations or a learning threshold, reflecting no further significant changes in the distribution of nodes. (As this paper is primarily concerned with the effective *application* of SOMs, the inclusion of formal definitions of the map construction process and associated parameters are not included here. Such definitions may be found in [8]).

The positive characteristics of SOMs that are particularly relevant here are their visual power and their richness of content in illustrating relationships. This is in contrast to the rather sparse output of commonly employed statistical modeling methods (and the black box nature of perceptron-based neural networks). The sometimes complex models built using techniques such as regression, while accurate, can also be cumbersome in structure, particularly if they include intercorrelated factors, interaction terms, dummy variables, and so on. This makes them challenging to interpret and to act upon in order to improve practice. Correlation statistics are undoubtedly useful but for reasons of understandability we tend to use bivariate analyses – comparing one variable with another in isolation of other effects. Important multivariate relationships can therefore be overlooked. Standard classification methods can also produce apparently arbitrary outcomes especially for artifacts that fall close to fixed class boundaries. Sound use of SOMs can enable more graduated boundaries to be used in classification, while taking into account highly non-linear relationships among multiple variables.

To date SOMs have not been used to any significant extent in empirical software engineering. Hong and Wu [5] assessed the efficacy of SOM-based prediction using simulated data, but found them to be less useful than multilayer perceptron neural networks for their purposes. More recently, Zhong *et al.* [21] reported encouraging results from their use of a self-organizing method similar to that used in SOMs (NeuralGas). The most active proponent of SOMs in this domain, however, is Pedrycz, who highlighted their potential in 1998 [16] and who illustrated their use with SE data in several subsequent papers [e.g. 17-19]. He has also provided support more generally for the use of other self-organizing approaches but with a greater focus on accuracy as opposed to visualization [14,15]. We here augment the work of Pedrycz and others by:

(i) comparing the effectiveness of SOMs against more traditional statistical analyses of two publicly available data sets; and
(ii) using a SOM to perform predictive modelling over a hold-out sample so that its effectiveness can be assessed in an unbiased manner.

There is a risk that this is a(nother) 'solution looking for a problem' – this claim would be valid if no gains were made. That said, we need to empirically assess the effectiveness of any method if we are to have a sound basis for using *or* discarding it. The next sections of the paper therefore assess the effectiveness of SOMS in two specific sets of tasks: classification and model fitting, and hold-out sample based prediction. We leave utility in practice to a later study.

3. SOMS IN CLASSIFICATION AND MODEL FITTING

The extent to which a software artifact falls into a particular category of interest can be assessed using metrics, the term commonly used to denote measured characteristics of the object under scrutiny. Assuming that the chosen metric constructs are valid and useful – that is, they measure what they purport to measure and the resulting data is of some practical use in software management – then evaluation of metric values enables us to classify artifacts as being, for instance, large or small, high-risk or low-risk, and so on.

To assess how well SOMs classify software artifacts we based this part of our work on a study reported by Cartwright and Shepperd [2]. They analyzed "…an industrial object-oriented (OO) system comprised of 133,000 lines of C++. The system was a subsystem of a telecommunications product and was developed using the Shlaer-Mellor method." [2, p.786]. Their study evaluated the relationships between nine class-based measures and two external variables: class size (in C++ lines of code (LOC)) and the count of defects per class, for the 32 classes in the system. Two of the tables from [2] are aggregated here as Table 1, listing the variables collected and the summary statistics for those variables.

As in [2] we first addressed the task of defect analysis. To illustrate the visualization capabilities of SOMs we constructed a map using the entire data set. Figure 1, built using 500 cells, depicts the clustering of the data. We used seven clusters – areas representing similar vectors in the source data set. This gave us sufficient differentiation without being too fine-grained to be impractical. In particular, aggregated groupings of classes with Very low/Low, Medium and Very high defect counts are evident. These clusters can be interrogated to reveal descriptive statistics for each, or to identify specific system classes.

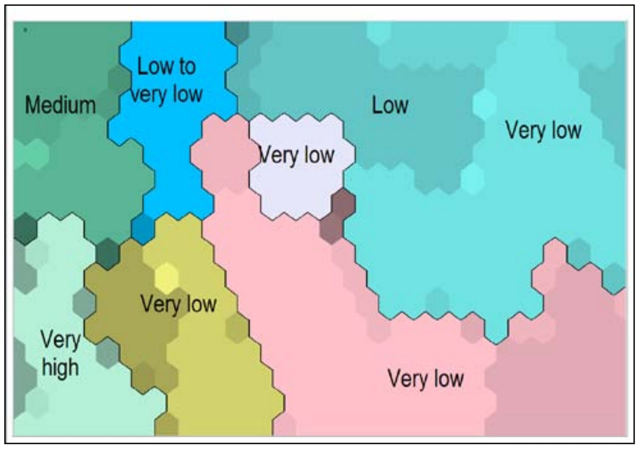

Figure 1. Overall cluster map for defect classification data

Perhaps more informative are the component maps (Figure 2) that depict the distribution of vectors for one variable per map. Each depicts the local average value at each cell in a particular colour, with the scale below showing the mapping between colour and value. This depiction enables the user to consider relationships *among* 'predictor' variables as well as between these and a dependent variable. In this case we can see that for several variables (ATTRIB, EVN, READ, STATE, WRITE, RWD) the higher value artifacts are clustered together in the bottom left-hand corner of the map. In contrast, the grouping based on inheritance tree depth (DIT) is quite different, suggesting that it represents a separate dimension to that embodied in other variables.

