2001

Pattern recognition in software engineering trend adapting

Dapeng Chen
West Virginia University

Pattern Recognition in Software Engineering Trend Adapting

Dapeng Chen

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
in
Computer Science

Ali Mili , Ph.D., Chair
Hany Ammar, Ph.D.
Robert Cowan

Department of Computer Science & Electrical Engineering
2001

Keywords: Neural Networks, Pattern, Software Engineering Trends

ABSTRACT

Pattern Recognition in Software Engineering Trend Adapting

Dapeng Chen

Whether and when to adapt to certain software engineering trends are difficult questions to be answered by many decision-makers. The main reasons are due to the fact that evolution of software engineering trends itself is determined by various factors, many of which come from the fields outside of the software technology, thus hard to predict. So it is even harder to estimate the cost and benefit when adapting to certain trends. This paper is intended to study ways to decrease the risk involved in such decision making processes, by developing a pattern from past software engineering trends. While the pattern cannot answer all the questions by itself, it can relief the decision makers in a large extent by providing the most important information relevant to the software engineering trends. The pattern recognition is achieved by using neural networks. Our result seems to be very encouraging, which begins to prove that there does exist pattern between the input data that we can observe and the output data that we need to know. Although more trends need to be observed and analyzed before we can reach a more concrete conclusion, it does show that neural network may be a valid approach in future research.

Table of Contents

Chapter 1 Introduction to Software Engineering Technology Watch

This paper is part of the research in the Software Engineering Technology Watch (Tech Watch) project. To illustrate the problem that this paper is trying to solve, it is necessary to review briefly the Tech Watch project.

The Tech Watch project is proposed to illusive the elusiveness of the software engineering trends. By launching a watch initiative in Software Engineering, Tech Watch raises a number of questions about the evolution of technology, and about the means that people can deploy to understand the forces that drive this evolution[1]. This approach can be characterized by the combination of the following two premises which will give sufficient latitude to gain some insight into the problem and make some tentative inroads towards addressing it.

- *Structuring the Problem. To pursue tech watch, one realizes that there are many questions that beg for answers. In most cases, all these questions are interrelated. The first order of business, for this project, is thus defined as building a questionnaire structure, which arrange all these questions in a way that attempts to highlight their interrelations.*
- *Diversifying the Solution. The Tech Watch project distinguishes among three research methods: analytical research, which attempts to understand the phenomena that underlie observed behavior, and build models that capture these phenomena; empirical research, which makes no attempt to understand cause/effect relationships, but merely attempts to capture observed behaviors*

by empirical models; experimental research, which intervenes after analytical or empirical research to validate the proposed models.

In order to fulfill the general goal of Tech Watch project, the questionnaire structure is divided into four layers. At the topmost level of the hierarchy is the distinction between four families of questions:

- *How to watch software engineering trends? This question deals with what indicators we need to monitor, where to find them, and how to interpret them.*
- *How to predict software engineering trends? This question deals with what lifecycle we believe that software engineering trends follow, and what drives the evolution of a trend from one phase to another along the lifecycle.*
- *How to adapt to software engineering trends? This question deals with how does one define institutional strategy in such a way as to maximize benefit from what is known about a trend and minimize risk from what is not known about it.*
- *How to affect software engineering trends? This question tries to identify where, in the cycle of a trend, is it possible to alter the course of the trend, and eventually how, and by whom.*

These four questions represent the four branches of the TECH WATCH research. The general goals and the state of art of these branches are as follows.

1. **Watching Software Engineering Trends.**

The General goal of watching trends is not offering any concrete answer about predicting and adapting trends. Instead, the goal is to determine what information we must maintain in order to gain a comprehensive view of the discipline and its evolution. The information in question must be rich enough to support both discipline-wide assessment and trend-specific analysis. The project not only concerned with what information to collect, but also where to find it, how to derive it, and how to keep it up-to-date.

Specifically, questions to be addressed while we do the trend-watching include:

- What is the relevant information that must be collected/monitored?
- Where do we find this information, or where do we infer it from?
- How do we interpret this information?
- How often do we need to update this information?

A number of software engineering-specific, and technology-related indicators have been identified and divided into the following categories: Classification Standings, Research and Developments, Science and Technology Output, Human Resources, Cost and Funding, Standards and Regulations and Best Practices.

2. Predicting Software Engineering Trends

Among all questions to be answered, how to predict software engineering trend is probably the most important one. As long as the lifecycle is identified, we will be able to answer questions such as: what factors determine the success (or failure) of a trend? How early can such factors be assessed? Which success factors are controllable? What phases in the lifecycle lend themselves to external interventions?

While a definite lifecycle that software engineering trends follow has not been derived, a generic evolutionary model which includes three cycles is proposed to capture these trends. The model is shown in figure 1.

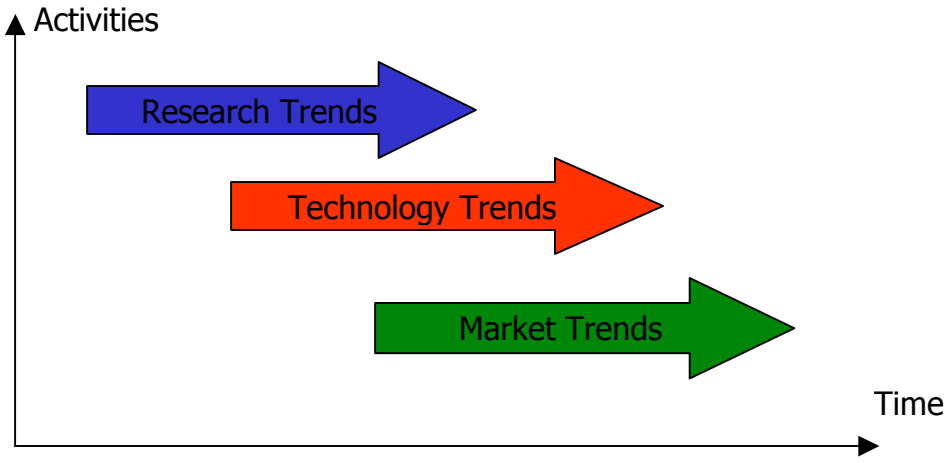


Figure 1. Generic Evolutionary Cycle Model

As we can see from above, three cycles are recognized to define the lifecycle of software engineering trends, which are Research, Technology Transfer and Market.

- ***Research Trends***. Research trends are driven by general perceptions of the state of the art and the state of the practice, by researcher perceptions of practitioner needs, by national funding programs that rally around specific strategic goals, and by sheer technical interest. For this trend, Tech Watch tries to identify research issues and non-issues, as well as theoretical and practical research goals. It also proposes analytical vs. empirical research methods, and formulates realistic expectations.
- ***Technology Trends***. Technology trends are driven by the maturation of applicable research ideas, and by the successful evolution of the idea to a useful, technologically viable product. In order for a research idea to turn into a concrete product, three conditions must be satisfied simultaneously: 1) the idea must be mature, 2) there must

be an actual or potential market for the product. 3) There must be an economically viable way to make this technology available on the market. As far as technology trends are concerned, Tech Watch identifies the following sub goals: 1) Track promising research ideas, measuring their maturity, market potential, and technological viability; 2) Identify technology bottlenecks; 3) Track current technological needs, and their evolution as market shift. 4) Identify/track general trends in venture capital.

- ***Market Trends***. Just as technological trends can influence research trends, market trends can in turn influence technological trends in the following two ways: either by providing new products, or by creating a new market. Thus, the goal of Tech Watch research will be watching changes in both supply side and demand side.

3. Adapting to Software Engineering Trends

Adapting to software engineer trend is the natural extension of watching technology trends and predicting technology trends. Actually to a large extent, it depends on the first two steps to have a better understanding of the software engineering trends, although it has its own research methods to asses adoption costs, adoption benefits and adoption risks. How to make decision in adapting to software engineering trends is the focus of this paper, and will be discussed in more details in a later chapter.

4. Affecting Software Engineering Trends

This aspect of the project is interested in analyzing to what extent it is possible to affect/control technology trends. The following questions are discussed in this part:

- Is it possible to affect technology trends?

- Who can affect technology trends?
- How can technology trends be affected?
- At what phase of its evolutionary lifecycle can a technology trend be affected?
- How can we quantify impact?

Chapter 2 Adapting to Software Engineering Trends

2.1 Goal of Research

In chapter one, we discussed four broad classes of issues that will be addressed in the software engineering technology watch project. Specifically, they are:

- How to Watch Trends?
- How to Predict Trends?
- How to React/Adapt to Trends?
- How to Affect/ Influence Trends?

While predicting a trend may be the critical part of the research, adapting to a trend is most critical part in the whole project, since the final goal is not just knowing the trend, but rather knowing what to do with it, and this is the goal of this research.

So, what exactly is the meaning of adapting to a trend? It is possible that this question has many different interpretations, and here are some common ones that we are trying to answer :

- A corporate manager hears about a particular trend (e.g. Linux) and wants to know what to do about it: ignore it? Adapt the corporate products to support it? etc.

- An Academic curriculum developer hears about a particular trend and wants to know what to do about it: ignore it? Change class content to include it? Ensure students know about upon graduation? Etc.
- A government acquisition manager hears about a particular trend and wants to know what to do about it: ignore it? Encourage government contractor to adhere to it? etc.

To fix our ideas, in this paper, we will only consider the issue of adapting to a software-engineering trend from a corporate perspective, but the conclusion can easily be spread to any other perspective since they all share the same nature.

2.2 Analytical and Empirical Approach

Two approaches have been considered to this problem, which are analytical approach and empirical approach. Analytical approach views the adoption decision as a return on investment decision, and the adoption process as an investment. This approach is much easier to understand than empirical approach, which I will discuss later, since it has a resemblance to other current business decision-making process. From corporate managers' point of view, although many factors need to be considered before any adapting decision is made, such as the stake of certain trend for the organization and the intrinsic technical merits, most important issues they need to consider are cost, benefit and risk from adapting the trend. Usually a rational decision-maker will only adapt to the trend if adapting benefit exceeds cost, and ignore it otherwise.

