DIVISION OF COMPUTER SCIENCE

Using Neural Networks to Analyse Software Complexity

Simon Field
Neil Davey
Ray Frank

Technical Report No 217

January 1995

Using Neural Networks to Analyse Software Complexity

Simon Field†*, Neil Davey*, Ray Frank*

†BNR Europe Limited,
London Road, Harlow, Essex, UK CM17 9NA
E-mail: S.D.H.Field@bnr.co.uk

**University of Hertfordshire,*
Hatfield, Hertfordshire, UK. AL10 9AB
E-mail: N.Davey@herts.ac.uk, R.J.Frank@herts.ac.uk

Abstract

Units of software are represented as points in a multidimensional space, by calculating 12 measures of software complexity for each unit. Two large sets of commercial software are thereby represented as 2236 and 4456 12–ary vectors respectively. These two sets of vectors are then clustered by a variety of competitive neural networks. It is found that the software does not fall into any simple set of clusters, but that a complex pattern of clustering emerges. These clusters give a view of the structural similarity of units of code in the data sets.

1. Introduction

In this paper we examine whether units of software fall into natural clusters when represented by a set of complexity measures. The work[1] extends an analysis [1] in which C source code was mapped onto vectors in a 12 dimensional space, and then clustered using a simple competitive neural net. They found that the vectors were represented by 3 code vectors, which they then labelled as identifying standard, marginal and non–standard code. If legitimate such a classification has obvious value as a tool for software maintenance and system engineering.

Our work extends this analysis in two ways:

- The source code used is two large sets of commercial software, giving the opportunity to evaluate the stability of the results with respect to the selection of software.
- The clustering methods used make use of several different types of self organising neural network, to give results less dependant on the limitations of one particular algorithm.

Incidentally we are also able to make a limited comparative evaluation of four different self organising neural networks.

[1] Details of this work were originally presented at the *International Workshop on Applications of Neural Networks to Telecommunications (IWANNT)*, Stockhomlm, Sweden, May 22–24 1995.

1.1 The Data

The first data set is a set of 2,236 procedures drawn from a single software product written in the proprietary BNR language PROTEL[2], a block structured language designed for the control of telecommunication systems. The second set is of 4,456 PROTEL procedures drawn form a variety of software products.

1.2 Networks Used

We use four self organising networks with a variety of parameter settings. These are: a simple competitive net (SCN); a competitive net with a conscience mechanism — Frequency Sensitive Competitive Net (FSCN); a self organising map (SOM); and Fuzzy ART. The motivation for this choice is:

- SCN: To attempt to replicate the original results of [1].
- FSCN: To examine if the small number of nodes used in the SCN classification and concomitant dead units could be overcome.
- SOM: To investigate if a global order was present in the representation space.
- Fuzzy ART: To avoid prescribing the number of categorising units to be used.

1.3 Organisation of the Paper

Section 2 of the paper deals with the motivation for finding clusters in the complexity space, overviews the original work of [1], discusses the complexity measures used, their extraction from the source code and the mapping to the final 12–ary vectors used as inputs to the classifiers. Section 3 describes the networks used and the parameter settings investigated. Section 4 discusses the results obtained from the experimental work and section 5 concludes.

2. Background

2.1 Why use software metrics?

Improving software product quality and performance and development team productivity has become a primary priority for almost every organisation that relies on computers [2]. As the size and complexity of software systems grows, then the management of these systems becomes ever more vital. For instance, the US NASA Space Shuttle software system required 25.6 million lines of code, 22,096 staff–years of effort, and cost $1.2 billion [3]. Some typical software system details are given in Table 1 below. With the large capital investment necessary to develop such systems, it is clear how important it is to have control over the whole development process. Software measurement is a valuable technique that can be used to understand, guide, control and improve software development [4], and many metrics have been proposed, including those based on complexity used here.

[2] As in [1], the method and metrics used are language independent.

Table 1: Typical Software Application Size and Investment [2]

Product Application	Size (million lines of code)	Cost ($millions)
Operating System, Large Communication System	2 – 5	150 – 350
Mid-sized Communication System	1 – 2	50 – 100
Data Base System, Compiler	0.4 – 1	9 – 22
Transaction Monitor	0.2 – 0.4	5 – 10
Monitoring System (Medical)	0.2 – 0.4	4 – 8

2.2 Metrics Used

Twelve metrics were used in [1], and thus the same twelve were used for this work. The measures include McCabe's Cyclomatic Complexity (V_g) [5], Halstead's Software Science measures [6,7], and a number of others including a simple count of lines of code, and a measure called the Control Density of the code.

