Fuzzy Neural Networks (FuNN) in the Palm Environment

Samuel Moyle, Michael Watts

December 2000-August 2002
Introduction

This document is prepared with two purposes in mind. The first is to introduce the roof maintenance problem and the proposed solutions. The second is to describe its implementation for the Palm platform.

The first section comprises a fairly comprehensive description of the roof maintenance problem and expected outcomes, as described by a roof maintenance expert. The purpose is not to train any individual to be come a roof maintenance expert, rather to demonstrate that the decision made by such an expert can be broken down into realistic components and subsequently acted upon. Because they can be broken down, it is possible to replicate and create an expert system able to make decision (at least in part) based on the values immediately available.

The second section concentrates on actual implementation. Firstly a prototype using desktop resources is described, then the re-development for Palm devices. Within this section, the decision process and steps taken are described – which will hopefully prevent others making some of the errors made in getting this far.

Section 3 describes the outcomes and directions that may be taken in the future. There are many things that can be done to improve interface design and make things easier for the end user– the roof maintenance expert.

Sam Moyle, August 2002
# Table of Contents

Introduction................................................................................................................. 1

Table of Contents ............................................................................................................ 3

Section 1 – The Roof Maintenance Problem .................................................................. 5
  Why Maintain? ................................................................................................................ 5
  What Can be Done? ......................................................................................................... 5
  How to Decide? .............................................................................................................. 6
    Roof Types? ................................................................................................................ 6
    PMT ............................................................................................................................ 6
    Concrete Tile ............................................................................................................. 9
    Iron ............................................................................................................................ 11
    ColourSteel .............................................................................................................. 13
  Other Roof Types ........................................................................................................ 14
    Fibrous roofing ...................................................................................................... 14
    Clay Tiles ................................................................................................................ 14
    Exotic Metals .......................................................................................................... 15
    Slate .......................................................................................................................... 15
    Membrane Roofing ................................................................................................. 15

Metallurgy 101 .............................................................................................................. 16

Inputs and Outputs for all roof types. ............................................................................. 16

Test cases ...................................................................................................................... 22

Section 2 - PalmOS ........................................................................................................ 27
  Current Development options - Analysis of tools......................................................... 30
    C and C++ ............................................................................................................... 30
    GUI Tools ............................................................................................................... 30
    Form Editors .......................................................................................................... 30
    Dev Utilities ........................................................................................................... 30

Install issues ................................................................................................................... 31

Common error messages .............................................................................................. 32

SDK Versions .................................................................................................................. 32

PalmOS Emulator (POSE) ............................................................................................ 32

Software Design Issues ................................................................................................. 33

UI issues ......................................................................................................................... 33

Review prototype software. .......................................................................................... 35

ANN ................................................................................................................................ 38

ANN Roof Type Classes ............................................................................................... 38

Software design ............................................................................................................. 39

Resources: ..................................................................................................................... 39

Fields ............................................................................................................................. 40

ScrollBar resource: ..................................................................................................... 42

SpinControl: .................................................................................................................. 43

List boxes and pop-ups: ............................................................................................... 44

Other useful things: ....................................................................................................... 45

CBIS Code Changes ...................................................................................................... 46

Friend ............................................................................................................................. 46

Int and Index .................................................................................................................. 46

Void vs. NULL ............................................................................................................... 46

NIL vs. 0 (zero) ............................................................................................................. 46

Assert.h ......................................................................................................................... 47

Extern handling ............................................................................................................. 47

Macros......................................................................................................................... 47
Section 1 – The Roof Maintenance Problem

Why Maintain?

It is generally accepted that the most expensive personal investment most people make is their home. It is, therefore, essential that this investment be maintained in the best manner possible, balanced with any expense involved. The roof of a home, providing protection for everything beneath, is one of the most important components making up a home. Maintenance of a roof is not to be taken lightly - errors or omissions can lead to devastating results, not the least of which are increased costs or extra maintenance in other areas.

Common residual maintenance resulting from damaged roofs include ceiling damage, damage to electrical fittings and electrical cabling, failed insulation, rot in timber-work, water damage to paint-work, wall-coverings or other furnishings. Ultimately, the roof material can fail, with other weather conditions adding to the damage (for example, wind damage).

What Can be Done?

The maintenance that is appropriate for a roof is very much dependant on the roof type, its condition, and factors specific to each roof type.

Standard roof paint is porous in nature, and generally slows (but does not stop) most forms of deterioration. Paint has the disadvantage of needing to be replaced or over-painted on a regular basis. A standard acrylic may last 5 years while newer varieties are being advertised as lasting 10 or more. Another disadvantage with most paint is adhesion - it is often difficult to get the paint to adhere to the chosen substrate. This can be overcome through the use of specialist adhesion layers. However, the greater the number of layers, the more likely that delamination (peeling) will occur.

The use of protective coating systems is an excellent means of extending the life of a roof while improving the appearance of the roof. As with paints, all 'protective coatings' are not created equally. Different roof types have different protective requirements; one coating will not suit all roof types (this is discussed further in the following sections). The thickness of the coating is also an important issue - too thick a material is as likely to cause problems as too thin a film of the same material. Roof surfaces that are textured run the risk of accumulating grime which can host moss and lichen growth. The feeding systems of organic growth will damage a roof no matter its material so it is recommended that roofs be kept clean. At the most extreme a roof may require water blasting to remove growth. More commonly a Calcium HypoChlorite gel solution (similar in nature to Sodium HypoChlorite found in household bleach) is applied to the roof. This has the advantage of killing growth spores found on the roof that may not otherwise be removed as well as killing existing growth. Reputable roof maintenance firms will use solutions that adhere to the roof and are slowly released only when the roof is wet. This has the benefit of keeping the roof clean for some time compared to a 'one off' application.

A roof may require no maintenance, or may need to be replaced. Replacing a roof is an expensive and time intensive exercise with removal sometimes costing as much as the installation of the new roof. A roof requiring no action takes no more than the time to establish the condition of the roof.
How to Decide?

It is sometimes appropriate to contact roof maintenance experts to assess the condition of a roof. Reputable experts will discuss with the property owner appropriate maintenance. This section discusses the main domestic roof types in New Zealand, the parameters to be considered, and the maintenance options that may be suggested along with their pros and cons.

Roof Types?

There are many roof types in use throughout New Zealand. The most common include Pressed Metal tile (PMT), Concrete Tile, Clay Tile, Iron (galvanised or zintelume finish), ColourSteel, Slate, Asbestos Sheet and some more exotic metals or composites.

PMT

![Figure 1 PMT displaying Chip Loss, Basecoat damage, corrosion, moss and lichen growth. This could still be repaired.](image)

Pressed Metal Tile roofs have been in existence since the mid 1960’s, the first being developed by Mr Lou Fisher. Mr Fisher, an Australian resident, served with the RAF during World War II. On a visit back to the UK in the late 1950's he discovered that most of the Nissen Huts erected during wartime were still in serviceable condition, even though largely ignored for more than 10 years, while newer buildings were showing signs of corrosion.