To compare the adapting cost and benefit, we need to quantify each cost and benefit factors, which are organized and reported in structured table (see figure 2). This may be the bottleneck of the analytical model, since there is a lot of relevant cost and benefit

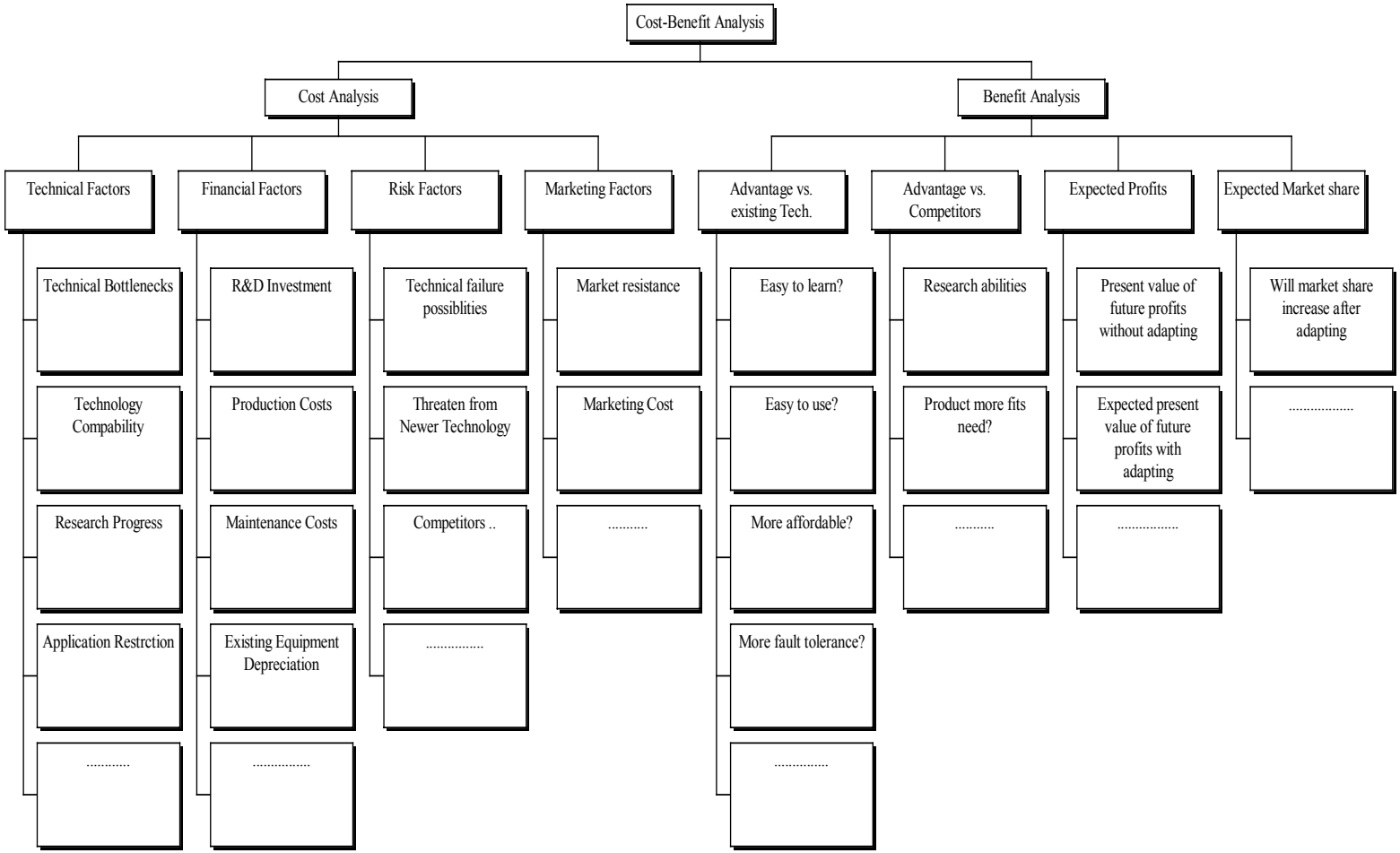
factors involved. For many of these factors, we have no other option but to consider discrete rating scales. Since in most cases we can not just add them up to get the total cost and total benefit, it is difficult to compare the cost and benefit. Thus, this approach limits itself to the theoretical stage.

Another approach is the empirical approach. This approach makes no effort to analyze or understand the precise economics of technology adoption, but attempts to derive relationships between relevant parameters of the technology (i.e. input) and the relevant parameters that help to make an adoption decision (i.e. output). To simulate this effect, we use neural network technology, which will be discussed in more details in the next chapter. Basically, we submit the historical data from past software engineering trends into the neural network tool, and let the tool build a neural network to discover the underlying relationship between the input data and output data. If this step is successfully accomplished, later we can just put the observed data from current trend into the network, and hope we can get the relevant outcome of an adoption decision.

2.3 A comparison of existing models.

Many Models have been done derived for this field. The main reason that the adapting process attracts so much attention is, enterprises worldwide have to operate in an environment of increasingly rapid change and increasingly strong competition. This is leading to major and continuing restructuring as the enterprises that struggle to survive by adapting to change. It is commonly agreed that "principled restructuring" or "adapting decision" is based on defining the basic goals of the enterprise, identifying the processes currently supporting those goals, reengineering the processes to serve the goals

Figure 2. Adapting to Software Engineering Trends



more effectively, and maintaining the quality of those processes by a program of continuous process improvement.

In addition to neural network, expert system is another popular approach in existing researches. There have been two major modeling methodologies developed in knowledge engineering: the KADS methodology[3] focusing on the derivation of the formal representation; and Checkland's[4] soft system methodology, which discusses basic modeling techniques in software engineering. These two methodologies are described briefly below.

The KADS methodology is the outcome of a number of ESPRIT project activities centered at the University of Amsterdam but involving researchers and practitioners from many institutions, countries and disciplines. KADS is intrinsically a modeling approach with seven types of models distinguished: 1) organizational model, which provides an analysis of the social-organizational environment in which the knowledge-based system will have to function. 2) Application model, which defines what problem the system should solve in the organization and what the function of the system will be in this organization. 3) Task model, which specifies how the function of the system, as specified in the application model, is achieved by defining tasks that the system will perform. 4) Cooperation model, which contains a specification of the functionality of those sub-tasks in the task model that require a cooperative effort between the agents to whom the sub-tasks have been distributed. 5) Expertise model, which is a central activity in the process of knowledge-based system construction. 6) Conceptual model, which includes abstract descriptions of the objects and operations that a system should know

about, formulated in such a way that they capture the intuitions that humans have of this behavior. 7) Design model, which specified separate design decisions.

Checkland[4] developed soft systems methodology in response to the failures of more conventional approaches to tackle problems that are hard to define, known as `soft' problems. Such `soft problems' are encountered frequently in organizations and cannot be solved by the same techniques that are used to solve `hard' problems. Soft systems methodology is a framework for system analysis that provides very powerful techniques for the analysis of systems with human and social components, and has been widely applied to difficult problem areas. There are seven stages of system analysis in soft systems methodology. The initial stages are concerned with system analysis and the later stages with system design.

The above two methodologies have been widely accepted and used in developing expert systems[5]. However, similar to the neural network approach, an expert system also has its own bottleneck, which is the knowledge acquisition and construction of expert systems. Experience in knowledge acquisition in an industrial setting shows that "it involves the gathering and management of large volumes of data from heterogeneous sources, and that this data gathering and management needs to become integrated with normal work processes if it is not to become such a burden as to undermine the knowledge acquisition activity[5]". Furthermore, "the knowledge engineer's job is to act as a go-between to help an expert build a system. Since the knowledge engineer has far less knowledge of the domain than the expert, however, communication problems impede the process of transferring expertise into a program. The vocabulary initially used by the expert to talk about the domain with a novice is often inadequate for problem-solving; thus the

knowledge engineer and the expert must work together to extend and refine it. One of the most difficult aspects of the knowledge engineer's task is helping the expert to structure the domain knowledge, to identify and formalize the domain concepts."[6]

Nevertheless, there is growing industrial interest in workflow and knowledge management tools that support existing processes and, incidentally, provide much of the data required for knowledge engineering.

Chapter 3. An Empirical Model and Neural Network Approach

3.1 A brief introduction to neural network

As we mentioned in Chapter two, the question we are trying to solve is to help company managers to decide whether and when to adapt to a software-engineering trend. It is not our goal to try to answer this question on behalf of managers. Instead, we are trying to provide enough information for them, so that they can make the decision correctly. We have chosen neural networks to help us accomplish this goal. So, why do we choose neural networks and how do they work? Before we get into our model, it is necessary to briefly introduce neural networks.

- **What is Neural Network?**

Unlike von Neumann machines, which are based on the processing/memory abstraction of human information processing, neural networks are based on the parallel architecture of animal brains. If we want to give a definition for neural network, or more precisely, Artificial Neural Networks (ANN). We can say that, ANN is an information processing paradigm that is inspired by the way biological systems, such as the brain, process

information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well.

- **How are neural networks used?**

Neural networks can be used in various ways. Typically, they are organized in layers. Layers can be made up of a number of interconnected "nodes" which contain an "activation function". Patterns are presented to the network via the "input layer", which communicates to one or more "hidden layers" where the actual processing is done via a system of weighted "connections". The hidden layers then link to an "output layer" where the answer is output.

Most Neural Networks contain some forms of "learning rules" which modify the weights of the connections according to the input patterns that it is presented with. There are many different kinds of learning rules used by neural networks, for example, the delta rule is often used.