Halstead's measures were developed as an approach to analysing software as independently of the programming language as possible, and thus the results should not differ between the 'C' code used in [1] and the PROTEL used for this study.

The twelve measures of complexity calculated for each procedure are shown in Table 2.

Table 2: Complexity measures used

LOC	number of lines of code
V_g	McCabe's cylomatic complexity measure
D	Vg:LOC
n_1	number of unique operators
n_2	number of unique operands
N_1	total number of operators
N_2	total number of operands
n	$n_1 + n_2$, the vocabulary of the procedure
N	$N_1 + N_2$, the length of the procedure
V	N(log n), the volume of the procedure
$\hat{L}$	$\frac{2}{n_1} \times \frac{n_2}{N_2}$ program level
E	$\frac{V}{\hat{L}}$ effort

Lines of code (LOC) is the simplest of the measures, and provides information on the size of the program. It includes both the executable and the comment lines. This measure is not a particularly good one as it will obviously vary according to the programmer's style, but it is a reasonable indicator of complexity.

McCabe's Cyclomatic Complexity (V_g) is based on the control–flow graph of a program/procedure. A simpler version of V_g, which is calculated by counting the decision points in a function (e.g. IF, FOR,

WHILE, CASE) and adding 1, was used in [1] and is therefore used for this work. V_g is not a sufficient measure of software complexity on its own as, for instance, it ignores the effects of nesting, however a set of possible standards have been supplied [8] to use when analysing code with V_g.

Control Density (D) is a measure that combines both LOC and V_g. It's interpretation in [1] proved to be somewhat uncertain. A high density should indicate a high complexity, however according to their results the opposite was true. Their SCN deemed it to be the significant feature when classifying Standard code — a high value of D was found in all the procedures that they classified as Standard.

The next nine measures are all Halstead's Software Science measures. They include the following: various measures which count operators and operands; a measure of the *volume* (size) of the function, based on an estimate of the number of bits required to store the function in memory; the *program level* of the function, intended to give an indication of how difficult it is to understand the function; and the *effort* of the function, based on both the volume and the program level, which gives an indication of the effort required to write the function.

Notice that the last five measures are defined from the numbers of operand/operators in the code. We would not, however, expect a self organising neural network to find the same clusters without this information as with it, since all such networks use a first order measure of similarity [9].

2.3 Sheppard and Simpson's work

A simple competitive net was used. Competitive learning was developed to identify features in the input space without the need to explicitly train the network to recognise these features. During training nodes within the network compete amongst themselves to respond to the input vector, and the node that subsequently wins this competition then has a feature associated with it. This continues until the network has developed a set of feature detectors which account for all of the input space. During recall the net will receive an input vector containing a learned feature, the activity of the network will then lead to a node becoming active, thereby identifying which of the features is present in that input vector.

Over 4,000 functions written in C were analysed, and these were taken from approximately 20 different programs, written by different programming teams. The net consisted of twelve input nodes and twelve output units. Nine of the output nodes were found not to be involved in the classification process. This may have been indicative of three natural clusters in the input space, but could have been caused by the frailty of the particular algorithm used. A second network was then constructed with twelve input nodes and three output nodes, and the results were the same.

The three categories identified were Standard (95.1% of code analysed), Marginal (4.75%), and Non–Standard (0.15%). Standard code was based on the value of D, and in fact classified code with a high control density which was contrary to what was expected. Marginal code was based on the values of n_1 and n_2, meaning that the vocabulary was excessive. Non–Standard code had a high value for lines of code and V_g.

2.4 Pre–processing the data

This task accounted for over 50% of the effort expended on this work. Because PROTEL procedures can be of a user–defined type it was necessary to develop a parser to correctly identify all the procedures in order to then record the information.

The following data was recorded for each procedure: a count of the lines of code; a measure of McCabe's Cyclomatic Complexity; a simple count of the unique operands and operators; and a cumulative total of operands and operators. This data was then further processed to produce the full twelve measures needed. See below for details of how the final vectors were produced.