We now turn our attention to building a classification model for defects. In order to maximize comparability of our results with those of [2] we built models using the same (small sets of) variables as they did. In the original study the count of defects per class (DEFEC) was modeled by a linear regression equation comprising a constant term, the count of events per class in the state model (EVN) and a dummy variable (Inherits), to reflect whether class inheritance had an impact on the number of defects found. A map comprising seven clusters was generated from the data set (Figure 3). The labels are annotations that we have added to characterize the defect count in each cluster.

Table 1. Classification task – variables collected and summary statistics (adapted from [2])

Label	Variable	Explanation	Mean	Median	Min	Max
ATTRIB	Attributes	Count of attributes per class from the information model	8.66	4.5	1	32
STATE	States	Count of states per class in the state model	18.03	13	0	114
EVN	Events	Count of events per class in the state model	20.53	10.5	0	122
READ	Reads	Count of all read accesses by a class contained in the CASE tool	16.25	11.5	0	83
WRITE	Writes	Count of all write accesses by a class contained in the CASE tool	14.22	8.5	0	56
DEL	Deletes	Count of all delete accesses by a class contained in the CASE tool	1.50	1	0	5
RWD	Read/ write/ delete	Count of all synchronous accesses (the sum of READS, WRITES and DELS) per class contained in the CASE tool	31.97	22	0	131
DIT	Depth of Inheritance Tree	Depth of a class in the inheritance tree where the root class is zero	0.44	0	0	2
NOC	Number of children	Number of child classes	0.31	0	0	4
DEFEC	Defects	Count of defects per class	8.09	2	0	47
LOC	Lines of Code	C++ lines of code per class	4178.50	3524.5	603	20165