On investigation it was found that those buildings in good condition had been treated with a bituminous material instead of the traditional paint and that, where the bitumen was intact, the metal had been protected. Taking this concept back to Australia he expanded his existing home maintenance business to lightweight roofing. Instead of competing with existing iron roof manufacturers he decided to replicate the appearance of roof tiles (clay or concrete) using a lightweight metal, then provide protection using the bituminous material. The products were known as Fisher Tiles, and came in a 4 and 7-pan profile (a pan being each shaped dip in the tile, profile being the shape of the tile).
After several years the lightweight roofing part of Fisher Industries was purchased by the Holt Brothers (later to become a part of Carter Holt Harvey). They altered the profile of the tiles and renamed it DecraMastic tile roofing.

While it is true that, in the presence of the bituminous material, the metal substrate remains intact and corrosion free there are some inherent problems that needed to be overcome. The first is related to the manufacturing process. The metal substrate was stamped into shape then each tile was hand-dipped for the galvanising process (Electro-plating a layer of zinc for corrosion resistance). Where there was a reasonable amount of zinc applied, the bituminous material tends to come off quickly. Where there is little zinc applied, the bituminous material adheres noticeably better. This leads to the problem of determining when red rust (corrosion of the iron substrate) will occur - almost impossible to guess without some indicator of the amount of zinc present. The general rule used by maintenance experts is that any metal visible is sufficient to warrant attention (refer Figure 1).

Having been galvanised, the tiles are stacked awaiting shipment. This tended to attract moisture between tiles, which may not be removed before 'white rust' (zinc oxide) has formed.

Only when the tiles are ordered, and a shipment is prepared, are the tiles coated with bitumen and stone chip applied (for appearance). The addition of stone chip is also a problem. This does a number of things - the most important being a weakening of the coating at the points the stone chip is stuck. When the chip is washed or knocked off the roof, the weak points wear prematurely forming spots through the bituminous material, allowing corrosion to form at points on the roof.

The chip is an annoyance for property owners when loose chip washes off the tiles and into spouting. It is not unknown for down-pipes to become blocked and metal spouting to corrode prematurely because of this.

The chip itself, being coloured, adds to the pleasant appearance of the tiles, but the chemical composition of some stone chip can cause problems. For instance, the green chip used on these tiles has high iron content; iron oxides washing off the roof can contribute to premature corrosion of metal spouting and show 'rust' streaks down the roof. These effects are particularly noticeable where there are lead flashings in contact with the roof (dissimilar metal corrosion the culprit).

People will inadvertently damage the coating, often in the process of installing the roof. Being relatively soft, bitumen is easily damaged by people walking on the roof with inappropriate footwear, through workmen dropping tools, or through rough handling. It is not unknown for a roof at 10 years of age to be perforated (having a hole as a result of corrosion) at the point a tile was damaged during installation.

Moss and Lichen growth also tends to facilitate the early demise of a PMT roof. This is because the root systems literally eat the bituminous material, exposing the metal substrate to the elements (refer Figure 2).
The bituminous material, although an excellent method of preventing corrosion and deterioration, will only provide protection while it adheres to the roof substrate. Where metal has become exposed due to wear, replacement coating is required.

In an effort to address some of these issues, Carter Holt Harvey developed an acrylic polymer material for use instead of the bituminous material; these tiles were known as DecraBond roof tiles. This coating system is more resistant to rough handling but issues related to chip loss are still applicable. While these tiles are more resistant to moss and lichen growth, this will still occur where dirt builds up or where birds congregate.

DecraBond II uses acrylic resin technologies and represent a further advance over previous tile finishes. The resin finish is extremely resilient even where chip loss occurs. Moss and Lichen growth, while inhibited, will cause damage when left unchecked. Damage during installation can also be a problem, although one that is limited through the use of a plastic film over the galvanising. This adds a level of protection for the roof.

Most recently, Carter Holt Harvey have released a range of ColourTiles. These tiles have the same profile as earlier tiles but use a derivative of the protective coating range of finishes. These tiles are an improvement over predecessors in that there is limited opportunity for moss and lichen to become established (the tile surface is smooth), the coating is UV resistant and tends not to fade (oxidise) and can come in almost any colour.

Paint is sometimes applied to DecraMastic roofs, without success. One property of bitumen is that it expands and contracts a great deal with temperature change. Paint is a hard surface that cannot cope with the continual expansion and contraction. Eventually it cracks then flakes, off leaving the roof in a worse condition than before.

Instead, flexible coatings have been developed that flex with the original bituminous material. These usually take the form of a two-layer system. The first layer, a bituminous material similar in nature to the original, is applied. This acts as a bonding surface,
encapsulates more stone chip (restoring the textured appearance for the roof), and has curing agents that react with the outer protective surface. The outer coating must be of a flexible nature, and have similar characteristics to the bituminous material beneath. In addition to providing a colour, this layer further encapsulates chip, provides a flexible, UV resistant, non-porous surface. Because the layers cure together, the final result is a single, continuous, protective layer of material for each tile. There are a number of products in the market place, but those utilising modified acrylic technology (termed Elastomeric) are most successful in practise.

Concrete Tile

![Concrete Tile](image)

**Figure 3** Concrete tiles, displaying wear and evidence of Moss and Lichen growth. This roof has been chemically treated the year before.

Concrete Tile roofs became popular following World War II, a time when other building materials (particularly metal) were in short supply. As there was a building boom at that time, there sprang up many manufacturers of concrete tiles for residential roofing.

In general the tiles are made of a coarse cement/aggregate mix with a slurry finish. The slurry was usually a mixture of cement, sand, silica and coloured ochre. Depending upon the manufacturer, the cement mix may be compacted or gravity fed, air or force dried.

The original outer surface wore off after 5 to 10 years of exposure, leaving the concrete substrate open to the elements. Concrete is by nature a porous material, and the most common forms of deterioration are related to water and wind damage. The most obvious is surface damage - where the exposed surface is worn through the action of wind and rain (refer Figure 3). Less noticeable, but equally damaging is the passage of water through the tiles. This leaches lime and cement, weakening the tile, the tile becoming brittle. This process is also known as carbonation, due to the action of carbon dioxide on the lime and other elements used in the manufacture of concrete. This latter damage is particularly noticeable when beach sand has been used.
Occlusions (air bubbles) in tiles can also contribute to leaks. These air bubbles are not usually visible, but will contribute to the premature failure of an affected tile.

Hidden damage can occur with the weight of the tiles when wet. A concrete tile roof for an average 3-bedroom home of around 120M² weighs in excess of 3 tons when dry. It is generally accepted that, when wet, this roof will double in weight causing added stresses on the structure of the home. As a result it is not uncommon to see noticeable dips along roof faces.

Later concrete tiles (since the 1970s) rely on acrylic coating technology for protection of the roof as a whole. In fact, while the original surface is intact, these can provide excellent appearance and protection for the home. However, once the surface has been compromised the concrete substrate deteriorates very quickly. It is recommended by the manufacturers that these roofs have suitable protective surfaces at all times.

Porosity of concrete has an added side effect largely ignored by property owners - that of moisture within the ceiling cavity. Insulation can harbour moisture for some time, electrical wiring can become damaged through prolonged exposure and ceilings can be damaged through the action of water. Often it is difficult to establish particular tiles as causing problems, as water can collect on battens and travel long distances before finally dropping onto a ceiling. It is not uncommon for the ceiling cavity beneath a concrete tile roof to appear to have drizzle or a moisture haze during wet weather.