2.5 The vectors produced

Each procedure is therefore represented by a 12–ary vector. Within a data set the vectors of raw measures are not used since they widely differ in magnitude; rather each feature is represented as a proportion of the largest value for that feature. Specifically for the raw set of vectors: $V_{raw} = \{\mathbf{v}_i\}$ the actual vectors used are: $V = \left\{ \dfrac{\mathbf{v}_i}{\max_j(\mathbf{v}_{ij})} \right\}$

3. Networks Used

3.1 Self Organising Neural Networks

There is now a whole range of neural network architectures that are self–organising. In these nets the output units, or more precisely the weight vectors of these units, move in the input space so as to minimise a cost function. The input vectors are then classified into clusters according to which output unit is closest. Problems arise when using such networks to find clusters in two respects:

- The number of output units may influence the number of clusters that will be found in the input.
- The network may converge on a poor classification of the input, with some of the output units not participating in the final classification. Such units are said to be dead.

A variety of methods have been proposed to overcome these problems, including: feature mapping [10]; a conscience mechanism [11]; and Adaptive Resonance Theorem (ART) [12].

We make use of four different networks to cluster the input. These are:

- Simple Competitive Net
  — the network uses Euclidean distance to find the winning unit and the winning unit moves towards the input.
- Frequency Sensitive Competitive Net: a simple competitive net with a conscience mechanism
  — the measure of success of a unit in the competition to classify an input is scaled by a factor inversely proportional to its historical success.
- Self Organising Map [10]
  — the units are arranged in a two dimensional grid, with a winning unit pulling its neighbours with it as it moves.
- Fuzzy ART [13]
  — a version of ART for continuous valued inputs using the fuzzy set theoretic measure of intersection.

These four networks are described in more detail below.

3.2 Simple Competitive Net

Initially the net consists of a set of nodes with small random initial weights. When an input is presented the winning node is selected as that with greatest similarity to the input. The winner is then modified to become more similar to the input. We use Euclidean distance as the measure of similarity and therefore simply move the weight vector of the winner towards the input in Euclidean space. The size of the update is determined by a learning rate.

For input x and weight vectors: $\{w_i\}_i$ the winner, w_{i^*} is defined by: $\forall \mathrm{i} \bullet |w_{i^*} - x| \leq |w_i - x|$

The change in the weight vectors is then given by:

if $i = i^*$ then $\partial \mathbf{w}_i = \rho(x - w_i)$ where ρ is the learning rate

else $\partial \mathbf{w}_i = 0$

This type of network is prone to poor classifications, due partly to the problem of dead units — units that accidentally never win in the competition and thus never move towards the inputs

3.3 Frequency Sensitive Competitive Net

In order to overcome the above problem one mechanism, originally suggested by [11], is to make it harder for frequently winning units to win again. This is done by subtracting a bias term from each unit, that is proportional to the number of times the unit has one. Stabilisation can be problem with this method and we follow the scheme suggested by [14]. Here the bias is defined by:

$$B_i = \gamma\left(F_i - \frac{1}{N}\right)$$

Where γ is a problem specific parameter, N is the total number of units and F_i is the win count of unit i

A potential problem with this type of network is that the biasing mechanism, whilst producing reasonable code vectors for the data set, may obscure natural clustering in the data.

3.4 Self Organising Map

In a SOM [10] the output units are arranged in a fixed topology, usually a two dimensional grid. We use a grid, wrapped around at the boundaries. Winning units pull their neighbours in the grid with them, towards the input. The neighbourhood of a unit should initially be set a large fraction of the output space and decrease over time. A useful heuristic for setting the neighbourhood is that it should start at roughly half the output space and decrease to unity, to ensure global order. We use a ten by ten SOM and therefore initialise the neighbourhood to five. We also include a conscience mechanism as in 3.3. A successfully trained SOM provides more information than the other networks used here. The SOM not only captures clustering in the input space but also the relative position of these clusters, which can be located from their position in the output space.

3.5 Fuzzy ART

Fuzzy ART [13] is an extension of the basic ART algorithm to allow for continuous, rather than binary, data. In the original ART1 network, proximity is measured by the dot product, or equivalently by mapping the Boolean AND over both vectors and by taking Norm 1 of the result. In Fuzzy ART the fuzzy logic AND connective, min, is used to extend the method to real values.

The algorithm then is:

1. Begin with a set of units, all of which are disabled.
2. For each epoch and each input vector
   2.1 Select from the enabled units the unit that is closest to the input. Check if this unit satisfies the vigilance test. If not, select the next closest enabled unit and recheck.
      2.1.1 If no enabled unit satisfies the vigilance test, commit one of the disabled units and set it equal to the input. If no disabled units exists continue from 2.
   2.2 If a winner is found move it towards the input.