Figure 2. Component maps for defect classification data

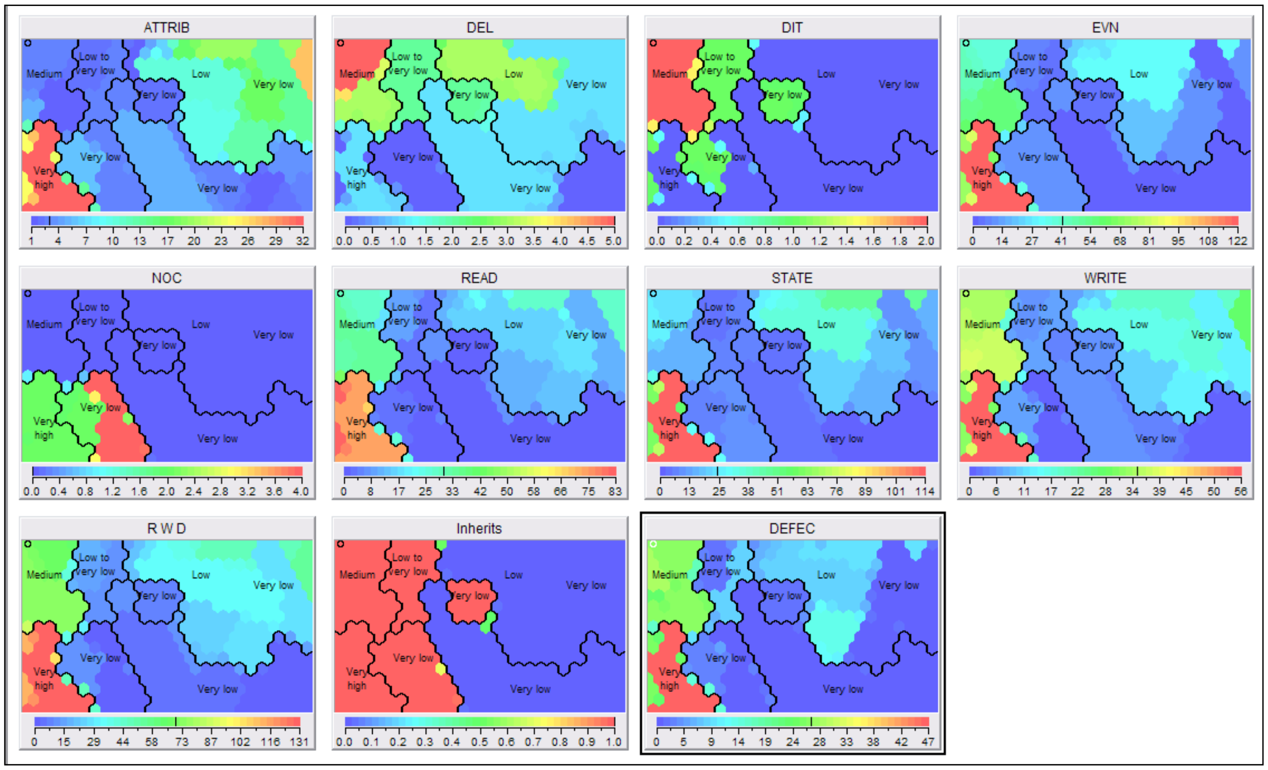

Figure 2. Component maps for defect classification data

While the clusters *appear* to be distinct (Figure 3) this is somewhat artificial, as their associated values overlap. However the clustering does enable us to get a sense of the distribution of multidimensional data projected onto a more easily viewed 2D space. Greater clarity on the data distributions and the relationships among the variables can be gained from viewing the component maps, shown in Figure 4. In this figure we can see the impact of the Inherits dummy variable on the clustering of classes. Furthermore, it becomes visually evident that (i) most classes have few events; (ii) as the number of events increases so does defect count (the shading (colour) moving from dark (blue) – low – through light (yellow/green) to dark again (red) –

high); and (iii) the degree of inheritance does have an additional impact on defect count per class.

While the clusters *appear* to be distinct (Figure 3) this is somewhat artificial, as their associated values overlap. However the clustering does enable us to get a sense of the distribution of multidimensional data projected onto a more easily viewed 2D space. Greater clarity on the data distributions and the relationships among the variables can be gained from viewing the component maps, shown in Figure 4. In this figure we can see the impact of the Inherits dummy variable on the clustering of classes. Furthermore, it becomes visually evident that (i) most classes have few events; (ii) as the number of events increases so does defect count (the shading (colour) moving from dark (blue) – low – through light (yellow/green) to dark again (red) – high); and (iii) the degree of inheritance does have an additional impact on defect count per class.

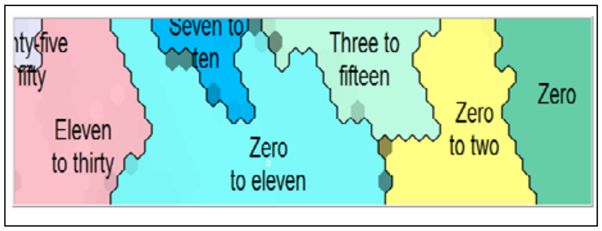

Figure 3. Clusters of DEFECs based on EVN and Inherits variables

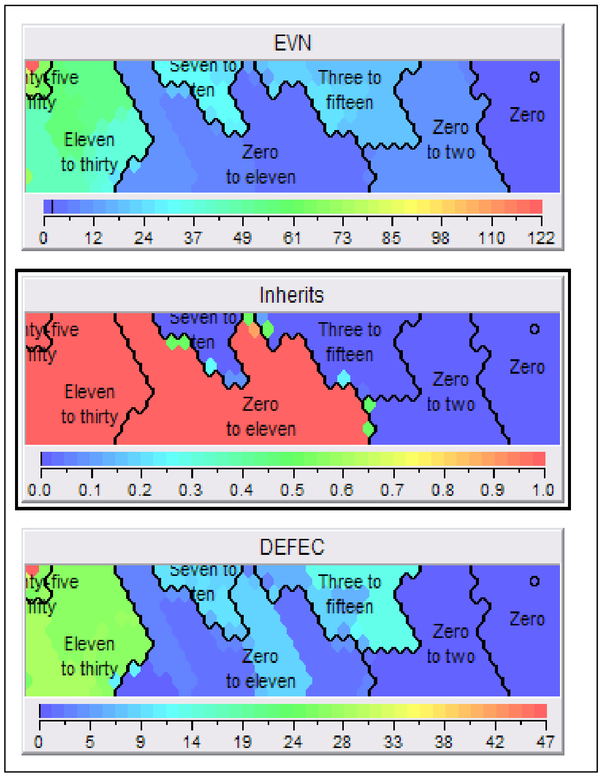

Figure 4. Component maps for classification of DEFEC based on EVN and Inherits

In terms of model performance, Tables 2 and 3 provide some indication of how well the SOM clusters the data. Cluster-based recall assigns all 32 classes correctly (Table 2). In itself the very good performance is not surprising – as a neural network it creates a non-linear mapping that fits the underlying data very closely (hence the inclusion of an unbiased test in Section 4). Such a classification could be used to allocate testing resources to those classes that contain more defects, or as the basis for informing refactoring decisions (if the number of events in a class is classed as high).