Moss and Lichen growth also damages tiles. Being cement, the tiles are attractive to organic growth - once established the tile substrate becomes food. It is not unknown for moss growth to eat completely through a concrete tile.

There are several schools of thought as to what is appropriate maintenance for concrete tile roofs. While moss and lichen growth can be obvious and damaging in nature, the real problems are wind and water.

Moss and Lichen can be removed through the application of Calcium HypoChlorite, which is similar in nature to Sodium HypoChlorite found in household bleaches. This has the advantage of killing spores as well as existing growth. Water blasting can also be undertaken where there is a substantial infestation, although there is the risk of water entering the ceiling cavity when inexperienced operators aim the cleaning tip inadvisedly.

Some people apply standard acrylic or alkyd roof paint to concrete tile roofs in the belief that it is extending the life of the roof. However, standard roof paint is porous (designed to allow the substrate to 'breathe'), therefore any advantage gained is temporary at best. It is notoriously difficult to get complete adhesion on concrete tiles - delamination is common in a short space of time. It is difficult to entirely remove paint from concrete roof tiles without water blasting, and the amount of work this involves makes it an uneconomic (not to mention risky) pastime.

A silicone solution may be sprayed onto the roof in order to attain a measure of water-protection. However, silicone treatments break down in UV light, rendering areas exposed to sunlight ineffective. Where the treatment has had the opportunity to become absorbed by the tile, effectiveness is reduced through incomplete coverage. Silicone treatments do nothing to prevent the wearing away of the tile itself through the action of wind and rain.

Provided that the roof is not too worn or tiles too brittle, there is the possibility of applying a protective coating to the roof. These protective coatings are of a specialised nature, allowing for the specific requirements of the concrete tile. One particular feature of concrete is that it will absorb moisture from its environment, moisture being an inherent component of concrete's strength.
The first consideration is that of adhesion. Specially designed adhesion layers have been developed that bind the outer cement surface, sinking into the tile to achieve this. An outer protective surface is then applied. This surface, when applied correctly over the adhesion layer, should have some specific properties. It must be UV resistant to avoid coating oxidisation and subsequent failure. No other layers of protection should be applied, as additional layers invite delamination (some materials require a clear UV protection layer to be applied over the coating). Protective coatings must be able to allow moisture through the coating from the adhesion layer side, while repelling moisture from above. This prevents the cement from breaking down beneath the protective coating. The protective material should cure into the adhesion layer creating a single, continuous, layer of protective material on each tile.

Iron

![Figure 4 Corrugated Iron displaying forms of corrosion.](image)

Galvanised iron roofing has been a popular material in New Zealand since settlers arrived in numbers during the early 1800's. Initially, indeed until the early 1960's, most galvanised iron came from England. This iron tended to be of a heavy gauge and had generous amounts of zinc for protection (often about 80 grams per square metre). In the 1960's, Australia became the prime source of Galvanised Iron, both the gauge and galvanising being of lesser quality than that sourced from England. Over the past 25 years, New Zealand has produced its own galvanised Iron in usable quantities (primarily from the Glenbrook Steel Mill near Waiuku). As time has passed the gauge of the iron has reduced, as has the amount of galvanising (to as little as 10 grams per square metre).

This has meant that the public perception of Galvanised iron as lasting for a long time has become a tradition. It has been experienced, even in areas like Central Otago (considered a place where metal does not corrode), that lesser grades of iron corrode noticeably. Corrosion protection has become an important issue (refer Figure 4).

In an effort to provide better corrosion resistance for iron, alloys have been introduced as a sacrificial coating. The most commonly marketed material is known as zincoalume. If the
roofing sheets are damaged during transit or when installed, the iron will tend not to corrode because the zinc/aluminium alloy will do so first. However, it was not appreciated early in the use of zincalume that most edge flashings, ridges, fixings and flashings for protrusions are made of lead, which reacts strongly with aluminium. Thus dissimilar metal corrosion occurs quickly, appearing as green or yellow oxides washing out from beneath the leaden areas. If left unattended for too long, damage (particularly perforation) is difficult to repair and sheet replacement may be necessary. Around 1995 soft-zinc or neutral plastic composite materials were developed to alleviate this problem. For similar reasons it is not recommended that galvanised iron and zincalume materials be mixed.

Traditionally, paint has been applied in an attempt to prevent corrosion. This can be successful in slowing, but does not completely prevent, the on-set of corrosion. There are two reasons for this. The first is that the adhesion layer used (galvanised primer) is designed to partially absorb the following paint layer. This has the drawback that, when paint cover is not yet applied, it absorbs moisture and holds it next to the metal - effectively hastening the onset of corrosion. This is a particular problem where galvanised primer is applied to sheet edges (laps) prior to laying.

The second is that standard roof paint is, by nature, porous, leaving the roof only partially protected. This is particularly true once the paint has become oxidised through the action of UV (sunlight). Paint provides a hard surface that does not always cope with the expansion and contraction experienced, particularly on large continuous surfaces. This problem shows itself as cracking of the paint surface but may also show as crazing, peeling and ultimately flaking of the paint. This is often associated with inconsistent layer thickness, either too much or too little material in places, or with application at extreme (cold or hot, ambient or material) temperature.

There are a number of successful methods of preventing corrosion long term. One (referred to in the PMT section) is the application of a bituminous material. This has excellent resistance to moisture, but, being soft, can easily be damaged by wind, hail, ill treatment and so on. Paint, being a hard surface, does not cope well with the continual expansion and contraction experienced, so a flexible outer protection is required where a bituminous material is applied to a metal surface.

Acrylic or Epoxy Resins can also be used as protective layers, however these are expensive and cannot easily be applied to roofing material in-situ. However, they have been used successfully for Pressed Metal tiles.

Cyano-acrylate paint materials also have excellent corrosion resistance (hence their use in the automobile industry). However, these are expensive and difficult to apply evenly over large areas. They should only be applied in controlled temperatures, particularly for the curing process. Specialist filters must also be available to capture residues and over-spray because of the toxic nature of the material in its uncured form. This is particularly true of the cyanide component - a cumulative poison that attacks the nervous system.

As with other roof types, protective coatings have been developed that allow for the nature of iron roofs and can be applied in-situ. These are usually two-layer systems - an adhesion layer and an outer protective surface.

The adhesion layer is a modified galvanised primer that allows for expansion and contraction and contains ingredients that allow curing with the outer protective coating. The outer layer is usually a modified acrylic that cures into the adhesion layer. The result is a single, continuous, non-permeable, UV resistant and flexible protective surface for metals exposed to the elements. Because of these various properties, quality protective coatings have a much longer expected life span than traditional paints.
ColourSteel is a pre-painted iron sheeting produced by the Glenbrook Steel Mill since the early 1980's. It is now in its third iteration of coating and substrate.

It was first formed as galvanised sheeting, onto which an adhesion layer, colour paint material and a clear UV resistant layer were applied. The second generation of material used zincalume protected iron sheeting, acrylic layers on both sides (for protection during storage and transit) with the original adhesion, colour and clear coats applied. The latest generation uses zincalume sheeting with a coating surface prepared by Carter Holt Harvey.