Three details must be specified:

- Proximity: the closeness, d(**x**,**w**) of the input **x** to the unit **w** is defined by:

$$d(\mathbf{x},\mathbf{w}) = \frac{\|\mathbf{x} \wedge \mathbf{w}\|_1}{\varepsilon + \|\mathbf{w}\|_1}$$

where $\|\mathbf{x}\|_1 = \sum_i x_i$, $\mathbf{x} \wedge \mathbf{w} = \left(\min(x_i, w_i)\right)_i$ and ε is a small value to break ties.

- Vigilance: the unit **w**, satisfies the vigilance test for input x, if:

$$\frac{\|\mathbf{x} \wedge \mathbf{w}\|_1}{\|\mathbf{x}\|_1} \leq \delta$$

where δ is the vigilance parameter

- Movement: the winner, w, moves towards the input x, according to:

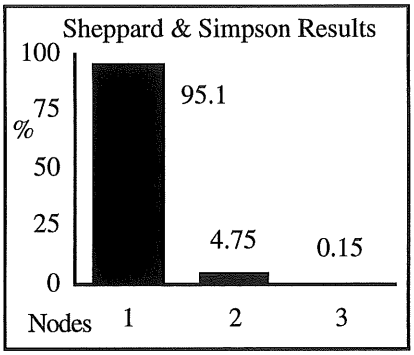
$$\mathbf{w}' = \mathbf{x} \wedge \mathbf{w}$$

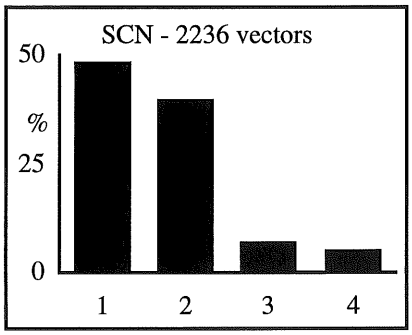
The behaviour of the net is determined by the value of the vigilance parameter and the number of nodes in the initial disabled set. In our experiments we used a pool of nodes greater than required, and a variety of vigilance parameters.

Fuzzy ART is appealing as the clusters are formed dynamically but the algorithm has some specific problems. As noted by [9] the weight vectors drift towards the origin. This is due to the update rule: components of weight vectors are monotonically decreasing. As described by [12] some of the difficulties of this can be overcome by complementary coding; but this has the unfortunate side effect of doubling the length of the input vectors.

4. Results and Discussion

The results obtained from the simple competitive net were radically different from the other three nets. The SCN found a simple pattern of 4 clusters whilst a more complex pattern emerged with the other networks — all found numerous clusters in the data. As demonstrated by the more sophisticated clusterers, this was the result of an inherent tendency of the SCN to not utilise some of its units; all the other networks have mechanisms for dealing with dead units. The FSCN and the SOM produced very similar clusters. Fuzzy ART produced many more clusters than any of the other nets, and clustered the data in a different way from the FSCN/SOM. The results are now given in more detail.





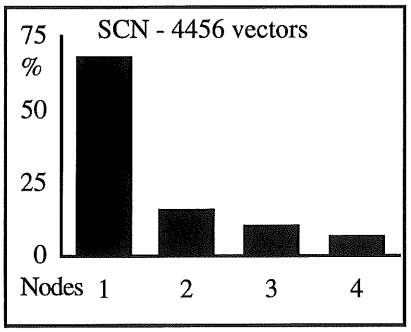
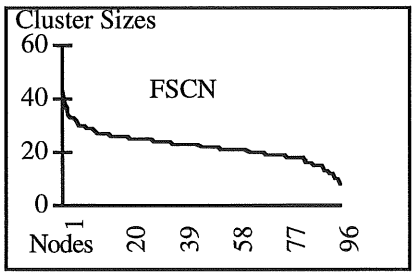
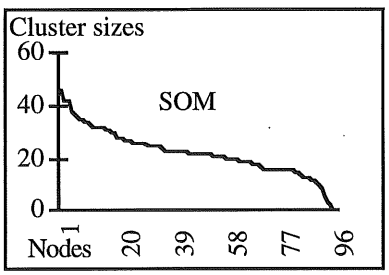
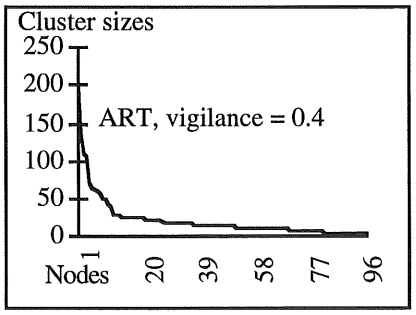