Since the nature of the error space can not be known a priori, neural network analysis often requires a large number of individual runs to determine the best solution. Most learning rules have built-in mathematical terms to assist in this process which control the "speed" and the "momentum" of the learning. The speed of learning is actually the rate of convergence between the current solution and the global minimum. Momentum helps

the network to overcome obstacles (local minima) in the error surface and settle down at or near the global minimum.

Once a neural network is "trained" to a satisfactory level it may be used as an analytical tool on other data. To do this, the user no longer specifies any training runs and instead allows the network to work in forward propagation mode only. New inputs are presented to the input pattern where they filter into and are processed by the middle layers as though training were taking place, however, at this point the output is retained and no backpropagation occurs. The output of a forward propagation run is the predicted model for the data which can then be used for further analysis and interpretation.

Here we just use a simple layered feed-forward neural network (see figure 3) to illustrate this procedure.

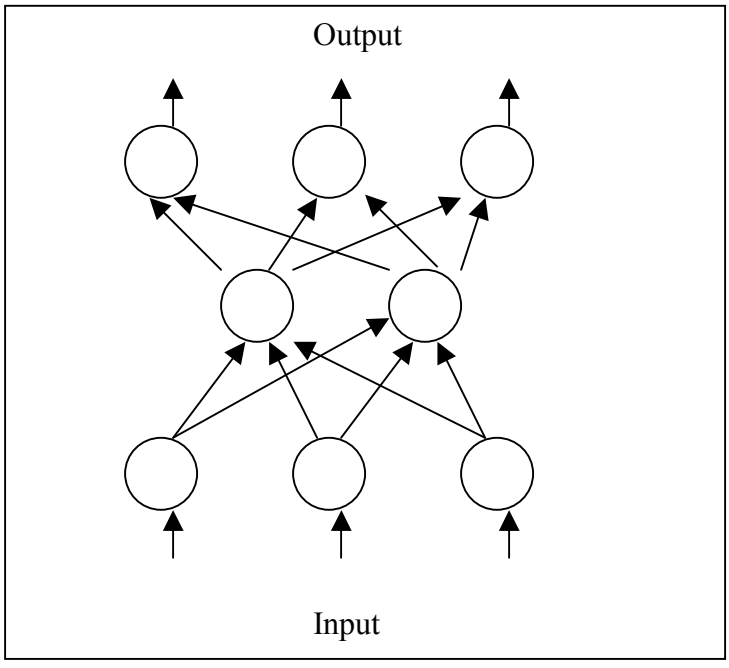


Figure 3. A typical neural network

As we can see from figure 3, a layered feed-forward neural network has layers, or subgroups of processing elements. A layer of processing elements makes independent

computations on data that it receives and passes the results to another layer. The next layer may in turn make its independent computations and pass on the results to yet another layer. Finally, a subgroup of one or more processing elements determines the output from the network.

Each processing element makes its computation based upon a weighted sum of its input. The first layer is the input layer and the last is the output layer. The layers that are placed between the first and the last layers are the hidden layers. In figure 3, only one hidden layer is shown, but sometimes, the number of hidden layer can be more than one.

The processing elements are seen as units that are similar to the neurons in a human brain, and hence, they are referred to as artificial neurons. Or they are simply referred to as neurons. Basically, the internal activation or raw output of a neuron in a neural network is a weighted sum of its inputs, but a threshold function is sometimes used to qualify the output of a neuron in the output layer. Synapses between neurons are referred to as connections, which are represented by edges of a directed graph in which the nodes are the neurons.

To construct a neural network, there are three aspects that we need to think about:

1) ***Structure***. Structure here refers to the architecture and topology of the neural network. This relates to how many layers the network should contain, and what their functions are, such as input, output, or feature extraction. Structure also encompasses how interconnections are made between neurons in the network and what their functions are. This is probably the most important aspect of neural network, since it usually decides the other two aspects.

2) ***Encoding***. Encoding is the method of changing weights. It refers to the paradigm used for the determination of and changing of weights on the connections between neurons. In the case of a multilayer feed-forward neural network, weights are initially defined randomly. Subsequently, in the process of training, if backpropagation algorithm is chosen, weights are updated starting from the output backwards. Once the training is finished, encoding is also finished since weights do not change after training is completed. However, in some other neural network, like the recurrent neural network that we are going to use for our empirical approach, encoding may be repeated again and again until a certain threshold is satisfied.

3) ***Recall.*** Recall is the method and capacity to retrieve information. It refers to getting an expected output for a given input. If the same input as before is presented to the network, the same corresponding output as before should result. The type of recall can characterize the network as being autoassociative or heteroassociative as we will mention in the next section.

- **Why neural networks ?**

The reason that we choose Neural Networks lies in their big advantages. Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either human or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions. Some other advantages include:

- Adaptive Learning: An ability to learn how to do tasks based on the data given for training or initial experience.
- Self-Organization: An ANN can create its own organization or representation of the information it receives during learning time.
- Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
- Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage.

Generally speaking, neural networks work best if the system used to model has high tolerance to error. Or in these two conditions: First, the problems are complex, and usually we can not devise a simple step-by-step algorithm or precise formula to generate an answer. Second, the data provided to resolve the problems is equally complex and may be noisy or incomplete.

In reality, Neural networks have been applied to a wide variety of areas. Most neural network applications, however, have been concentrated in the area of pattern recognition, where traditional algorithmic approaches have been ineffective. In this approach, the patterns can be represented by binary digits in the discrete cases, or real numbers representing analog signals in continuous case.

Pattern classification is a form establishing an autoassociation or heteroassociation. If we input a corrupted or modified pattern A to the neural network, and receive the true pattern A, this is termed autoassociation. In contrast, associating different patterns is

building the type of association called heteroassociation. For example, we associate A with B, give A, we can get B and vice versa.

In our research, we want to associate input data sets with output data sets. Autoassociation, then, is useful in recognizing or retrieving patterns with possibly incomplete information as input. What about heteroassociation? Ideally, we can store the input data sets of current pattern and retrieve the output data sets, which can be used by company managers to make decisions.

3.2 Recurrent Neural Network Architecture --- An empirical approach

Today the most popular neural network architecture is probably multilayer perceptron (MLP) architecture. The applications of MLP architecture to many different application areas have been very successful. It is nowadays commonly used as a mechanism for learning the input-output mapping of an underlying system, from input-output data alone. As long as the underlying input-output mapping is static, and the training data set is sufficiently large and representative of normal operation of the system under study, it is commonly believed that MLP is valid and simple to use.

MLP is a feed forward multilayered neural architecture, which is shown in figure 3. As we mentioned before, it consists of an input layer, which is assumed to have linear activation neuron characteristics; an output layer, and one or more hidden layers of neurons, which are neither input nor output neurons. The hidden layer neurons are assumed to be nonlinear. Common nonliearities include: sigmoid function, hyperbolic tangent function, radial basis function. These nonlinearities can be very general, satisfying some very general conditions. The main reason why MLP is popular is that it

has been shown, under very general assumption on the nonlinear activation functions, that this architecture is a universal approximator, for very general static nonlinear input output maps, provided that a sufficient number of hidden layer neurons is being used.

MLP architecture could have been used in our research. However, MLP model is more suitable for static models. In our case, the objects that we are studying are software-engineering trends, which basically consist of time series data sets. For these time series data, it is quite possible that the previous inputs and outputs may have influence on the current outputs, thus it is more suitable to use a dynamic model to analyze these data. Thus, recurrent neural networks (see figure 4), which takes into account any possible temporal correlation of the data, seem to be our best choice.

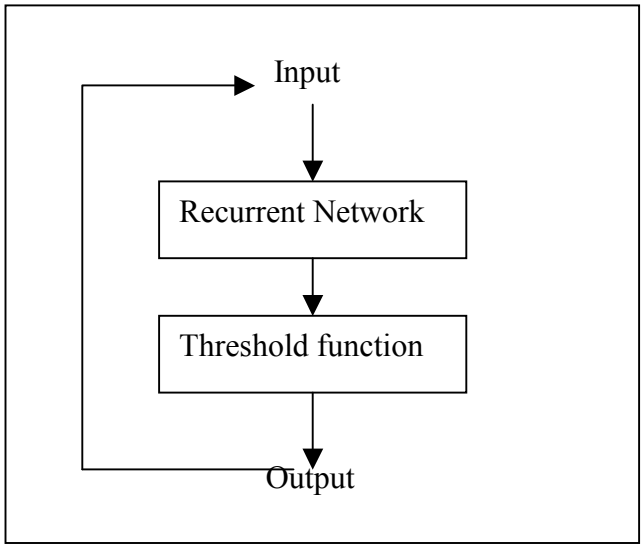


Figure 4. Architecture of Recurrent Neural Networks

We have chosen recurrent neural network as the structure. How about encoding then? In other words, how the weights will be adjusted in the recurrent neural network? Since a canonical form of the recurrent neural network can be derived easily based on the MLP, let's see how MLP adjusts its weights first.

In general, a MLP can be described by the following model:

$$y = f(c^T z + c_0) \quad (1)$$

$$z = F_n(bx + b_0) \quad (2)$$

Where:

z : n dimensional vector, denoting the outputs of the hidden layer neurons.

y: scalar variable, denoting the output of the output neurons.

x: m dimensional vector, denoting the inputs to the MLP.

$\mathbf{c_0}$: denotes the threshold of the output neurons.

$\mathbf{b_0}$: n dimensional vector, denotes the threshold of the hidden layer neurons.

c: n dimensional vector, denoting the weights connecting the input layer to hidden layer neurons.

f(.):denotes the nonlinear activation (weighted sum of its inputs) of the neurons. for example, $f(a) = (1+ e^{-a})^{-1}$ is a common sigmoidal activation function.

Usually, the output is allowed to have a range $-\infty < y(t) < \infty$, hence, it is more reasonable to modify (1) as follows:

$$y(t) = c^T z(t) + c_0 \quad (3)$$

i.e. the output neuron has a linear activation function.