Table 2. SOM classification of defects – 7 clusters

Cluster	Correct:incorrect
Zero	9:0
Zero to two	5:0
Zero to eleven	6:0
Three to fifteen	2:0
Seven to ten	4:0
Eleven to thirty	4:0
Twenty-five to fifty	2:0
Overall	32:0

A comparison of the performance of the SOM and the regression model from [2] is shown in Table 3. We use two bias measures to indicate the fitness of the two models – sum of error and sum of absolute error. The former is the sum of the difference between the actual and modeled defect count over all 32 classes. If the model over-estimates the number of defects then the error is a negative number; if the model underestimates, the error is positive. The errors are totaled over all classes to give the sum of error shown. Use of this measure of bias is appropriate when it is acceptable to have over- and underestimates that balance out to close to 0 over the entire data set. If, however, *any* error is considered unacceptable then the sum of the absolute error is a more relevant indicator. (Note: in Table 3 we have also included a column for 'Adjusted Regression'. As the original regression model included a negative constant term several of the classes were assigned a value of -1 for the defect count. For these cases we changed the count to 0 and recalculated the bias.)

Table 3. Model error comparison for defect count

	Regression	**Adjusted Regression**	**SOM**
Sum of error	-1	-13	0
Sum of absolute error	92	85	3

The results presented in Table 3 indicate that both the regression and SOM models fit the data well in terms of the sum of error. A difference in performance is more evident, however, when the absolute errors are taken into account. Overall the regression models are out by 92 and

85 defects, whereas the SOM model is out by just three defects.

We then undertook model fitting of class size. This is not strictly a prediction exercise as again all of the data were used in the construction of the model (as in [2]). The overall multidimensional cluster map for this data is shown in Figure 5 and the associated component maps in Figure 6. The seven clusters enable us to identify similar classes in terms of their size and the underlying factors that may determine (or are at least related to) class size. In this case the strong relationships among the variables EVN, READ, STATE, WRITE and RWD are visually evident. The fact that these variables are also likely to be positively related to class size (in LOC) is also apparent.