Figure 1: The number of clusters by three SCNs







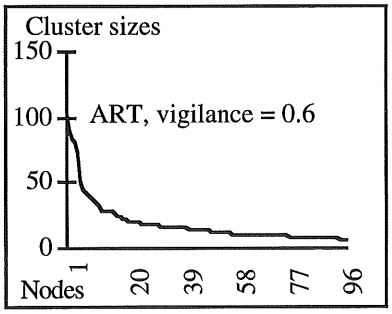


Figure 2: 2236 Vectors set, classified by four networks

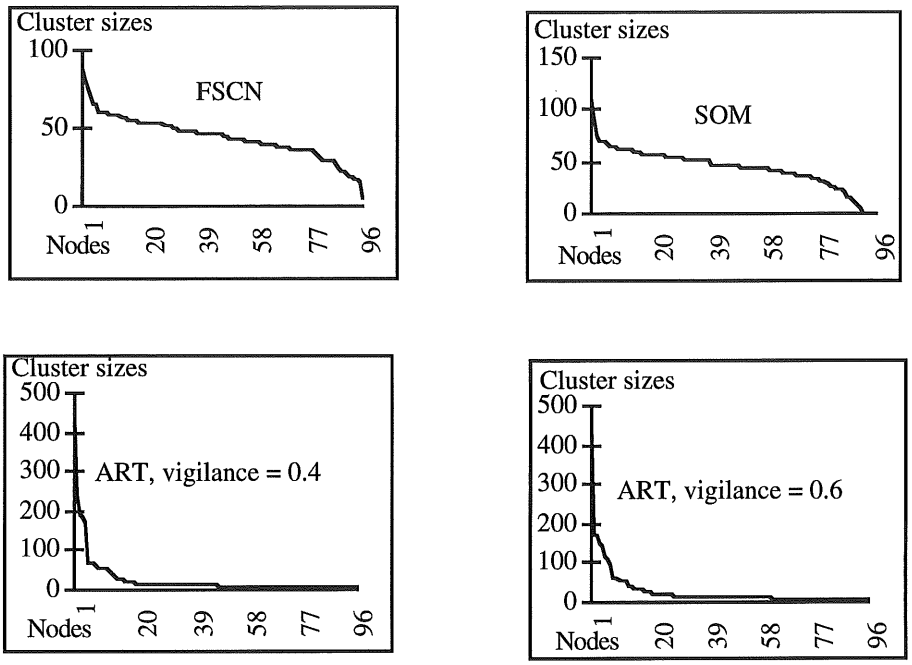


Figure 3: 4456 vectors classified by four networks

4.1 SCN

This experiment was an attempt to replicate the results obtained in [1] and so the net was initialised for both data seta with 12 units. In both cases only four of the twelve units were used. The clusters were identified as follows: Node 1 clustered "average" procedures, i.e. those with no particularly special features; Node 2 clustered procedures with a large language content compared to their size; Node 3 clustered very small procedures — those with less than 10 lines; and Node 4 clustered very large procedures — those with large values for all the measures.

4.2 FSCN

For both data sets the net was initialised with 100 units, all of which were used after training. For the smaller data set the resulting clusters ranged in size from 43 vectors to 8, with a uniform spread.

4.3 SOM

Again 100 units were used, this time arranged in a ten by ten grid. The initial neighbourhood was set to five and decreased over the training time to one. After training two units of both nets did not participate in the classification. The classification produced was quite similar to the FSCN. However with the SOM additional proximity information could be inferred from the position of the classifying nodes in the SOM grid. For example two vectors grouped by Fuzzy ART were found to be classified by adjacent nodes in the SOM.

4.4 Fuzzy Art

The net was trained over both data sets with two settings of the vigilance parameter: 0.4 and 0.6, and a pool of 2000 uncommitted nodes. As expected the net produced more clusters with the higher vigilance than lower. Table 3 below details the results from the two data sets.

Table 3: Nodes used by Fuzzy ART

Data set	2236 vectors		4456 vectors	
Vigilance	0.4	0.6	0.4	0.6
Total Number of nodes	155	227	619	930
Number of Dead Units	17	17	70	252

It is noticeable that the number of clusters found is not linear in the size of the data set; the net may be identifying increased variation in the larger data set, due to its multi–product source.