While there is only one general MLP architecture, there are a number of alternative recurrent neural network architecture which have been proposed by various groups[7]. However, based on the MLP model, we can derive a canonical form of the recurrent neural network:

$$y(t) = c^T z(t) + c_0 \qquad (4)$$

$$z(t) = F_n(bv(t) + b_0) \qquad (5)$$

where y(t) and v(t) are respectively the scalar output and m+d dimensional input vector to the multilayer perceptron. z(t) denotes the concatenation of the outputs of the n hidden layer neurons into one vector. b and c are respectively matrix and vector of appropriate dimensions. b_0 and c_0 denote respectively the thresholds of the hidden layer neurons and the output neuron. v(t) is a concatenation of the m input x(t) and the feedback signals from a feedback signals from a feedback block of the following form:

$$\zeta(t) = \begin{pmatrix} y(t-1) \\ y(t-2) \\ \ldots. \\ y(t-d) \end{pmatrix}$$

$$\text{i.e., } v(t) = \begin{pmatrix} x(t) \\ \\ \zeta(t) \end{pmatrix}$$

Chapter 4 Data Collection

4.1 Pattern Recognition and Data needed.

In general, there are two sets of data we need to recognize the pattern of software engineering trend: input data and output data. Input data are referred to the data that can be observed from historical or current software engineering trends, while output data is the information that is crucial and helpful for decision makers to make decisions.

Considering the fact that the software engineering trends we are going to deal with are always changing over time, we realize that the factors that are relevant to the trends vary according to its life cycle. This means that both the input factors and the output factors should also change for different phases. For example, in the research phase, when we try to collect input data, we do not have to worry about the market factor. It is same for the output data in this phase, since people are more interested in knowing technology bottlenecks when they make adopting decision, rather than the future product's market risk level, market factors should not be included either.

Then, how can we adjust the input and output data sets according to time? In chapter one, I briefly discussed the generic evolutionary model, which includes three phases: research phase, technology phase and market phase. This model is not only the basis of predicting software engineering trends, but also used here as a direction for pattern recognition, since these three phases basically illustrate the life cycle of the software engineering trends. Based on this model, we can assume that the patterns to be recognized in each phase may be different, and the input and output data sets for each phase should not be

exactly the same. On the other hand, we also realize that the software engineering trends are continuous. This means that there are no distinct gaps between each phase. And the input data sets and output data sets are not totally different either.

To select the exact input and output data, first thing we need to consider is which factors are most relevant to the software engineering trends themselves. These factors should then be grouped into each phase according to the maturity of the major technology in the software engineering trends. Here is the list of factors that we are most interested[1].

- Research Phase.
  - Intrinsic Technical Merit. Amount of Support (Government, Industry). Amount of Acceptance/Interest (Academia, Research Labs, Standards Bodies).
  - Potential Applicability. Scope of Application. Cost of Application. Criticality/Impact of Application.
  - Potential Risks/Hurdles. Threats from Competing Solutions. Potential bottlenecks.
- Technology Phase.
  - Intrinsic Technical Merit. Amount of Support (Government, Industry). Amount of Acceptance/Interest (Academia, Research Labs, Standards Bodies).
  - Potential Applicability. Scope of Application. Cost of Application. Criticality/Impact of Application.
  - Potential Risks/Hurdles. Threats from Competing Solutions. Potential bottlenecks.

    - Market outlook. Actual Market, measured by some quantification of: Innovators; Early adopters; Early majority. Potential Market. Untapped market.

- Market Phase.
    - Potential Applicability. Scope of Application. Cost of Application. Criticality/Impact of Application.
    - Potential Risks/Hurdles. Threats from Competing Solutions. Potential bottlenecks.
    - Market outlook. Actual Market measured by some quantification of: Innovators; Early adopters; Early majority. Potential Market. Untapped market.

Our challenge now is to quantify these factors in a meaningful way, so that we can use them to build neuron network to serve our purpose. This will be discussed next.

4.2 Data format.

It seems a neural network is a relatively easy approach since most of the analytical job can be done with neural network tools. Researchers using neural networks nowadays do not even bother to write their own programs. However, this is not the case in many fields, especially in the software engineering area. The potential serious problems often come from the data collection. We know the neural network is typically proposed for domains that contain large data sets. But many domains lack such large data sets, which is also true in our research. Some of the data we mentioned before is either unavailable or

incomplete. This lack of data is so acute that building a neural network could be a mission impossible.

The solution to this data-lacking problem is to rely heavily on the true-false questions and range parameters. As we can see from previous part of this paper, there are two kinds of data we really want. One is the statistical data from the real world, such as adoption cost, amount of support, etc. This kind of data is hard to collect due to the fact that the financial information is often ranked as top secret in the private sectors. Even if collecting these data is theoretically possible, the collection cost can be very high.

Thus, we try to use true and false questions to quantify them. To a large extent, this method is applicable. For example, we assume ample and stable research funding will have positive effects on the software engineering trends. We don't have to know the exactly amount of funding, although that will be a perfect solution, we just need to know the source of funding. If the funding comes from government, we assume it usually will be stable and sufficient. So we just ask true-false question like " Is the research sponsor affiliated to government? ". And later when we put data into the neural network tools, we will use 1 for yes, and -1 for no.

The other kind of data is data that needs to be estimated, such as risk level, social acceptance level of certain trends. To get this kind of data, we will ask questions and limit the answer to a certain range, say 0-10. Thanks to the nature of neural network, we don't have to answer the question very accurately. For questions that ask estimating level or range, sometimes it is impossible to tell the difference between level 5 or level 6, but it is quite possible that we can tell the difference between level 3 and level 7. This is

usually good enough since we are dealing with a large set of data. The neural network is good at recognizing patterns from noisy and incomplete data.

To be specific, we use the following list of questions to get input and output data:

- **Input data:**
  - **Research Phase**

    Intrinsic Technical Merit.

    - Is the research sponsor affiliated to government?
    - Do other researchers hold positive view about this research?
    - What is the social awareness level of this new technology?

    Potential Applicability

    - Is this research widely applicable?
    - Will the research sponsor share this new technology free (or at minimal cost)?
    - Will this research contribute to the current trend greatly?

    Potential risks/hurdles

    - Is there any similar research as this one?
    - What is the level of competition between researches?
    - What is the level of potential bottleneck?

  - **Technology Phase**

    Intrinsic Technical Merit.

    - Is the research sponsor affiliated to government?
    - Do other researchers hold positive view about this research?
    - What is the social awareness level of this new technology?

Potential Applicability

- Is this research widely applicable?
- Will the research sponsor share this new technology free (or at minimal cost)?
- Will this research contribute to the current trend greatly?

Potential risks/hurdles

- Is there any similar research as this one?
- What is the level of competition between researches?
- What is the level of potential bottleneck?

Market Outlook

- Are the innovators of this technology eager to commercialize it?
- Are the early adopters of this technology well funded?
- Does early majority of this technology have enough influence to direct the market?
- Is there a large (compared to current market size) potential market existing?
- Is there any untapped market?

- **Market Phase**

Potential Applicability

- Is this research widely applicable?
- Will the research sponsor share this new technology free (or at minimal cost)?
- Will this research contribute to the current trend greatly?

Potential risks/hurdles

- Is there any similar research as this one?
- What is the level of competition between researches?
- What is the level of potential bottleneck?

Market Outlook

- Are the innovators of this technology eager to commercialize it?
- Are the early adopters of this technology well funded?
- Does early majority of this technology have enough influence to direct the market?
- Is there a large (compared to current market size) potential market existing?
- Is there any untapped market?

- **Output data:**
  - **Research phase:**

Time aspect:

- Expected period before the idea turning into product.

Applicability aspect:

- Expected influence level of this trend on current technologies.
- Level of support available.

Risk aspect:

- Expected risk level of research failure
- Expected level of remaining technical bottleneck.
- Estimated adoption cost level.

- **Technology Phase:**

Time aspect:

- Expected period before the adapter can make profits, if adapting decision is made.

Applicability aspect:

- Expected influence level of this trend on current technologies.
- Level of support available.

Risk aspect:

- Expected level of social resistance.
- Expected level of remaining technical bottleneck.
- Estimated adoption cost level.

Market aspect:

- Estimated (potential) market size.
- Estimated market share.
- Level of market loss if not involved.
- Estimated level of competition among early adopters.

- **Market phase:**

Time aspect:

- Expected life span of the trend. (How long will it last?)

Applicability aspect:

- Expected influence level of this trend on current technologies.

Market aspect:

- Estimated (potential) market size.
- Estimated market share.
- Level of market loss if not involved.
- Estimated level of competition among early adopters.

Based on these questions, we collect time series data for the following 6 software engineering trends: ADA (1979-2000), UNIX (1969-2000), JAVA(1991 – 2000), CORBA (1992-2000), XML(1996-1998) and LINUX(1991-2000). The data can be found in Appendix.

4.3 Collection Procedures and Data Source

To collect the above data, first we need to get familiar with the selected software engineering trends. And then try to answer the questions with our knowledge. In most cases, we can not answer all the questions using single source. We have to search as much as possible, so that knowledge we have will be sufficient enough to answer the questions without too much bias. If there is discrepancy about a certain question among different resources, we will find the mainstream opinion and use that as the base to get the necessary data. Some of the sources of information are listed below:

- **Industry**. Industry is often the sponsor and consumer of the software engineering trends. Large industrial corporations can affect trends by funding research, adopting early product and developing industry standard. Since it is the front end of the technology transfer lifecycle, industry itself also pays a lot of attention to the software engineering trends. Many companies have their own technology watch agent. There are also companies that are dedicated to collecting and analyzing such

information, such as IDC. Thus industry is in a good position to have a thorough understanding of technology trends, which makes it a good starting point for us too. Thanks to the fast growing of the Internet, a lot of information from industry can be found on the web. So it is relatively easy to search for data for modern trends, usually within the past 10 years.