As the third generation of material has only recently been released, it is as yet unknown how it will react to weathering. The following comments will, therefore, only address issues related to the earlier generations of ColourSteel.

ColourSteel, as supplied from the mill, has good corrosion resistance at its outer surface. However, individual roofing manufacturers then roll-form the sheets into the profile desired by the customer. This introduces micro cracking at areas of coating stress or where rollers have manipulated the metal. These micro cracks allow UV to damage the colour coat beneath the clear UV protection layer, resulting in oxidation of the colour coating. Oxidisation is a good sign that the clear coat has been compromised or is no longer evident. Once oxidisation has occurred, it can reasonably be assumed that moisture is reaching the metal substrate, with corrosion occurring beneath the colour coat.

Often the first signs of corrosion are pitting, or crystalline structures at points along stress lines. As the corrosion grows, the paint delaminates from the substrate. As underlying corrosion protection is limited, this quickly resolves to patches of red iron oxide.

Corrosion is also common where metals are cut on-site; with corrosion beginning at cuts and paint delamination occurring in the same manner as with pitting sites. This is particularly true when abrasive cutting wheels are used as they introduce significant amounts of heat to the sheet, hastening the onset of corrosion. Carbon and other materials used in the manufacture of the abrasive wheel also contribute to corrosion forming electro-chemical reactions.

Particularly with the second generation of ColourSteel, dissimilar metal corrosion is common. This is because lead flashings were commonly used at ridges, hips and protrusions. Lead reacts strongly with the zincalume alloy, a fact not widely appreciated early in the use of ColourSteel. This results in corrosion growing quickly in these areas that are particularly difficult to repair satisfactorily. Around 1995 neutral soft-zinc and plastic composite materials were developed to alleviate this problem.

Corrosion of exposed surfaces at the underside of ColourSteel sheeting is also a problem, particularly in areas exposed to salt air. Pitting is common, to the point that BHP New Zealand (operators of the Glenbrook Steel Mill) will not issue warranty for roofs in proximity to salt water environments.

One attempt at disguising the deterioration of the roof material is that of using an automotive cutting compound to remove the oxidised layer of paint material. This certainly helps the roof to look better temporarily, but does nothing to prevent coating oxidisation or corrosion.

Using a standard paint on ColourSteel surfaces also helps the roof to look better for a short time. However the nature of the original paint material and clear UV protection layer resist adhesion with the result that paint literally falls off after a relatively short time. This can be partially alleviated through the use of an etching primer as an adhesion layer, but applying paint has the drawbacks listed in earlier sections. Of prime importance is that there is only limited protection resulting from the application of standard paints, along with the increased
risk of delamination of multiple paint layers. Etching primers are expensive and are difficult to apply evenly over a large surface. A self-etching material has been developed that allows a single layer of material to be applied that exhibits excellent adhesion to the original colour surface. This material forms a non-porous, flexible material that is able to expand and contract with the substrate, while providing protection and good looks for the roof.

**Other Roof Types**

**Fibrous roofing**

The most common form of fibrous roofing in use is Asbestos Sheeting in various profiles. This uses 'white asbestos' fibres set in cement, then formed and dried into the profile required. Usually these have no form of protection and have lasted remarkably well in practice. Strong and light, this is a favoured roofing material for commercial buildings from the late 1950's through till the early 1970's. However, over time these roofs become brittle and crack, particularly at the high points of corrugations and other stress areas. These roofs are susceptible to moss and lichen growth, which, when removed, reveal the asbestos fibres. Recent health scares related to Asbestos have made many people particularly nervous about working with Asbestos Sheeting, and many property owners are concerned about the long-term effects of runoff from asbestos sheeting on the environment. As a result of public hysteria there are serious and expensive requirements set down by OSH - the Occupational Safety and Health division of the Department of Labour - for those wishing to work with or remove entirely Asbestos Fibre panels.

Protective coatings are available that work extremely well on fibrous concrete material. There is a problem related to OSH restrictions in that such material may not be force cleaned (water blasted), which makes the cleaning of the roof in preparation for application of any protective surface difficult. Some experimentation has occurred in relation to water blasting with extraction and filtering tools, however the expense involved in getting these approved by OSH is proving a stumbling point in their utilisation. There is one company in Australia claiming that a two-pack (acrylic material, although similar to cyano-acrylates) material will be suitable, even though a roof may not have been cleaned first. This fails to completely neutralise the action of moss and lichen growth, as moisture can continue to feed it from the underside of the roofing material, causing the material to fail. There are concerns that this sort of activity will bring the roof maintenance industry into disrepute.

Protective coatings will not restore Asbestos Fibre material once delamination of the material has begun to occur (that is, the material has become friable). In this instance, replacement is the only realistic long-term option available for the roof.

**Clay Tiles**

Clay tiles have been manufactured all over New Zealand. These tiles are generally formed of local terracotta clays, fired with a clear or coloured glaze on the outer surface for weather protection and left unglazed on the lower surface. The purpose for not glazing the lower surface is similar to that of not applying a protective coating to the lower surface of a concrete tile - the tile must breathe. Where roofs are susceptible to condensation within the roof cavity, deterioration of the tile happens underneath, the tile 'rotting' and becoming brittle. The tile may, from the outer surface, appear to be in good condition.
Once the tile has rotted, even relatively minor weather conditions can cause holes to appear, hail being a fairly common cause of this form of damage. Because of the 'hard' nature of clay tiles, large hailstones can cause a relatively small hole on the upper surface and a large amount of material broken away from the underside; similar to the way a dumdum bullet causes damage.

Moss and Lichen growth will also damage the outer glaze and, left unattended, has been known to eat completely through clay tiles. Once a tile has been compromised the use of a silicon solution may help temporarily, but ultimately the tile will need to be replaced. Without its protective glaze, clay tiles deteriorate rapidly in weather.

Apart from keeping the roof clean and free from growth there is relatively little maintenance that can be undertaken. Replacement of damaged tiles, as necessary, will help maintain roof integrity. Where a large number of tiles have rotted, it may be more economic to replace the roof.

Protective coatings have not proven successful for clay tiles, as there is insufficient adhesion to the clay material available.

**Exotic Metals**

Metals other than iron are also used in roofing, however these are relatively rare, and can be treated separately.

Copper panels may be used in some instances. Where the entire roof (including flashings, ridging etc) is of copper, there tends to be relatively little corrosion evident, even after several years of exposure.

Aluminium sheeting, panels or tiles can also be found. Aluminium tends to be relatively stable once initial layers of oxides have formed. However, continuous exposure results in 'stress cracks' appearing, pre-cursors to intra-cellular delamination and the breakdown of the entire structure. Stress cracks appear first where material has been manipulated (during the forming process), and can be revealed by pressing on the material - if present, cracks following the grain of the metal will appear, centred on the stress point. There are coatings available that will extend the life of aluminium used in roofing, but these should be applied prior to stress cracks becoming evident.

**Slate**

Slate tiles have been used for centuries as a long-lasting, maintenance free roofing material. Slate tiles do, however, become brittle with age and may eventually be compromised by moss and lichen growth. Natural slate tiles are quarried but more recently reconstituted granite tiles have come available. These tiles offer similar wear characteristics, with a regular shape (making installation simpler), better weatherproof characteristics and lesser price.