A problem with the Fuzzy ART network becomes apparent when the weight vectors of the classifying units are examined — they are generally very small. The mean weight vector for the smaller data set and vigilance 0.4 is:
(0.026, 0.003, 0.120, 0.062, 0.031, 0.036, 0.079, 0.040, 0.024, 0.031, 0.031, 0.001) with length 0.178.

This compares with the mean vector for the raw data set:
(0.076, 0.034, 0.245, 0.101, 0.075, 0.071, 0.126, 0.079, 0.057, 0.150, 0.140, 0.013) with length 0.394.

Another major difference between the earlier nets and Fuzzy ART is that the former are sensitive to frequency densities in the input space: if an area of the input space is heavily populated then many units will move there. This is not the case with Fuzzy Art, since a single unit will classify an area if all the vectors within it are sufficiently close and satisfy the vigilance test.

5. Conclusions

This work highlights the care with which unsupervised clustering must be undertaken. It is clear that resultant clusters are highly dependant on the architecture and parameter settings used. In particular SCNs may give unrepresentative results due to the problems of dead units.

Some of the limitations of using the ART type architecture are described above and it is difficult to make a direct comparison due to the different distance metric used. The FSCN and the SOM gave similar results, with the SOM providing additional information: the topological map.

These methods do not provide a simple way of partitioning software units on their complexity, however the pattern of similarity could be potentially useful in locating procedures of related complexity. A similar application of neural nets to the location of software clones is described in [15].

Finally, it should be noted that the major activity of work such as this is the pre–processing of data.

Acknowledgements

We would like to acknowledge the financial assistance of the Department of Trade and Industry of the UK Government, who have partly funded the placement of Simon Field on a Teaching Company Scheme (TCS) at BNR Europe Limited, through the Teaching Company Directorate (TCD) organisation, under grant no. TCS–1326.

References

[1] J.W. Sheppard and W.R. Simpson,"Using a competitive learning neural network to evaluate software complexity", ACM, pp. 262-267, 1990.

[2] K.H. Möller & D.J. Paulish, *Software Metrics*, London: Chapman & Hall, 1993.

[3] B.R. Schendler, "How to Break the Software Logjam", *Fortune*, pp. 72-76, 25 September 1989.

[4] C.M. Lott and H.D. Rombach, "Measurement-based guidance of software projects using explicit project plans", *Info. and Soft. Tech.* Vol. 35 No. 6/7 pp. 407-419, 1993.

[5] T.J. McCabe, "A Complexity Measure", *IEEE Trans. on Software Engineering*, Vol. 2 No. 4, pp. 3-15, 1976.

[6] M.H. Halstead, *Elements of Software Science*, New York: Elsevier Science, 1977.

[7] P.G. Hamer and G.D. Frewin, "M.H. Halstead's Software Science – A Critical Examination", in *Proceedings of the 6th International Conference on Software Engineering* pp. 197-206, 1982.

[8] T.C. Jones, *Applied Software Metrics,* New York: McGraw-Hill, 1991.

[9] B. Moore, "ART1 and Pattern Clustering", in *Proceedings of the 1988 Connectionist Models Summer School* , eds. D. Touretsky, G. Hinton and T. Sejnowski, San Mateo: Morgan Kauffman, 1988, pp. 174-185.

[10] T. Kohonen, "Self-Organised Formation of Topologically Correct Feature Maps", *Biological Cybernetics* Vol. 43, pp. 56-69, 1982.

[11] S. Grossberg, "Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neural Feature Detectors", *Biological Cybernetics* Vol. 23, pp. 121-134, 1976.

[12] G.A. Carpenter, and S. Grossberg, "A Massively Parallel Architecture for a Self-Organising Neural Pattern Recognition Machine", *Computer Vision, Graphics and Image Processing* Vol. 37, pp. 54-115, 1987.

[13] G.A. Carpenter, S. Grossberg and D.B. Rosen, "Fuzzy ART: Fast Stable Learning and Categorisation of Analog Patterns by an Adaptive Resonance System", *Neural Networks* Vol. 4, pp. 759-771, 1991.

[14] D. DeSieno "*Adding a Conscience to Competitive Learning*", IEEE International Conference on Neural Networks, Vol. 2, pp. 117-124.

[15] S. Carter, R.J. Frank, and D.S.W. Tansley, "Clone Detection in Telecommunication Software Systems: A Neural Net Approach", in *Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications* pp. 273-280, 1993.