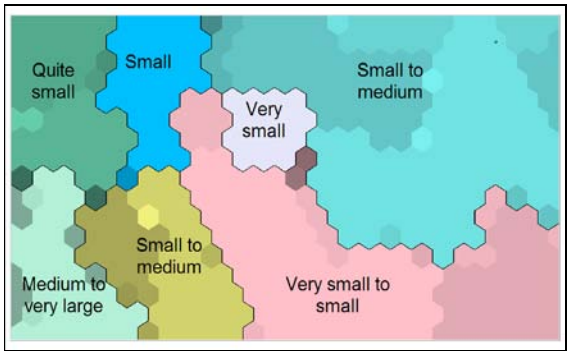

Figure 5. Overall cluster map for size classification data

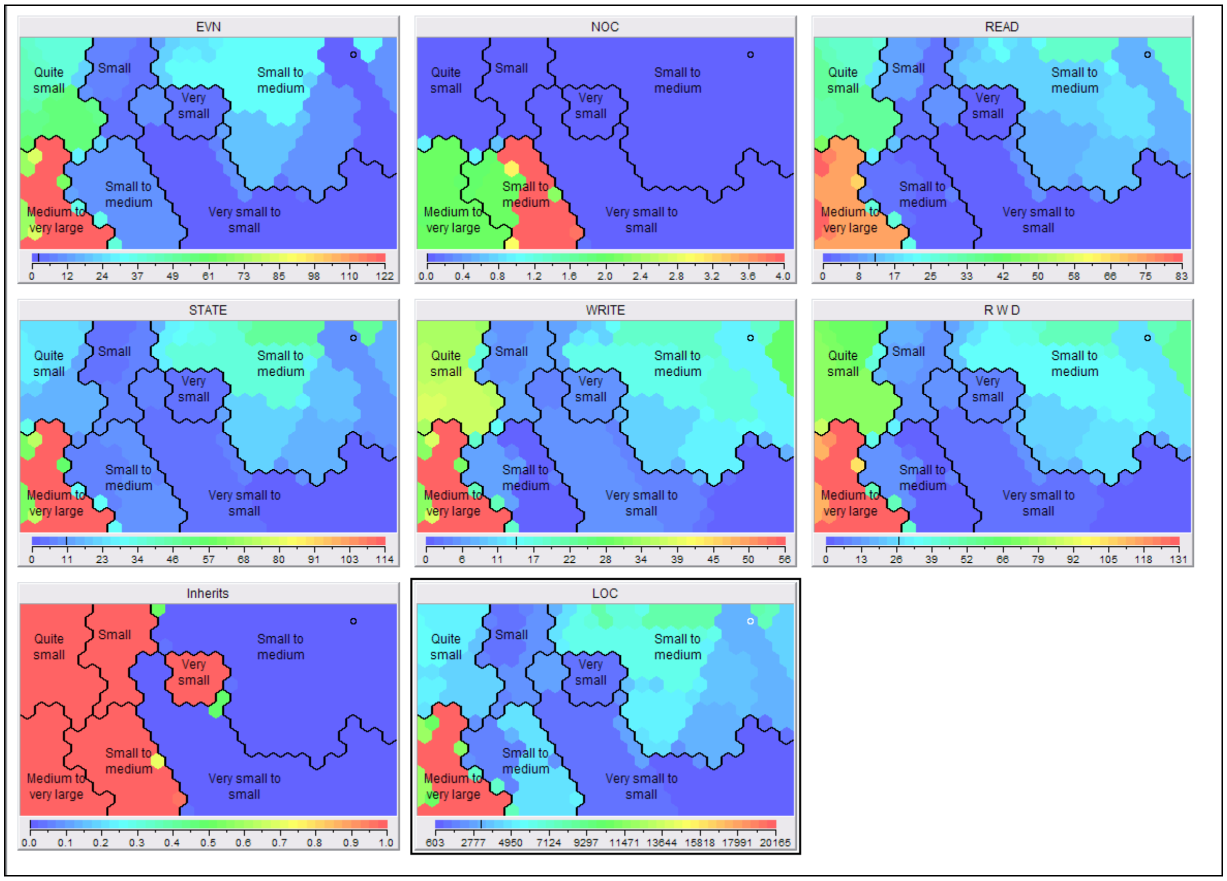

Figure 6. Component maps for size classification data

The original linear regression model [2] fitted a single independent variable – the count of states per class (STATE) – to size in LOC. We built a map using the same variable, again employing seven clusters as this produced a manageable number of groupings but with sufficient differentiation between them. The resulting cluster map is shown in Figure 7. The component maps underlying these clusters are shown in Figure 8. While the correlation coefficient can tell us something of the relationship strength between STATE and LOC (at 0.97) evidence of this very strong relationship can be seen more graphically in the map. We could use such a clustering, for instance, to allocate programming tasks among novice and experienced personnel on the basis that, other factors aside, the larger classes are going to be more challenging to develop, review and test. We again assess SOM performance based on classification robustness and compare its modelling accuracy to that of the regression equation (shown in Tables 4 and 5).

The classification results reported in Table 4 indicate that for all but one class the SOM proved effective. For one class (of 2699 LOC) the map assigned it to the Very small cluster (700-2400 LOC) whereas it should have been categorized as Small. Overall performance, however, is very positive. Our comparison of model error in this case (Table 5) reveals a similar outcome to the defect count case, in that both models perform well as measured by the sum of error but the SOM provides a better fit to the data if absolute error is taken as the accuracy criterion.

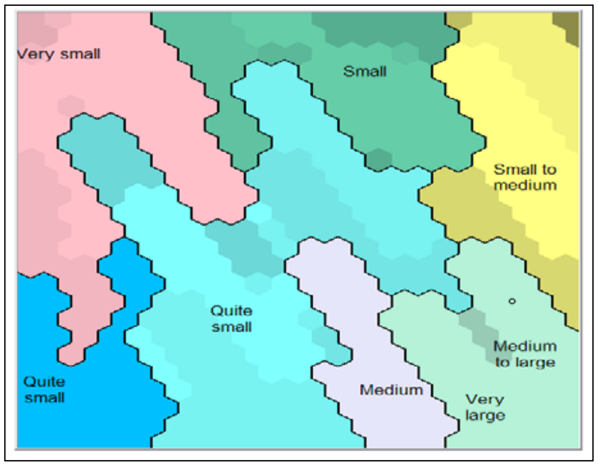

Figure 7. Clusters of LOC based on STATE

Table 4. SOM classification of LOC – 7 clusters

Cluster	Correct:incorrect
Very small (700-2400)	13:1
Small (2200-5200)	4:0
Quite small (3400-5000)	3:0
Quite small (3400-6000)	5:0
Small to medium (3700-8200)	3:0
Medium (6700-7000)	1:0
Medium to large; Very large (8000-33000)	2:0
Overall	31:1

Table 5. Model error comparison for LOC

	Regression	SOM
Sum of error	-6	1
Sum of absolute error	18926	1611

The results of these two analyses indicate that SOMs can be effective classifiers in terms of both accuracy *and* meaning, the latter being due to the graphical nature of the maps, enabling relationships to be depicted with more information content than a simple boxplot or correlation coefficient.

4. SOM-BASED PREDICTION

We now turn our attention to the use of SOMs in unbiased prediction. In this case we evaluate SOM effectiveness on a data set we analyzed in a previous study using linear regression methods [11].