**Membrane Roofing**

A flexible rubber or butyl material laid over ply, cement or metal, especially for flat roof areas, is a popular roofing material. These roofs tend to be very much waterproof, and, provided adhesion to the underlying structure is maintained, will last for a very long time indeed. The only maintenance that should be undertaken is the removal of moss and lichen growth, as this will compromise the integrity of the material if permitted to grow unchecked. These roofs should also be checked from time to time for lifting or delamination of the butyl material.
Metallurgy 101

Metals, in a natural state, are nearly always found in the form of oxides. Most metals are unstable and will, without protection, attempt to return to an oxidised form as quickly as possible. Putting dissimilar metals and moisture into the equation will encourage oxidisation (corrosion).

Oxidisation of metals is a result of electro-chemical reactions that are accelerated by water, heat, and the presence of different metals/electrical potential in close proximity. Some metals oxidise more slowly than others - a property that can be taken advantage of. For example, Galvanising is the process of electroplating zinc onto a less stable metal. The zinc corrodes slowly, while the underlying metal remains intact unless exposed to the elements. Zinc is often used as an adhesion layer between chromium and other metals - chromium being an extremely stable metal but nonadhesive.

Metals can also be used in a sacrificial manner. Boat builders attach zinc or magnesium blocks to steel hulls - the block corrodes, leaving the hull intact. This concept is used on roofs by zincalume material. This has a zinc/aluminium alloy electro-plated onto the iron. If the sheet is damaged, revealing the iron beneath, the alloy corrodes before the iron.

There are many forms of corrosion, each having quite different characteristics. Different metals will display different forms of corrosion according to the makeup of the material. The most common forms are described here:

**Surface oxidisation.** This is where the surface of a metal corrodes more or less evenly. This tends to form a thin film of oxide, which for some metals is quite stable and slows further oxidisation. Zinc, lead, aluminium and other metals exhibit this property.

**Pitting corrosion.** This is a localised form of corrosion, where corrosion occurs in a small area, while not affecting surrounding areas to the same degree. Crystalline structures may form early (small bumps of oxidised metal) while further corrosion occurs beneath. These crystalline structures will eventually be forced off, and corrosion of the substrate revealed which will be evident as pits in the material surface. This often leads to:

**Delamination.** This is the effect where corrosion of one material occurs beneath another layer, forcing the outer layer away from the substrate. This can be evident when zinc oxides lift paint material, when the iron substrate corrodes lifting the zinc coating, or with aluminium where corrosion will follow the grain of the metal from cuts, cracks, or stress risers.

**Flaking.** When a metal corrodes unevenly, patches of metal oxide will form and force other pieces of corroding material off the substrate. Iron, and the more unstable metals, tend to exhibit this property.

Commonly, roof maintenance staff use the terms 'White Rust' (meaning zinc oxide) and 'red rust' (meaning iron oxide). Lead Oxide tends to be a white/grey colour, but combined with zinc or aluminium oxide will be a yellow/green colour.

**Inputs and Outputs for all roof types.**

As previously discussed, there are 4 roof types that currently respond to maintenance, with a view to the long-term life of the roof. The following tables discuss the various parameters that roof maintenance experts consider when viewing a roof, along with possible outcomes.

Most ratings can be considered as being at a point along a continuum. This continuum may be represented as a line, the expert choosing a point on the line that is appropriate. The line can be considered as being made up of 100 points, the rating therefore is returned as a number between 0 (great) and 100 (lousy).
<p>| Inputs:         | Fungal growth | Fungal growth has the potential to do a substantial amount of damage to a roof if left unchecked. Commonly growth is found in sheltered portions of roof, or where material has been allowed to gather (where birds congregate, for instance). The area affected and maturity of growth may be rated - mature fungus of a substantial size being more damaging than many small new growths here and there. This is because many small growths do less damage individually than one that has been permitted to grow over time. |
| Chip loss | Chip loss is a good indicator of wear, and is related to base-coat loss. Some loss is expected early in the life of a roof, in later years loss is usually attributed to people walking inappropriately on the roof, weathering or fungal growth. |
| Base coat wear | Over time, or with harsh treatment, the base-coat may become damaged. This is a measure of how much of the roof is affected by base-coat damage. This is not just a measure of how much substrate is showing, as it is not uncommon for base coat still to be on the roof but not actually adhering to the metal substrate - with corrosion forming underneath. |
| Rust in Pans | Where the base-coat has become damaged or worn, corrosion will form. Initially this is in the form of zinc oxide, but over time (dependant upon the amount of galvanising present) the zinc will wear and iron oxide will form. This is an indication of the area of roof affected by iron oxide in particular. Once iron oxides are present there is generally not a lot of time available for repair to take place. |
| Rust in valleys | Valleys are prone to corrosion, usually because these have never been protected at all. Also, Valleys are difficult to replace making preventative maintenance an excellent idea. This is a measure of corrosion and its severity. |
| Output: | Do Nothing | Roof is clean and in good condition |
|          | Clean      | The roof has some moss and lichen growth, but is otherwise in acceptable condition |
|          | Re-surface | The roof requires protection to be re-established, and any other repairs as necessary |
| Replace | The roof is past saving, and replacement is now the only available option |</p>
<table>
<thead>
<tr>
<th>Inputs: Fungal Growth</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fungal growth has the potential to do a substantial amount of damage to a roof if left unchecked. Commonly growth is found in sheltered portions of roof, or where some attractive material has been allowed to gather (where birds congregate, for instance). How much of the roof affected in this manner can be estimated by looking at the roof. Growth may be rated also dependent on the age and type of infestation, mature fungus of a substantial size being more damaging than small new growths here and there.</td>
<td></td>
</tr>
<tr>
<td>Wear</td>
<td>The original coating is worn away, revealing the concrete substrate. The value used reflects the amount of wear evident at the surface of the tile, based on an expectation for the age of the roof. Wear can change rapidly from year to year, particularly at the latter stages of tile life. This may be different depending on location, weather experienced, tile type and so forth</td>
</tr>
<tr>
<td>Output: Do Nothing</td>
<td>The roof is clean and in good condition</td>
</tr>
<tr>
<td>Clean</td>
<td>Roof has some moss and lichen growth, but is otherwise in good condition</td>
</tr>
<tr>
<td>Re-surface</td>
<td>The roof requires protection to be re-established, along with any other repairs as necessary</td>
</tr>
<tr>
<td>Replace</td>
<td>The roof is too worn for application of protective material to be effective</td>
</tr>
<tr>
<td>Inputs:</td>
<td>Surface Corrosion</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------</td>
</tr>
<tr>
<td></td>
<td>White Rust</td>
</tr>
<tr>
<td></td>
<td>Red Rust</td>
</tr>
<tr>
<td>Outputs:</td>
<td>Do nothing</td>
</tr>
<tr>
<td></td>
<td>Add protective surface</td>
</tr>
<tr>
<td></td>
<td>Replace</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Faded ColourSteel**