- **Government**. Like industry, government is also the major sponsor and consumer of technology. Compared to industry, government is more likely to sponsor long-term research. And we have noticed the fact that research funding that comes from government tends to be more stable than other sources. Government funding agencies, such as DARPA, NASA and NSF usually keep records of technology information, and thus is another major data source for us, when this data is available to public.
- **Academia**. Academia plays an important role in software engineering trends. It is often the active participants of the technology innovation. Discussions from academia also provide us a general outlook of the software engineering trend and intensive knowledge of the technology.
- **Panel Sessions**. Panel sessions in major software engineering conferences have been a favorite forum for discussion of research trends and technology trends in software engineering. While this may be somewhat overlapping with previous sources, we think proceedings from these conferences are usually more concentrated and focused on advanced issues of software engineering trends. So we single it out as an independent data source.

- **Journals and other Media**. Although sometime not as convenient when they are not available on the Internet, Journals and other media's role can not be replaced by others. In many cases, Journals from outside technical and industrial areas may reflect the social opinion about software engineering trends. And sometimes, they can have effect on the software engineering trends too. Although this kind of effect mostly restricted to the market side. Therefore, we do include them in our data sources and hope to know the society opinion of software engineering trends and how those trends affect the real world.

Chapter 5 Analytical Result

5.1 Performance of Neural Network on Sample Data

As we mentioned in chapter 4, for each different phase, we need to consider different input and output. Thus, different patterns should be recognized for all three phases. Based on the time series data that we collected from following five software engineering trends: ADA, UNIX, JAVA, CORBA and XML (training data), we built three recurrent neural networks. And then, we used the data that we collected from LINUX(testing data) to test the performance of these three neural networks. The results are discussed below.

- ***Performance of Neural Network for Research Trends***

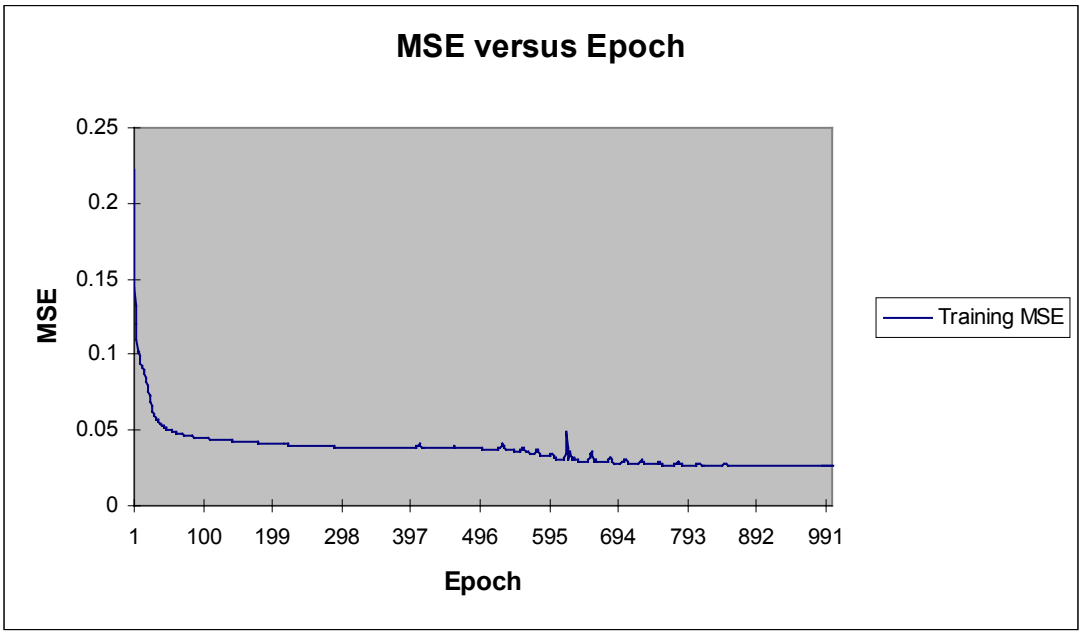


Figure 5. MSE of Neural Network for Research Trends

Figure 5 shows the mean square error(MSE) for the neural network built based the training data of research trends. As we can see from the figure, the MSE keeps decreasing and approaching zero over the 1000 epochs. This tells us that the neural network is able to recognize the pattern hiding in the data.

The testing result of research trends is as follows:

Performance	*Expected period before the idea turning into product*	*Expected influence level of this trend on current technologies*	*Level of support available*	*Expected risk level of research failure*	*Expected level of remaining technical bottleneck*	*Estimated adoption cost level*
MSE	0.742047128	4.298648819	6.970812067	5.833274986	8.521306207	2.681751058
NMSE	1.686470743	5.656116805	6.702703619	3.645796839	5.917573667	4.396313278
MAE	0.662092182	1.88954939	2.453564353	1.948390395	2.408939591	1.314169287
Min Abs Error	0.035133362	0.63788271	0.033361912	0.210062981	0.149491787	0.138772011
Max Abs Error	1.562837005	3.118252933	4.146792829	4.806399822	5.453811646	2.524163246
r	-0.099383777	0.955988283	-0.873796821	0.508944248	-0.825478041	-0.487111775

Table 1. Testing result of research trends.

In the above table, ***MSE*** stands for the means square error, ***NMSE*** stands for the normalized mean square error, ***MAE*** is the mean absolute error, ***Min abs Error*** is the minimum absolute error, ***Max Abs Error*** is the maximum absolute error; ***r*** is linear correlation coefficient.

Since r^2 is the estimation of R^2, which is the coefficient of determination, and always used to determine the quality of certain variable fitting into the model. From table 1, we can see that, although the overall performance of this model is not very satisfied, it does have certain explanation power, especially for some variables, such as expected influence level and level of support available. Also all the MAEs are less than 2.5, which means when we use the model for prediction, the expected predication error is within a reasonable range.

- ***Performance of Neural Network for Technology Trends***

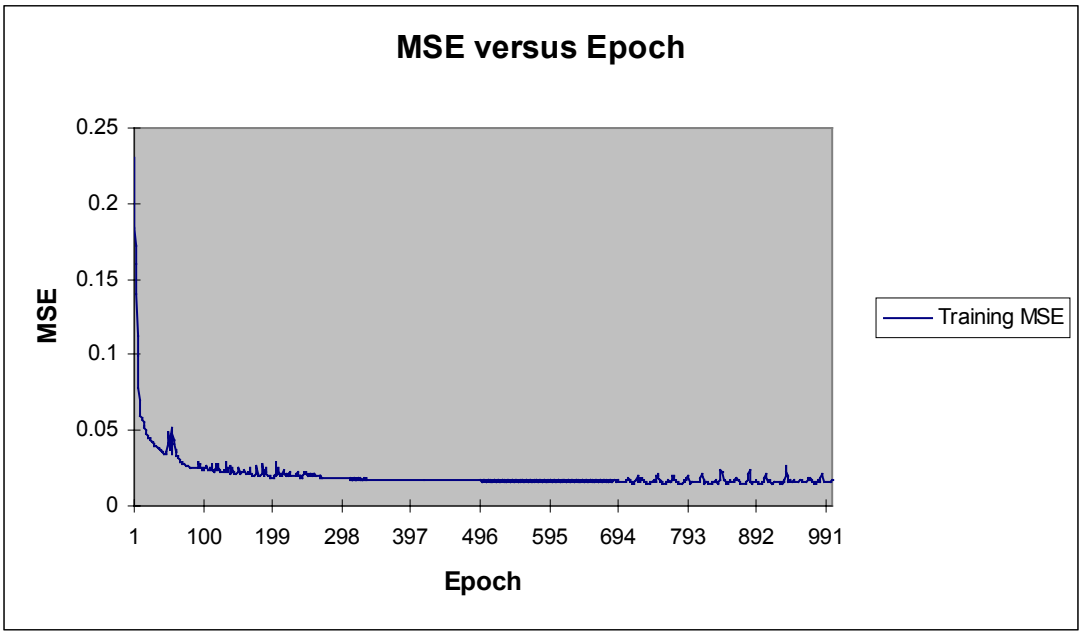


Figure 6. MSE of Neural Network for Technology Trends

Figure 6 shows the MSE for the neural network build based on the training data of the technology trends. As we can see from the figure, although the MSE curve is not very smooth, it still keeps decreasing and approaching zero over the 1000 epochs, which is the sign of the neural network being able to recognize the pattern hiding in the data.

The testing results of technology trends are as follows:

Performance	Expected period before adapter can make profits	Expected influence level of this trend on current technologies	level of support available	Expected level of social resisitance
MSE	1.398498282	3.401297362	6.291099643	4.376043047
NMSE	1.344709828	4.475391216	6.049134009	1.910935741
MAE	0.680004919	1.730153116	2.435174921	2.088503359
Min Abs Error	0.004837096	0.151595592	0.067912817	0.936580181
Max Abs Error	2.982817173	2.976890326	3.329780579	2.476973534
r	0.023434425	0.767658908	-0.839757089	-0.004602783

Performance	Expected level of remaining technical bottleneck	Estimated adoption cost level	Level of market loss if not involved	Estimated level of competition among early adopters
MSE	6.268497709	1.649868595	1.856820564	7.643253921
NMSE	4.353123344	2.704702658	1.03156698	1.06749358
MAE	2.077779627	1.111357245	1.148012475	2.650815924
Min Abs Error	0.387123108	0.158650398	0.026091576	1.089980602
Max Abs Error	4.796465635	1.655030251	2.099197388	3.831763744
r	-0.69295523	0.652160029	0.431600086	0.762072358

Table 2. Testing result of technology trends.

Again, the overall performance of this model on the testing data is not quite satisfied, but it does have some explanation power on certain variables. And most of the MAEs of all the variables are less than 2.5, which means when we use the model for prediction, the expected predication error is within a reasonable range.