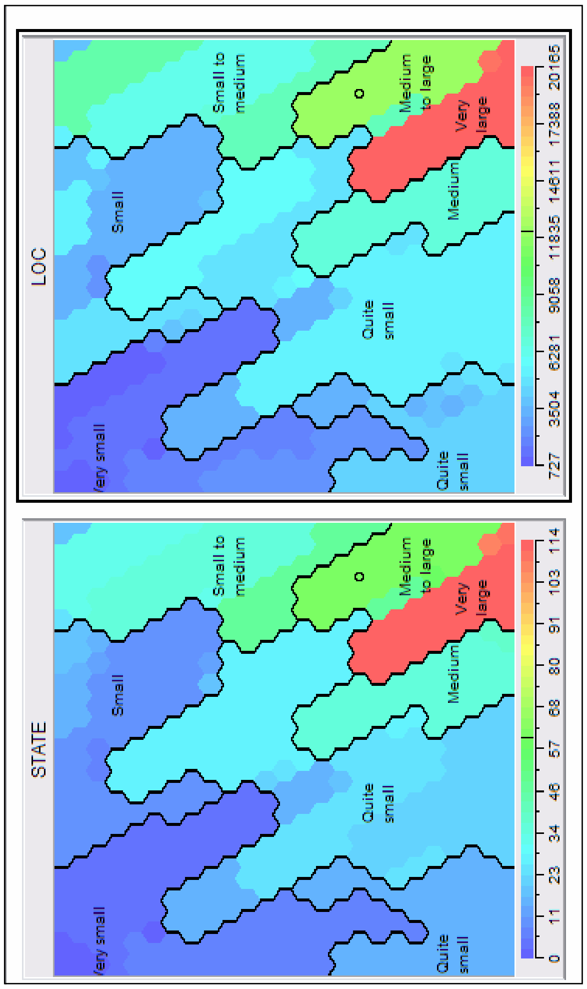

Figure 8. Component maps for classification of LOC based on STATE

The data set comprised more than seventy small-scale systems built using a common process and the same toolset – a data modeling tool and a 4GL. The systems "…were built over a period of five years by groups of senior students in the Department of Information Science at the University of Otago. Every system was built to satisfy the real requirements of an external client, normally a small business or a department in a larger organisation… Each system addressed transaction processing, data retrieval and reporting, and file maintenance activities performed by the organisation." [11, p.100]. The objective of the study was to examine whether a system's size (in source statements) could be predicted using a variety of measures taken from that system's specification. The variables and summary statistics relevant to this study are shown in Table 6.

Table 6. Prediction task – variables collected and summary statistics (adapted from [11])

Mnemonic	Variable	Explanation	Mean	Median	Min	Max
ENT	Entities	Count of entities depicted in the entity-relationship diagram (ERD)	11.6	11.0	4	26
RSHIP	Relationships	Count of relationships depicted in the ERD	10.2	9.0	3	25
ATTRIB	Attributes	Count of attributes associated with the ERD	64.5	59.5	25	141
MENU	Menus	Count of menu screens depicted in the Functional Decomposition Chart (FDC)	5.6	5.0	4	14
EDIT	Entry/Edit	Count of data entry/edit screens depicted in the FDC	12.0	11.0	4	27
REPORT	Reports	Count of reports depicted in the FDC	6.8	6.0	1	17
NONMENU	Non-menu functions	Count of non-menu functions depicted in the FDC	18.8	17.5	10	39
FDCSIZE	FDC Size	Count of all functions depicted in the FDC	24.4	22.0	14	45
SIZE	System Size	Count of all non-comment source statements in the implemented system	1106.0	993.5	309	2605

As in the model-fitting cases in the previous section the original analysis in this study utilized a very simple regression model, predicting system size based on the count of non-menu functions (NONMENU) depicted in the system Functional Decomposition Chart (FDC).

To perform an unbiased assessment of model performance we split the data set at random into training and testing subsets. From the total of seventy vectors we placed around two-thirds (47) into a training data subset, the remaining one-third (23) comprising the testing set. In order to reduce the likelihood of obtaining results that were a function of the particular data split rather than a more general outcome [20] we repeated this process five times. In each of five runs a SOM was then constructed using only the training data subset. We could then 'recall' the cluster membership of the vectors in the testing subset against the map, also producing a predicted value for system size. To illustrate this process, the maps and outputs produced for run 1 of this analysis are shown in Figures 9 (cluster map) and 10 (component maps) and in Table 7.

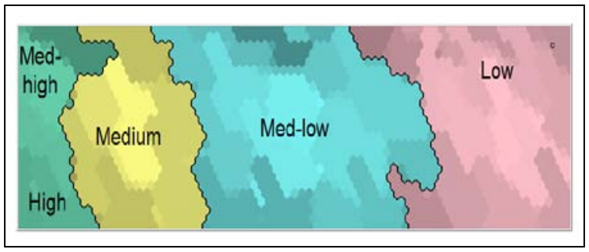

Figure 9. Clusters of SIZE based on NONMENU – run 1

Although four distinct clusters are evident in the cluster map for run 1 we were able to annotate the map with an extra 'cluster' representing those systems with medium-high counts of lines of code. This structure was the most effective in terms of the balance between cluster differentiation and the number of clusters. Clearly the larger the number of clusters the more precise the classification becomes; however, there are two negative consequences: 1. the clusters begin to overfit the training data reducing the map's general applicability; 2. it becomes impractical to act on such a highly fragmented classification in terms of adopting management strategies for different clusters.