<table>
<thead>
<tr>
<th>Inputs:</th>
<th>Chalking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The clear UV protective glaze is no longer evident on the roof, and the colour coating is oxidising. The current ratings are light, moderate or severe. The roof maintenance expert often needs to accurately reflect the depth and effect of damage. This will never be less than base-coat exposed.</td>
</tr>
</tbody>
</table>

| Base-coat       | A measure of the amount of roof showing its base-coat. As the base-coat is an adhesion layer only, exposure is an indicator that corrosion is not far away. This is a measure of the extent of base-coat evident and on the current form is rated as light, moderate or severe. |

| White Rust      | Corrosion of galvanising (or zincalume) forms what is commonly termed white rust. This is most commonly found at flashings where dissimilar metal corrosion is occurring. This is a measure of the amount and severity of white rust evident, on the current form this is rated as light, moderate or severe. Note - there is often a limited amount of zinc oxide visible on the surface of the roof, as the base-coat tends to hide this until iron oxides remove the coating. |

| Red Rust        | Iron oxides evidence themselves with red rust. This is usually at flashings first, but ColourSteel in particularly harsh environments will have iron oxides forming under leading edges, at laps, fixings or on the surface of the roof (particularly where pitting has first occurred). This input value reflects the amount and severity of this form of corrosion. On the current form this is rated as light, moderate or severe. |

<table>
<thead>
<tr>
<th>Output:</th>
<th>Do Nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The roof is in good condition and requires no further maintenance</td>
</tr>
</tbody>
</table>

| Re-surface      | The roof has some damage evident and will respond well to having new protective material applied |

| Replace         | This option is only applicable when the roof is severely damaged, and application of a new protective surface will not significantly enhance the life of the roof. |
**Test cases**

Test cases allow the developer and expert to agree on expected outcomes given particular input values. As each roof type has quite different properties, the inputs will vary.

The location and profile adjustments add to complexity. Although hidden in code, it is useful to know what values are used, as these will often influence outcome. Larger values indicate shorter life span; smaller values indicate a longer life span. In all instances the default is '1' indicating that a life span of around 50 years is expected.

It is useful to cover all possible input values, but this would not be realistic in most instances. Instead, we can put forward some extreme values, and a number between that cover likely input values. These are shown in the form of a table for each roof type.

<table>
<thead>
<tr>
<th>PMT</th>
<th>Location adjustment:</th>
<th>Profile adjustment:</th>
<th>Age:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Default</td>
<td>Default</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>Wanaka</td>
<td>DecraMastic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Invercargill</td>
<td>Harvey</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ocean Grove</td>
<td>Fisher</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HalfWay Bush</td>
<td>DecraBond I</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DecraBond II</td>
<td></td>
</tr>
</tbody>
</table>

**Inputs:**

<table>
<thead>
<tr>
<th>Growth</th>
<th>Chip loss</th>
<th>Base loss</th>
<th>Rust (pans)</th>
<th>Rust (valleys)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Do nothing</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Do nothing</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Do nothing</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Repair, re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Severe</td>
<td>Severe</td>
<td>Repair, re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Severe</td>
<td>Moderate</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Re-surface</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>------------</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Severe</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Low</td>
<td>Repair, re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Moderate</td>
<td>Repair, re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Severe</td>
<td>Repair, re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Low</td>
<td>Replace</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Replace</td>
</tr>
<tr>
<td>Low</td>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Replace</td>
</tr>
<tr>
<td>Moderate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Clean</td>
</tr>
<tr>
<td>Severe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Clean</td>
</tr>
</tbody>
</table>

Note: chip loss will always be equal to or higher than base coat loss. This is because, in order to damage the base, the chip must be removed.

Heuristics that can be observed are as follows:

- If growth is low and coating damage is low then no action needs to be taken.
- If growth is moderate or greater, the roof should be cleaned.
- If base loss is moderate or greater, then the roof should be re-surfaced.
- If rust is severe in pans, the roof should be replaced. However, if other conditions are low, the roof may be re-surfaced before perforation occurs.
- If the valleys have moderate rust, they should be repaired. If they have severe rust (or exhibit perforation) valleys should be replaced.

There are exceptions or cases when other conditions contradict a single severe item. The tools should be able to determine these instances and make adjustment as necessary. For example, a roof may have severe base-coat loss, but low rust or growth. Rather than replace the roof, it may be more appropriate to emphasise re-surfacing the roof.

<table>
<thead>
<tr>
<th>Concrete tiles:</th>
<th>Age: 35</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location adjustment</td>
<td>Profile adjustment</td>
</tr>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td>Wanaka</td>
<td>1</td>
</tr>
<tr>
<td>Invercargill</td>
<td>1.5</td>
</tr>
<tr>
<td>Ocean Grove</td>
<td>1</td>
</tr>
</tbody>
</table>
Heuristics that can be observed are as follows:

- If growth is low, coating damage is low and rust is low, then no action needs to be taken.
- If growth is moderate or greater, the roof should be cleaned.
- If the original protective surface is damaged or missing then the roof should be re-surfaced.
- If wear is severe the roof should be replaced. However, if other conditions are low, the roof may be re-surfaced before the number of tiles requiring replacement makes re-surfacing un-economic.
<table>
<thead>
<tr>
<th>Low</th>
<th>Low</th>
<th>Low</th>
<th>Do Nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Moderate</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Low</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Low</td>
<td>Protect</td>
</tr>
</tbody>
</table>

Heuristics that can be observed are as follows:

- If a roof has severe red rust, it will ultimately require replacement - no amount of surface protection will help.
- If a roof has moderate red rust, protecting the roof is still an option, but should be undertaken before laps are damaged. Also, if white rust and chalking are severe, consideration should be given to replacement.

There are exceptions or cases when other conditions may take precedence. The system should take these events into consideration.

<table>
<thead>
<tr>
<th>Faded ColourSteel</th>
<th>Age:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location adjustment:</td>
<td>Profile adjustment</td>
</tr>
<tr>
<td>Default</td>
<td>1</td>
</tr>
<tr>
<td>Wanaka</td>
<td>1.6</td>
</tr>
<tr>
<td>Invercargill</td>
<td>1.5</td>
</tr>
<tr>
<td>Ocean Grove</td>
<td>1.2</td>
</tr>
<tr>
<td>HalfWay Bush</td>
<td>1</td>
</tr>
</tbody>
</table>

Inputs:

<table>
<thead>
<tr>
<th>Chalking (UV and Coating)</th>
<th>Base coat wear</th>
<th>White Rust</th>
<th>Red Rust</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Protect</td>
</tr>
<tr>
<td>Moderate</td>
<td>Moderate</td>
<td>Severe</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Moderate</td>
<td>Moderate</td>
<td>Severe</td>
<td>Moderate</td>
<td>Protect</td>
</tr>
<tr>
<td>Moderate</td>
<td>Moderate</td>
<td>Severe</td>
<td>Severe</td>
<td>Replace</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>--------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Low</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Severe</td>
<td>Replace</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Moderate</td>
<td>Replace</td>
</tr>
<tr>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Severe</td>
<td>Replace</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe</td>
<td>Low</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Moderate</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Severe</td>
<td>Replace</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Low</td>
<td>Protect</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Moderate</td>
<td>Replace</td>
</tr>
<tr>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Severe</td>
<td>Replace</td>
</tr>
</tbody>
</table>

Note: chalking will always have a value greater or equal to base coat wear. This is because chalking of the original outer surface must occur before the base coat can become worn.