- ***Performance of Neural Network for Market Trends***

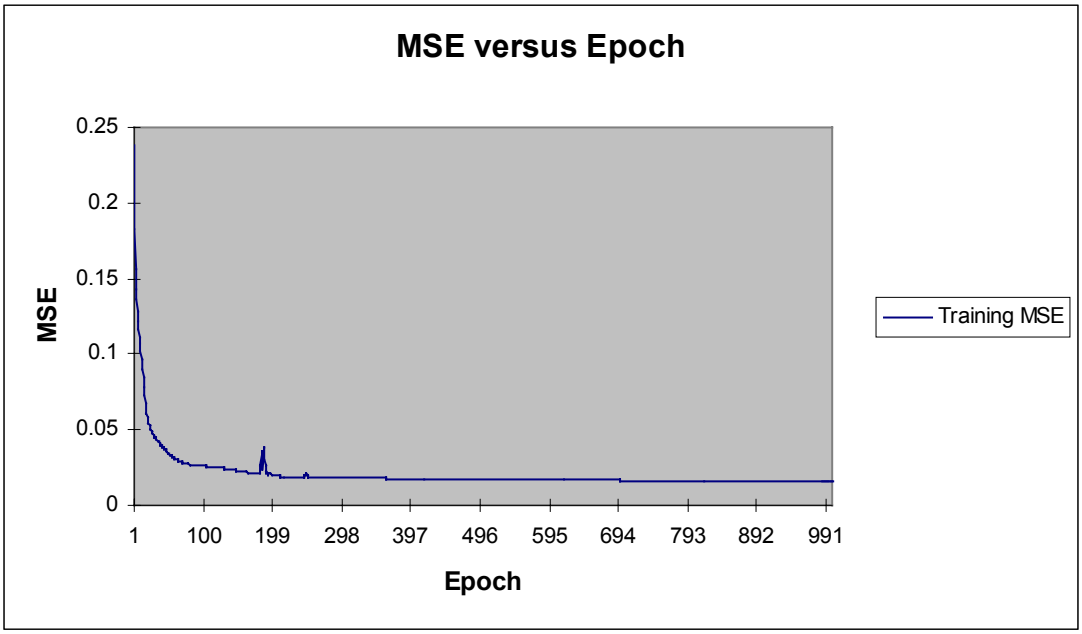


Figure 7. MSE of Neural Network for Market Trend

Figure 7 shows the MSE for the neural network built based on the training data of market trends. As we can see from the figure, the MSE curve is very smooth in this model, it keeps decreasing and approaching zero over the 1000 epochs, which is the sign of the neural network being able to recognize the pattern hiding in the data.

The testing result of market trends is as follows:

Performance	*Expected life span of the trend*	*Expected influence level of this trend on current technologies*	*Level of market loss if not involved*	*Estimated level of competition among early adopters*
MSE	27.70426046	5.778870372	0.687901049	7.913403223
NMSE	#DIV/0!	7.603776722	0.38216725	1.10522393
MAE	4.811357212	2.096890087	0.436343896	2.311358534
Min Abs Error	1.135578156	0.300107002	0.042851925	0.028770566
Max Abs Error	6.771296501	2.190961838	0.911922932	4.22270298
r	#DIV/0!	0.768014399	0.979392206	0.80973709

Table3. Testing result of market trends.

This time the overall performance of neural network on the testing data is quite satisfied. But it seems to have no explanation power on expected life span of the trend, which is indicated by the error signs. The reason may lie in the fact that we don't have a relatively precise way to estimate the life span for all the software-engineering trends that we observed.

5.2 Preliminary Conclusions

The results shown in last section shows that our neural network approach does recognize patterns from input and output data. To further prove this pattern does exist, we use the data from Linux to compare the desired output, which we observed from the history, and the estimated output, which we calculated based on the pattern we derived. And here are the results.

	Expected life span of the trend	Expected influence level of this trend on current technologies	Level of market loss if not involved	Estimated level of competition among early adopters	Expected life span of the trend	Expected influence level of this trend on current technologies	Level of market loss if not involved	Estimated level of competition among early adopters
	Desired	Desired	Desired	Desired	Esti.	Esti.	Esti.	Esti.
LINUX91	20	4	1	1	10	3	1	1
LINUX92	20	4	1	1	10	3	1	1
LINUX93	20	4	1	1	10	3	1	1
LINUX94	20	5	3	1	15	5	3	3
LINUX95	20	5	4	5	14	4	3	4
LINUX96	20	6	4	6	15	5	5	5
LINUX97	20	6	4	6	13	6	4	6
LINUX98	20	6	4	7	13	5	4	5
LINUX99	20	6	4	7	14	5	5	6
LINUX00	20	6	4	7	14	6	5	6

Table4. Comparing result of Linux.

As we can see, the performance of the Neural Network on sample data shows that it is an acceptable approach to analyze the noise data generated from the software engineering trends, and it can be used to predict the desired output within a reasonable range. Thus, if we are to estimate certain information about software engineering trends given a set of input data, such as Java, CORBA or Linux , we can just calculate the output data based on the pattern we have derived (see Table 5) , and we are sure that it is much better than randomized.

	Expected life span of the trend	Expected influence level of this trend on current technologies	Level of market loss if not involved	Estimated level of competition among early adopters	Adoption Decision
Java01	17	8	7	6	Yes?
CORBA01	13	4	4	3	No?
LINUX01	14	6	5	6	Yes?

Table5. Estimated Results of Java, Corba and Linux.

Now the question is, how can these output data help the decision makers to make adopting decision?

In chapter 4, we mentioned that the output data sets were chosen according to their degree of relevance to the software engineering trends. We believe that these chosen variables provide the most important information that decision-makers need to know before they make decisions.

Generally speaking, all data can be divided into two categories. One is a positive factor, this kind of data refers to variables that are helpful to reduce the adoption risk. Since most output data are represented using category levels, the higher the level of positive

factors, the less the adoption risk involved, thus it is more likely the decision makers should adopt the trend. For example, in the output data set for research trend, support level and influence level of software engineering trends are positive factors, since we know the more support and influence the technology has, the less risk involved when we decide to adopt to this decision.

The other category is a negative factor. In contrast to a positive factor, this kind of data refers to variables that have negative relationship with adopting decisions, which means they will increase the adoption risk. For negative data, the higher the level, the greater the risk to adopt the trend. Examples of negative factor are level of research bottleneck and social resistance.

When decision-makers make an adopting decision, they need to consider both the positive side and negative side factors. While it is their responsibility to make the final decision, it is our task to provide them criterion on how to balance these two sides. One thing we did not mention in chapter 4 is that, when we collect the output data, we have been using the level 5 as the break point for both positive and negative factors. That is, if the level of certain variable is not good/bad enough to adopt/ignore trends, we use value below 5 to represent the variable. On the contrary, any level value above 5 is significant enough to influence the trends, while 5 stands for undetermined. Thus, for positive factors, such as influence level of the technology, if the value is greater than 5, we will think it is significant enough to suggest to the decision-maker to adopt the trend from that variable's perspective. As to negative factors, such as expected risk level of research failure, if the value is greater than 5, we will think that from the standpoint of that variable, it is too risky to adapt to the certain trend. Overall, if the number of significant

negative variables exceeds the number of positive variables, the possibility of adoption failure will be greater than that of adoption success.

Chapter 6 Summary

In this paper, we discussed the adapting software engineering trend problem faced by many decision makers, and its role in the whole project of software engineering technology watch. We also introduced the neural network approach and the reason we want to apply it in this field. In addition, most important, we have attempted to do two things:

- *Quantify the historical data of software engineering trends in terms of input and output data*. This establishes the basis of our neural network approach, and makes the complex problem that we are trying to solve becomes more understandable.
- *Build a recurrent neural network based on the historical data we collected*. This approach tried to analyze the noise data and develop a pattern with the help of a neural network.

The results of this paper are encouraging but I am not yet satisfied. It is encouraging in the sense that the neural network we built, based on the historical data, is able to recognize patterns from noise data, and for some variables that we observed, the prediction outputs seem to be statistically significant. I am not satisfied because there are also some other variables that are statistically insignificant. Also, we need more justified ways to utilize the output data to help decision makers to make decisions.

Overall, our conclusion is neural networks is an acceptable approach to analyze the noise data generated from the software engineering trends and can be used to predict the desired output within a reasonable range. We can use the pattern we derived to estimate certain information needed to know before we make adapting decision, and we are sure that is kind of information can be relied on.

In the future, more efforts should be put in the following aspects:

- Reexamine the input and output data. Some output data we listed in this paper contains the information that decision makers need to know to make decisions. However, it has been shown statistically insignificant and not fitting in the model, which means either it can not be explained by the input data, or there is more suitable way to represent it. Due to the time constraint, this job has not been finished to my satisfaction.
- Collect more information about software engineering trends. In this paper, we used the data we collected from six software engineer trends. Obviously, the more data we have, the more robust will be our model.
- Consider other machine learning approachs. Neural networks are very powerful tool to analyze noise and incomplete data. It could become more powerful if we combined with other machine learning approaches like fuzzy logic on this issue.