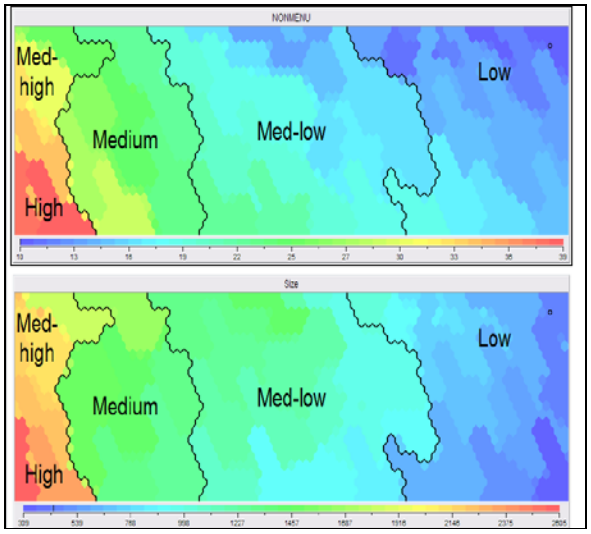

Figure 10. Component maps for classification of SIZE based on NONMENU – run 1

The component maps for run 1 illustrate the strong relationship between NONMENU and SIZE, with the two distributions following a similar pattern – systems with low counts are to the right and those with higher counts are to the left. While the correlation statistic (Pearson's rho = 0.87) also indicates this strong positive association it does not provide any indication of the dispersion of vectors underlying this relationship, something that the SOM representation is able to do. (Admittedly a scatter plot may also provide this sort of insight, but only for bivariate relationships.)

Predictive effectiveness over the testing data subset for run 1 is shown in Table 7 and in the first line of Tables 8 and 9. The results in Table 7 show the number of systems classified correctly or incorrectly when 'shown' to the independently trained map. We note that performance for all but the Medium cluster is very good. The system incorrectly classified as Med-low comprised 1573 lines of code (so should have been classed as Medium) and the two systems wrongly classed as Medium were made up of 1192 (Med-low) and 1831 (Med-high) lines of code respectively.

Table 7. SOM prediction of SIZE – 4 clusters, run 1

Cluster	Correct:incorrect
Low (300-1050)	11:0
Med-low (700-1500)	6:1
Medium (1250-1750)	2:2
Med-high; High (1750-2700)	1:0
Overall	20:3

We show the comparative results for predictive accuracy using SOMs and linear regression across all five runs in Table 8, based on the sum of error, and Table 9, based on the sum of absolute error. In Table 9 we also show the number of clusters used in the SOM and the aggregated classification performance.

Table 8. Model error comparison for SIZE – Sum of error

	Regression	**SOM**	**No. of clusters**	**Correct: incorrect**
Run 1	-526	615	4	20:3
Run 2	-1965	439	3	21:2
Run 3	-5078	35	4	21:2
Run 4	235	-8	5	22:1
Run 5	2171	498	4	20:3
Mean	*-1033*	*316*		
Median	*-526*	*439*		

Table 9. Model error comparison for SIZE – Sum of absolute error

	Regression	**SOM**
Run 1	6849	1367
Run 2	6221	1663
Run 3	5951	1113
Run 4	4941	892
Run 5	6520	1092
Mean	*6096*	*1225*
Median	*6221*	*1113*

We can see that in all but the first run the SOM outperformed the regression model in terms of the magnitude of the sum of error. It is also of interest to note that the SOM tended to overestimate SIZE, evident in four of the five runs, whereas the regression model underestimated SIZE in three of the five runs. Classification effectiveness over all five runs was strong, the two worst runs classifying three of the twenty-three systems incorrectly. The comparison of performance in terms of the sum of absolute error criterion is even more favorable for the SOM solutions. In all five runs the SOM led to substantially lower error totals than its regression counterpart.

5. DISCUSSION AND CONCLUSIONS

In this paper we have described the empirical analysis of self-organizing maps when applied to three software engineering problems, comparing them to benchmark regression models. There are three findings:

1. the visualization capabilities of SOMs can reveal useful information relating to dispersion of artifacts/vectors and the interrelationships among and between factors
2. SOMs appear to perform favourably in classification and when compared to 'equivalent' benchmark regression models
3. SOMs also appear to perform favourably in unbiased prediction and when compared to 'equivalent' benchmark regression models.

In all three of our analyses – defect classification, size classification and size prediction – we found the SOM method to be very effective in terms of accuracy. In all but one run of the size prediction exercise the SOM outperformed the matching regression model. Classification of individual vectors to clusters was also very positive. In particular, the strong showing in the prediction exercise is encouraging as this represents an unbiased assessment of SOM efficacy.

Furthermore, because of the structural simplicity of the SOMs these were challenging comparative tests. That is, given the strong linear relationships evident in the correlations for each analysis (between EVN and DEFEC, STATE and LOC, and NONMENU and SIZE respectively) we could have expected linear regression to perform very well. While it did, it was outperformed by the SOM in almost every case. Given that SOMs are also able to deal with nonlinearity and colinearity we suspect that they will prove to be just as effective for more complex models. This will form part of our future work, along with an assessment of the utility of the method in project management practice.