Heuristics that can be observed are as follows:

- If all inputs are low, no action is required.
- If red rust is severe, the roof will (ultimately) require replacement.
- If red rust is moderate, the roof should be Re-surfaced.
- If white rust is moderate or greater, the roof should be Re-surfaced.
- If chalking is severe, or base coat wear is moderate or greater, the roof should be Re-surfaced.

There may be exceptions or cases when other conditions may take precedence. The system should take these events into consideration.

The heuristics above are purely rules of thumb and are appropriate for the default values suggested. Adjustments dependant upon age, profile and location will mean that the results will become more (or less) severe.
Section 2 - PalmOS

"It needs to take into account small screen size. It needs to limit text input on the handheld. It needs to seamlessly sync to a program on a desktop computer. It needs to be small. It needs to be fast." (Rhodes and McKeenan 1999, Ch 1)

PalmOS has been developed with the use of the tool in mind. It complements the need for applications that are fast to load, simple to use, and can be considered an extension of the desktop computer.

In the late 1980's and early 1990's a number of PDA's (Personal Digital Assistants) came into the public arena. As a general rule these fell into two broad categories. The first were those with a specific task in mind - unable to do anything else. The second were tools that seek to replicate what you find on a desktop computer. In most cases the latter failed because they were bulky, slow and, in some instances, difficult to use.

Jeff Hawkins, the man instrumental in bringing the Palm devices into being, had a hand in a number of these devices. It was his experience that brought him to the realisation that, in order to succeed, a hand-held device needs to be writeable, wearable, extend (not replace) the desktop, and be fast.

What does this mean? Writeable refers to the ability to add information quickly, and in some intuitive fashion. People use pens to write normally, so the use of a stylus is a logical extension. In surveys, people frequently requested handwriting recognition, but to develop software that could decipher all writing styles and characters would cause the device to require too much memory and be far too slow to be of any practical use. Instead, Jeff Hawkins developed 'graffiti', a form of shorthand that allows individual characters to be easily distinguished. This made the process simpler, faster and more fun for the user (the assumption being that if someone wants a toy, they generally like to learn new things too) (Dillon 1998).

Wearable refers to a device being small enough that it can be carried in a pocket but large enough that a sufficient amount of information can be displayed. There are now computing devices the size of a credit card, but these are difficult to use and suffer from little screen display area. Larger and heavier devices also abound without great advantage (Rhodes and McKeenan, Foreword).

Extending the desktop took some consideration. Often people want to replicate some basic functionality to be found on the desktop then transfer information between the palm-top and the desktop machine. Users are not expected to browse the Internet, but may want to receive e-mail. Appointments and other notes of relevance should also be shared between the desktop and palm. If people want all the functionality and power to be found on the desktop they will get a laptop for this purpose.

Speed is not referring to number crunching power, but how quickly applications make themselves available and useful tasks are completed. For example, a user may want to record some data (say a telephone number) - they will not be prepared to wait before being able to write whatever they want. The HCI rule-of-thumb is 3 mouse-clicks (stylus taps, in this case) from nothing to using (Rhodes and McKeenan, 1999).

PDAs ideally should not have a 'boot up' time as found on a traditional desktop machine. Again, this is because of the user expectation that everything should be available instantly (Mykland 2000, Page 370).

The PalmOS screen (display/input region) is 160x160 pixels - not a lot of real estate considering that a desktop computer displays 800x600 pixels or more. This means that
anything not relevant should be omitted or moved. Some of the questions to be considered, as posed by Mykland (2000, Page 371), are:

"Is this piece of information adding value to the user?
Is this piece of information making the screen easier to use?
Is this piece of information placed where the user can see it easily?
Is the screen too cluttered for easy use; in other words, is this really the right screen for this information to reside in?
Is this information used frequently or infrequently? (If infrequently, you may want to consider making it part of the "details" screen that requires more taps to get to, in interest of keeping the frequently used screen uncluttered and optimally useful)"

Space is important. Blank space is as important as text or imagery. Blank space, used wisely, will help the user to pick out valuable information quickly and easily. Consider also that most users will not have colour displays, usually monochrome or 4-shade display.

Alignment is vital - the human eye will pick out inconsistencies immediately.

Design should also allow for the lack of stylus (it is bound to happen). If possible, all options should be available without having to use a stylus. Single-tap choices are to be used wherever possible as fewer taps are better.

It is also too easy to accidentally choose an option placed too close to another. For this reason it is wise to place dangerous options far from commonly used options (like save and delete).

Calculations slow things mightily. For this reason, any calculations should be either asked for, or done at the end of input (as processing will slow proceedings noticeably).

Finally, it should be pointed out that early Palm devices have only limited memory available, often 1 or 2MB in total. This means that any application must be frugal in its use of memory. Remember that there is no secondary storage - data, run-time and background applications share the available memory. Another issue is that, when in use, certain chunks of memory will be locked - the memory portion is readable only (Refer Figure 5).

The following information is taken from Chapter 4 of "Palm Programming: The Developer's Guide" (Rhodes and McKeehan 1999).

The memory in a palm device is separated into two areas - storage area and dynamic memory. Storage can be read from and written to but is treated separately from memory that can be allocated temporarily. Databases are stored here, as are 'ROM' components. Databases may be resources (applications) or record structures.

Dynamic memory allocations may contain any other form of memory usage as required. In particular, there are two main forms of usage - handles (re-locatable chunks of memory) or pointers (non-re-locatable chunks of memory).

Pointers are treated as fixed memory chunks and should, therefore, be avoided where possible. The OS in the interests of memory use optimisation may move around handles as necessary. However, to be manipulated (read, write, etc), a memory chunk must be locked, and unlocked after use. If a global variable is used repeatedly throughout the life of an application, it makes sense to use a pointer. Where a variable is used only occasionally, handles are better. Bear in mind that there may be a performance problem should repeated locking/unlocking of handles occur.

The number of times a single handle can be locked has a maximum of 15 times concurrently, and may remain locked if not unlocked the same number of times. This can be evident where
a handle is locked/unlocked in multiple locations. This can partially be alleviated in two ways. One is to add a 'LockHandleCounter', incremented when the memory chunk is locked and decremented when unlocked. Although this adds a level of complexity to code it does allow the developer to build safeguards. Another method is the use of temporary pointers that are created, used, and then disposed of when finished with.

As mentioned earlier, the restricted memory available on a PalmOS device means that tight control on memory is required. This creates problems when manipulation is required - more chunks must be allocated to cope with demand, the number and size of these chunks will determine how much you can do. With smaller RAM sizes, memory chunks can be 32k only. For devices with more memory larger chunks can be assigned - an 8mb device can explicitly allocate chunks up to 128k in size.

Memory should be 'freed' when finished with. Memory leaks - where chunks are allocated, then not destroyed when finished with - can occur. Although rare, these can be avoided through careful coding and attention to detail.

Thus the challenge is set for developers to build fast, robust, memory efficient software, while retaining a usable graphical interface providing acceptable levels of functionality.

Figure 5 Memory allocation within Palm devices (Rhodes and McKeehan 1999, Ch 4)
Current Development options - Analysis of tools

At present there are several means of building application software for the palm environment. All have benefits and disadvantages, some of which will be discussed below. There are 4 camps of development, with some overlap.