Appendix

Table 1 Input Data for Research Trends

	Is research sponsor affilated to government?	Do other researchers hold positive view?	What is social awareness level?	Is this research widely applicable?	Is this new technology free?	Will this trend contribute to the current trend?	Is there any similar research?	What is the level of competition between researches?	What is the level of potential bottleneck?
Ada79	1	1	3	1	1	-1	1	8	4
Ada80-85	1	1	8	1	1	-1	1	7	4
Ada86-90	1	1	9	1	1	-1	-1	3	5
Ada91-95	1	-1	8	1	1	-1	1	8	5
Ada96-00	1	-1	6	1	1	-1	1	4	0
Unix 69	-1	-1	1	-1	1	-1	-1	0	7
Unix70-75	-1	1	8	1	1	-1	-1	2	5
Unix76-80	-1	1	9	1	1	-1	1	8	5
Unix81-85	-1	1	10	1	1	-1	1	9	6
Unix86-90	-1	1	10	1	1	-1	1	10	6
Unix91-95	-1	1	10	1	1	-1	1	10	3
Unix96-00	-1	1	10	1	1	-1	1	8	5
Java91	-1	1	0	-1	-1	-1	-1	0	0
Java92	-1	-1	0	-1	1	-1	-1	1	9
Java93	-1	-1	0	-1	1	-1	-1	1	8
Java94	-1	-1	0	1	1	-1	-1	1	8
Java95	-1	1	8	1	1	1	1	3	4
Java96	-1	1	9	1	1	1	1	4	3
Java97	-1	1	10	1	1	1	1	4	3
Java98	-1	1	10	1	1	1	1	4	3
Java99	-1	1	10	1	1	1	1	4	3
Java00	-1	1	10	1	1	1	1	4	3
CORBA92	-1	1	5	1	1	1	1	4	8
CORBA93	-1	1	4	1	1	1	1	4	7
CORBA94	-1	1	6	1	1	1	1	5	7
CORBA95	-1	1	6	1	1	1	1	6	7
CORBA96	-1	1	7	1	1	1	1	7	6
CORBA97	-1	1	8	1	1	1	1	8	4
CORBA98	-1	1	8	1	1	1	1	8	4

CORBA99	-1	1	8	1	1	1	1	8	4
CORBA00	-1	1	8	1	1	1	1	8	4
XML96	-1	1	7	1	1	1	1	5	4
XML97	-1	1	8	1	1	1	1	4	3
XML98	-1	1	9	1	1	1	1	4	3
LINUX91	-1	-1	1	1	1	1	1	3	7
LINUX92	-1	-1	1	1	1	1	1	3	6
LINUX93	-1	-1	1	1	1	1	1	3	6
LINUX94	-1	1	5	1	1	1	1	6	6
LINUX95	-1	1	6	1	1	1	1	7	6
LINUX96	-1	1	7	1	1	1	1	7	4
LINUX97	-1	1	8	1	1	1	1	8	4
LINUX98	-1	1	9	1	1	1	1	8	4
LINUX99	-1	1	9	1	1	1	1	8	3
LINUX00	-1	1	9	1	1	1	1	8	3

Table 2 Input Data for Technology Trends

	Is research sponsor affilated to government?	Do other research hold positive view?	What is social awareness level?	Is this research widely applicable?	Is this new technology free?	Will this trend contribute to the current trend?	Is there any similar research?	What is the level of competition between researches?	What is the level of potential bottleneck?	Are the innovators of this technology eager to commercialize it?	Are the early adopters well funded?	Does early majority have enough fluence to direct the market?	Is there a large potential market existing?	Is there any untapped market?
Ada79	1	1	3	1	1	-1	1	8	4	-1	1	-1	-1	1
Ada80-85	1	1	8	1	1	-1	1	7	4	-1	1	-1	1	1
Ada86-90	1	1	9	1	1	-1	-1	3	5	1	1	-1	1	1
Ada91-95	1	-1	8	1	1	-1	1	8	5	1	1	-1	1	1
Ada96-00	1	-1	6	1	1	-1	1	4	0	1	1	-1	1	1
Unix 69	-1	-1	1	-1	1	-1	-1	0	7	-1	-1	1	1	1
Unix70-75	-1	1	8	1	1	-1	-1	2	5	-1	-1	1	1	1
Unix76-80	-1	1	9	1	1	-1	1	8	5	-1	-1	1	1	1
Unix81-85	-1	1	10	1	1	-1	1	9	6	-1	1	1	1	1
Unix86-90	-1	1	10	1	1	-1	1	10	6	1	1	1	1	1

Unix91-96	-1	1	10	1	1	-1	1	10	3	1	1	1	1	1
Unix96-00	-1	1	10	1	1	-1	1	8	5	1	1	1	1	1
Java91	-1	1	0	-1	1	-1	-1	0	0	1	-1	-1	-1	-1
Java92	-1	-1	0	-1	1	-1	-1	1	9	1	-1	-1	-1	-1
Java93	-1	-1	0	-1	1	-1	-1	1	8	1	-1	-1	-1	-1
Java94	-1	-1	0	1	1	-1	-1	1	8	1	1	-1	1	1
Java95	-1	1	8	1	1	1	1	3	4	1	1	1	1	1
Java96	-1	1	9	1	1	1	1	4	3	1	1	1	1	1
Java97	-1	1	10	1	1	1	1	4	3	1	1	1	1	1
Java98	-1	1	10	1	1	1	1	4	3	1	1	1	1	1
Java99	-1	1	10	1	1	1	1	4	3	1	1	1	1	1
Java00	-1	1	10	1	1	1	1	4	3	1	1	1	1	1
CORBA92	-1	1	5	1	1	1	1	4	8	1	1	-1	1	1
CORBA93	-1	1	4	1	1	1	1	4	7	1	1	-1	1	1
CORBA94	-1	1	6	1	1	1	1	5	7	1	1	-1	1	1
CORBA95	-1	1	6	1	1	1	1	6	7	1	1	-1	1	1
CORBA96	-1	1	7	1	1	1	1	7	6	1	1	-1	1	1
CORBA97	-1	1	8	1	1	1	1	8	4	1	1	-1	1	1
CORBA98	-1	1	8	1	1	1	1	8	4	1	1	-1	1	1
CORBA99	-1	1	8	1	1	1	1	8	4	1	1	-1	1	1
CORBA00	-1	1	8	1	1	1	1	8	4	1	1	-1	1	1
XML96	-1	1	7	1	1	1	1	5	4	1	1	1	1	1
XML97	-1	1	8	1	1	1	1	4	3	1	1	1	1	1
XML98	-1	1	9	1	1	1	1	4	3	1	1	1	1	1
LINUX91	-1	-1	1	1	1	1	1	3	7	1	-1	-1	1	1
LINUX92	-1	-1	1	1	1	1	1	3	6	1	-1	-1	1	1
LINUX93	-1	-1	1	1	1	1	1	3	6	1	-1	-1	1	1
LINUX94	-1	1	5	1	1	1	1	6	6	1	1	-1	1	1
LINUX95	-1	1	6	1	1	1	1	7	6	1	1	-1	1	1
LINUX96	-1	1	7	1	1	1	1	7	4	1	1	-1	1	1
LINUX97	-1	1	8	1	1	1	1	8	4	1	1	-1	1	1
LINUX98	-1	1	9	1	1	1	1	8	4	1	1	-1	1	1
LINUX99	-1	1	9	1	1	1	1	8	3	1	1	-1	1	1
LINUX00	-1	1	9	1	1	1	1	8	3	1	1	-1	1	1

Table 3 Input Data for Market Trends

	Is this research widely applicable?	Is this new technology free?	Will this trend contribute to the current trend?	Is there any similar research?	What is the level of competition between researches?	What is the level of potential bottleneck?	Are the innovators of this technology eager to commercialize it?	Are the early adopters well funded?	Does early majority have enough fluence to direct the market?	Is there a large potential market existing?	Is there any untapped market?
Ada79	1	1	-1	1	8	4	-1	1	-1	-1	1
Ada80-85	1	1	-1	1	7	4	-1	1	-1	1	1
Ada86-90	1	1	-1	-1	3	5	1	1	-1	1	1
Ada91-95	1	1	-1	1	8	5	1	1	-1	1	1
Ada96-00	1	1	-1	1	4	0	1	1	-1	1	1
Unix 69	-1	1	-1	-1	0	7	-1	-1	1	1	1
Unix70-75	1	1	-1	-1	2	5	-1	-1	1	1	1
Unix76-80	1	1	-1	1	8	5	-1	-1	1	1	1
Unix81-85	1	1	-1	1	9	6	-1	1	1	1	1
Unix86-90	1	1	-1	1	10	6	1	1	1	1	1
Unix91-96	1	1	-1	1	10	3	1	1	1	1	1
Unix96-00	1	1	-1	1	8	5	1	1	1	1	1
Java91	-1	-1	-1	-1	0	0	1	-1	-1	-1	-1
Java92	-1	1	-1	-1	1	9	1	-1	-1	-1	-1
Java93	-1	1	-1	-1	1	8	1	-1	-1	-1	-1
Java94	1	1	-1	-1	1	8	1	1	-1	1	1
Java95	1	1	1	1	3	4	1	1	1	1	1
Java96	1	1	1	1	4	3	1	1	1	1	1
Java97	1	1	1	1	4	3	1	1	1	1	1
Java98	1	1	1	1	4	3	1	1	1	1	1
Java99	1	1	1	1	4	3	1	1	1	1	1
Java00	1	1	1	1	4	3	1	1	1	1	1
CORBA92	1	1	1	1	4	8	1	1	-1	1	1
CORBA93	1	1	1	1	4	7	1	1	-1	1	1
CORBA94	1	1	1	1	5	7	1	1	-1	1	1
CORBA95	1	1	1	1	6	7	1	1	-1	1	1
CORBA96	1	1	1	1	7	6	1	1	-1	1	1
CORBA97	1	1	1	1	8	4	1	1	-1	1	1
CORBA98	1	1	1	1	8	4	1	1	-1	1	1
CORBA99	1	1	1	1	8	4	1	1	-1	1	1

CORBA00	1	1	1	1	8	4	1	1	-1	1	1
XML96	1	1	1	1	5	4	1	1	1	1	1
XML97	1	1	1	1	4	3	1	1	1	1	1
XML98	1	1	1	1	4	3	1	1	1	1	1
LINUX91	1	1	1	1	3	7	1	-1	-1	1	1
LINUX92	1	1	1	1	3	6	1	-1	-1	1	1
LINUX93	1	1	1	1	3	6	1	-1	-1	1	1
LINUX94	1	1	1	1	6	6	1	1	-1	1	1
LINUX95	1	1	1	1	7	6	1	1	-1	1	1
LINUX96	1	1	1	1	7	4	1	1	-1	1	1
LINUX97	1	1	1	1	8	4	1	1	-1	1	1
LINUX98	1	1	1	1	8	4	1	1	-1	1	1
LINUX99	1	1	1	1	8	3	1	1	-1	1	1
LINUX00	1	1	1	1	8	3	1	1	-1	1	1