Even though the models were simple it was still an advantage to be able to visualize the relationships, both in aggregated cluster and component map forms. This enabled us to gain a sense of the overall distribution of artifacts in multidimensional space. Such a capability may enable users to manage artifacts more effectively e.g. by allocating testing resources according to likely defect counts, or by considering refactoring options for artifacts that are likely to become very large. The influence of factors both separately and together can be quantified *and* visualized using the maps. In returning to a question posed earlier, is this a solution looking for a problem? Perhaps. However the visualization capabilities offer an additional dimension to that afforded by other more commonly used methods that, in conjunction with good model accuracy, should encourage further consideration of the approach.

6. ACKNOWLEDGMENTS

We are grateful to the authors of the paper used in the classification case study [2] for publishing their raw data, enabling us to mirror their work.

7. REFERENCES

[1] Bril, R.J. and Postma, A. (2001) An architectural connectivity metric and its support for incremental re-architecting of large legacy systems. In *Proc. 9th Intl Workshop on Program Comprehension*, Toronto, Canada, pp.269-280.

[2] Cartwright, M. and Shepperd, M. (2000) an empirical investigation of an object-oriented software system. *IEEE Trans. Softw. Eng.* 26(8), pp.786-796.

[3] David, A. (2002) Tulip. *Lecture Notes in Comp. Sci.* 2265, pp.435-437.

[4] Fioravanti, F. and Nesi, P. (2000) A method and tool for assessing object-oriented projects and metrics management. *Jnl Sys. Softw.* 53, pp.111-136.

[5] Hong, E.-S. and Wu, C.-S. (1997) Criticality prediction models using SDL metrics set. In *Proc. 4th Asia Pac. Softw. Eng. Conf./4th Intl Comp. Sci. Conf.*, Hong Kong, pp.23-30.

[6] Irwin, W. and Churcher, N. (2003) Object oriented metrics: precision tools and configurable visualizations. In *Proc. 9th Intl Softw. Metrics Symp.*, Sydney, Australia, pp.112-123.

[7] Kang, B.-K. and Bieman, J.M. (1998) Using design abstractions to visualize, quantify, and restructure software. *Jnl Sys. Softw.* 42, pp.175-187.

[8] Kohonen, T. (1995) *Self-organizing Maps*. Springer, Berlin.

[9] Lee, J., Podlaseck, M., Schonberg, E., Hoch, R. and Gomory, S. (2000) Analysis and visualization of metrics for online merchandising. *Lecture Notes in Artif. Intell.* 1836, pp.126-141.

[10] Lee, J., Podlaseck, M., Schonberg, E. and Hoch, R. (2001) Visualization and analysis of clickstream data of online stores for understanding web merchandising. *Data Mining and Knowl. Discov.* 5, pp.59-84.

[11] MacDonell, S.G., Shepperd, M.J. and Sallis, P.J. (1997) Metrics for database systems: an empirical study. In *Proc. 4th Intl Softw. Metrics Symp.* Albuquerque, USA, pp.99-107.

[12] Noble, J. (1998) Integrating metric visualisation into a commercial user interface builder. In *Proc. Australian Conf. Comp.-Human Interact.*, Adelaide, Australia.

[13] Noble, J. and Constantine, L.L. (1996) Interactive design metric visualization: visual metric support for user interface design. In *Proc. Australian Conf. Comp.-Human Interact.*, Hamilton, New Zealand, pp.213-220.

[14] Oh, S.-K., Pedrycz, W. and Park, H.-S. (2002) Self-organising networks in modelling experimental data in software engineering. *IEE Proceedings – Computers & Digital Techniques* 149(3), pp.61-78.

[15] Oh, S.-K., Pedrycz, W. and Park, B.J. (2004) Self-organizing neurofuzzy networks in modeling software data. *Fuzzy Sets and Sys.* 145, pp.165-181.

[16] Pedrycz, W. and Peters, J.F. (1998) *Computational Intelligence in Software Engineering*. World Sci., Singapore.

[17] Pedrycz, W., Succi, G., Musílek, P. and Bai, X. (2001) Using self-organizing maps to analyse object-oriented software measures. *Jnl Sys. Softw.* 59, pp.65-82.

[18] Pedrycz, W. (2002) Computational intelligence as an emerging paradigm of software engineering. In *Proc. 14th Intl Conf. Softw. Eng. Knowl.Eng. (SEKE'02)*, Ischia, Italy, pp.7-14.

[19] Reformat, M., Pedrycz, W. and Pizzi, N.J. (2002) Software quality analysis with the use of computational intelligence. In *Proc. 2002 IEEE Intl Conf. Fuzzy Sys*, Honolulu, USA, pp.1156-1161.

[20] Shepperd, M. and Kadoda, G. (2001) Comparing software prediction techniques using simulation. *IEEE Trans. Softw. Eng.* 27(11), pp.1014-1022.

[21] Zhong, S., Khoshgoftaar, T.M. and Seliya, N. (2004) Unsupervised learning for expert-based software quality estimation. In *Proc. 8th IEEE Intl Symp. on High Assurance Sys. Eng.*, Tempa, USA, pp.149-155.