Table 4 Output Data for Research Trends

	Expected period before the idea turning into product	Expected influence level of this trend on current technologies	Level of support available	Expected risk level of research failure	Expected level of remaining technical bottleneck	Estimated adoption cost level
Ada79	2	5	8	5	6	8
Ada80-85	0	5	5	2	4	8
Ada86-90	0	4	4	1	5	7
Ada91-95	0	3	4	1	3	6
Ada96-00	0	2	2	1	0	6
Unix 69	1	3	1	6	7	4
Unix70-75	0	6	4	4	5	4
Unix76-80	0	7	5	1	5	3
Unix81-85	0	7	7	1	6	3
Unix86-90	0	8	7	1	6	3
Unix91-95	0	9	7	1	3	3
Unix96-00	0	9	8	1	5	3
Java91	4	0	2	9	9	4
Java92	3	0	2	8	9	4
Java93	2	0	3	8	8	4
Java94	1	0	3	4	6	5
Java95	0	8	6	1	5	5
Java96	0	8	8	1	5	6
Java97	0	9	8	1	4	6
Java98	0	8	8	1	4	5
Java99	0	9	8	1	4	4
Java00	0	9	8	1	4	4
CORBA92	2	5	4	8	6	7
CORBA93	1	5	3	8	7	6
CORBA94	1	4	5	7	6	7
CORBA95	1	4	5	6	4	6
CORBA96	0	3	3	4	5	6
CORBA97	0	4	2	2	4	6
CORBA98	0	4	3	2	4	6
CORBA99	0	4	3	2	4	6
CORBA00	0	4	3	2	5	6
XML96	3	7	3	4	4	4
XML97	2	6	2	5	3	3
XML98	1	7	2	5	2	3
LINUX91	4	4	5	6	7	6
LINUX92	3	4	5	4	8	5
LINUX93	2	4	4	4	7	4
LINUX94	1	5	4	3	5	4
LINUX95	0	5	6	3	4	5
LINUX96	0	6	6	2	5	5
LINUX97	0	6	7	2	5	6
LINUX98	0	6	7	2	5	6
LINUX99	0	6	8	2	5	6
LINUX00	0	6	8	2	5	6

Table 5 Output Data for Technology Trends

	Expected period before adapter can make profits	Expected influence level of this trend on current technologies	level of support available	Expected level of social resistance	Expected level of remaining technical bottleneck	Estimated adoption cost level	Level of market loss if not involved	Estimated level of competition among early adopters
Ada79	2	5	8	9	6	8	1	2
Ada80-85	1	5	5	8	4	8	3	2
Ada86-90	1	4	4	5	5	7	3	2
Ada91-95	1	3	4	5	3	6	3	2
Ada96-00	1	2	2	4	0	6	3	2
Unix 69	5	3	1	9	7	4	1	1
Unix70-75	4	6	4	8	5	4	1	2
Unix76-80	2	7	5	4	5	3	3	4
Unix81-85	1	7	7	3	6	3	4	4
Unix86-90	1	8	7	2	6	3	4	4
Unix91-95	1	9	7	1	3	3	5	4
Unix96-00	1	9	8	1	5	3	5	4
Java91	5	0	2	9	9	4	1	1
Java92	4	0	2	9	9	4	1	1
Java93	3	0	3	9	8	4	1	1
Java94	2	0	3	8	6	5	1	1
Java95	1	8	6	4	5	5	5	4
Java96	1	8	8	3	5	6	7	5
Java97	1	9	8	1	4	6	7	6
Java98	1	8	8	1	4	5	8	6
Java99	1	9	8	1	4	4	8	6
Java00	1	9	8	1	4	4	8	6
CORBA92	4	5	4	9	6	7	1	1
CORBA93	3	5	3	9	7	6	1	2
CORBA94	2	4	5	7	6	7	1	2
CORBA95	1	4	5	6	4	6	2	2
CORBA96	1	3	3	5	5	6	3	3
CORBA97	1	4	2	5	4	6	4	3
CORBA98	1	4	3	5	4	6	4	3
CORBA99	1	4	3	5	4	6	4	3
CORBA00	1	4	3	5	5	6	4	3
XML96	3	7	3	5	4	4	4	5
XML97	2	6	2	3	3	3	5	5
XML98	1	7	2	1	2	3	7	5
LINUX91	4	4	5	7	7	6	1	1
LINUX92	3	4	5	6	8	5	1	1
LINUX93	2	4	4	6	7	4	1	1
LINUX94	1	5	4	4	5	4	3	1
LINUX95	1	5	6	3	4	5	4	5
LINUX96	1	6	6	3	5	5	4	6
LINUX97	1	6	7	3	5	6	4	6
LINUX98	1	6	7	3	5	6	4	7
LINUX99	1	6	8	3	5	6	4	7
LINUX00	1	6	8	3	5	6	4	7

Table 6 Output Data for Market Trends

	Expected life span of the trend	Expected influence level of this trend on current technologies	Estimated market size($M)	Estimated market share (%)	Level of market loss if not involved	Estimated level of competition among early adopters
Ada79	20	5			1	2
Ada80-85	20	5			3	2
Ada86-90	15	4			3	2
Ada91-95	10	3			3	2
Ada96-00	10	2			3	2
Unix 69	30	3			1	1
Unix70-75	30	6			1	2
Unix76-80	30	7			3	4
Unix81-85	30	7			4	4
Unix86-90	30	8			4	4
Unix91-95	25	9			5	4
Unix96-00	20	9			5	4
Java91	10	0			1	1
Java92	10	0			1	1
Java93	10	0			1	1
Java94	15	0			1	1
Java95	20	8			5	4
Java96	20	8			7	5
Java97	20	9			7	6
Java98	20	8			8	6
Java99	20	9			8	6
Java00	20	9			8	6
CORBA92	20	5			1	1
CORBA93	20	5			1	2
CORBA94	20	4			1	2
CORBA95	15	4			2	2
CORBA96	15	3			3	3
CORBA97	15	4			4	3
CORBA98	15	4			4	3
CORBA99	15	4			4	3
CORBA00	15	4			4	3
XML96	10	7			4	5
XML97	10	6			5	5
XML98	10	7			7	5
LINUX91	20	4			1	1
LINUX92	20	4			1	1
LINUX93	20	4			1	1
LINUX94	20	5			3	1
LINUX95	20	5			4	5
LINUX96	20	6			4	6
LINUX97	20	6			4	6
LINUX98	20	6			4	7
LINUX99	20	6			4	7
LINUX00	20	6			4	7

Bibliography

1. Albertini, F., Sontag, E. “For neural networks, function determines form”. *Neural Networks*. Vol 6, pp975 –900, 1993
2. Back, A.D., Tsoi, A.C. “FIR and IIR synapses, a new neural network architecture for time series modeling”. *Neural Computation*. Vol 3, No.3 pp 375 – 385, 1991.
3. Bengio, Y. and Gingras, “Recurrent neural networks for missing or asynchronous data”, in *Advances in Neural Information Processing Systems*. Vol 8., The MIT Press, 1996.
4. Box, G.E.P., Jenkins, G. *Time Series Analysis*, Holden Day, 1967.
5. Elman, J. “Finding structure in time”, Cognitive Science. Vol. 14 pp 179 – 211, 1990.
6. Frasconi, P., Gori, M., Soda, G. “Local feedback multilayered networks”. Neural Computation. Vol. 4, pp251 – 257, 1990.
7. Giles, Lee and Gori, Marco, “Adaptive Processing of Sequences and Data Structures”, Springer, 1998
8. Haykin, S. “Neural Networks, A comprehensive foundation.” MacMillan College Pub Co. 1994
9. Kailath, T. *Linear Systems*. Prentice Hall, Englewood Cliffs, N.J., 1980.
10. MacGregor, R.J. Neural Brain Modeling, Academic Press, Orlando, FL, 1987.
11. Mili, Ali and Cowan, R.D. “Software Engineering Technology Watch”, 2001
12. Narendar, K.P., Parthasarathy, K. “Identification and Control of Dynamic Systems using Neural Networks.”. IEEE Trans Neural Networks, Vol 1. pp4-27, 1990

13. Nerrand, O., Roussel-Ragot, P., Personnaz, L., Dreyfus, G., Marcos, S. “Neural Networks and nonlinear adaptive filtering: Unifying concepts and new algorithms”. *Neural Computation*. Vol5, pp165 – 197, 1993.
14. Norman Foo and Randy Goebel (Eds) , "Lecture Notes in Artificial Intelligence, 1114". Springer-Verlag Berlin Heidelberg, 1996
15. Pavlidis, T., *Structural Pattern Recognition*. Springer-Verlag, 1977.
16. Petra Perner and Maria Petrou (Eds), "Machine Learning and Data Mining in Pattern Recognition", Springer-Verlag Berlin Heidelberg, 1999
17. Pineda, F.J. Generalization of back-propagation to recurrent neural networks. *IEEE Trans. on Neural Networks*, special issue on recurrent networks.
18. Rao, Valluru and Rao, Hayagriva, “C++ Neural Network and Fuzzy Logic”, MIS press, 1995
19. Schalhoff, R.J., *Pattern Recognition: Statistical, Structural and Neural Approaches*. John Wiley & Sons, 1992.
20. Tsoi, A.D., Back, A.d. “Discrete time recurrent neural network architectures: a unifying review”. Neurocomputing, Vol.15, pp183-224, 1997
21. Williams, R., Zipser, D. “A learning algorithm for continually running fully recurrent neural networks”. Neural Computation. Vol. 1, pp270- 280, 1989.