C and C++

The first are C or C++ based compilers. These result in compact binary code that can then be loaded onto the Palm. The primary examples are GCC (free software, command line plus some plug-ins) and CodeWarrior (a commercial GUI IDE from MetroWerks).

GCC is fast, compact, stable, and is available for the Unix/Linux and Windows platforms. However, the interface is difficult for novice code writers. It relies on additional software for displaying results and debugging facilities.

MetroWerks CodeWarrior is more thoroughly developed, providing debugging facilities within the IDE. It incorporates a graphical design environment along with line editing, allowing the creation of complex code.

GUI Tools

The second group comprises graphical front-ends to the command line software. Although few in number, those that have survived have done so by successfully building on existing software code development tools. One example is VCP that builds on either the CodeWarrior or GCC code-base.

Form Editors

The third group work differently, being form editors without the availability of adding other functionality. The two most widely known examples are Pendragon Forms and Satellite Forms. Pendragon Forms has the reputation for not being the best software in the world, and is not widely recommended. Satellite forms software is better, but is limited to building forms based on existing database tables (usually from MS Access). This means that there is no provision for adding functionality, nor is there provision for linking tables on one form (for example an order entry system on a desktop computer may have several tables accessed to create one screen form).

Another example, allowing the building of forms using standard components, is PDAToolbox. This rapid development tool can connect to existing relational database tables, and is useful for building data capture or input work. However, limitations include not being able to put complex code beneath, inability to link to code modules, and that custom input functionality is not available.

Dev Utilities

The final group is that of utility software for software design. One commonly used tool is POSE (PalmOS Emulator). This allows the developer to run a program on the desktop computer as though it were running on a palm device. This tool includes some memory debugging options, including one known as 'gremlins'. Gremlins have the ability to randomly choose input, thus testing that an application is robust and will not crash in operation. POSE is a development of such tools as PiLRC (Pilot Resource Compiler) and PiLRC-UI (Pilot Resource Compiler - User Interface), both of which have been integrated into other development environments.
One further group of development tools are, strictly speaking, not development environments but extensions of the existing tools. These are 'conduit developers', that allow applications to communicate with their desktop counterparts.

One other important component of all these development tools is the PalmOS SDK. This provides the basic resources that can be used by PalmOS applications, ensuring consistency of operations and limiting opportunity for code to crash. The SDK is used by all of the development environments listed above, and constructs are similar whether developing in a Windows, Macintosh or Unix environment.

**Install issues**

Take the CD and throw it at the machine. In theory, it will actually go!!

But seriously folks...

The following instructions describe the installation process for CodeWarrior on the MS-Windows platform.

The installation software comes on CD. There should be a file named setup.exe (or similar) that will run through a set-up process, in a similar manner to other software. You have the option of destination and so forth.

One glitch in software is that of locating correct fonts for use. Once CodeWarrior is installed, use windows explorer to locate the 'fonts' directory in your 'windows' location. This allows CodeWarrior to use the correct fonts.

Having set this up, it is worth installing and testing POSE (PalmOS Emulator). This tool allows testing of development software without the need for connecting an actual device. Note that final software should actually be user tested on actual PDAs, not just on the emulator. Installation of POSE is discussed later.

Having completed the installation process, there are some tasks to be completed before work commences.

This option allows the 'int' type to match the ANSI C version:

With an open project, choose Edit/<projectName> preferences.

Choose Code Generation/68k Processor.

Check the box labelled '4-Byte Int'.

If you want to re-use code, you may need to store your project as 'stationery'. This is, effectively, being able to copy whole projects and work on them as separate projects. This means any project can be copied whole into your project folder and accessed by any other project.

If you wish to modify an existing project, creating a new and improved version, the following settings will be useful to alter:

Once CodeWarrior is open, open the newly created/copied project.

Select Edit/<OldProjectName> Settings.

Choose Target/Target Settings. Change the Target from <OldProjectName> to <NewProjectName>. Save the settings.

Choose Target/68K Settings, then change the file name from <OldProjectName>.tmp to <NewProjectName>.tmp.

Choose Linker/PalmRez PostLinker. Change the Resource Files setting from <OldProjectName>.tmp to <NewProjectName>.tmp. Change the Output File setting from
<OldProjectName>.prc to <NewProjectName>.prc. Change the Database Name setting to <NewProjectName>-<CreatorID>

**Common error messages**

At this time, most errors that came with CodeWarrior 6 Lite, our initial version, have been addressed in CodeWarrior 7 (the version we are currently using). Most known errors are listed on the Metrowerks website:

http://www.metrowerks.com/products/palm/

**SDK Versions**

There are a number of Software Development Kits made available by Palm, version 4 being the latest. We began with SDK3.0 and are currently using version 3.5. We have resisted changing to version 4.0 until upgrading to CodeWarrior 8 or later.

As with all changes to underlying development material, there are problems that may occur. It is, therefore, worth checking toolmakers’ websites for latest information. For example, Palm has a full list of available information available through:

http://www.palmos.com/dev/tech/tools/

This site hosts all SDK versions and FAQ's associated. There are also full installation instructions for the major development environments.

Installing the latest SDK into CodeWarrior is usually pretty easy - installers are available for this. As at January 2001 SDK 3.5 is the most current version and the self-installer is named 'sdk35-cw.zip'. Provided you choose the correct directory for installation, this will do everything for you.

There has, at this time, been an update issued, which fixes a couple of minor programming issues. The file 'sdk35-update1.zip' should be extracted to a temporary directory and the instructions provided followed.

**PalmOS Emulator (POSE)**

POSE is a tool that emulates the look and feel of a PalmOS device on your desktop development machine. In testing with CodeWarrior, POSE is proving an excellent means of testing software. It is fast, stable and integrates with the IDE - for example using the debugger component allows visualisation of the current state and value of variables, run-time code-breaks and step-through code checking to take place.

For CodeWarrior:

I recommend installing POSE to its default location (following the appropriate prompts) then copy the directory into the CodeWarrior installation directory. Run CodeWarrior and follow this process:

Select Edit/Preferences.

In the left tree list, under the 'Debugger' entry select 'Palm Connection Settings'.

Change the 'Target' combo box to read 'PalmOS Emulator'.

Check the 'Always Launch Emulator' option.

Click the 'Choose' button and find the path to the desired copy of 'Emulator.exe'.

Click the 'Save' button.

Shut down CodeWarrior, ensuring that POSE is not running.
When you re-launch CodeWarrior, POSE should start also.

POSE requires the use of a PalmOS ROM. There is an extraction or transfer program available for this task. You must first install the file 'transfer.prc' onto your device, then with the device still in the cradle, run POSE. One of the options at startup is that of 'Download'. Choosing this option will take you through a number of steps (most can be set at the default values). Once the ROM is on your PC then it can be accessed and used by POSE for testing purposes.

Once this is working correctly, you will find that, when debugging (choose 'debug' from the project menu) the program will compile and run on POSE. This allows you to use the program as though it were on a device. Note: always test a completed program on your device before rolling out to clients. This helps avoid unpleasant surprises.

**Software Design Issues**

UI issues

Why use a Palm, not a laptop or paper?

In the beginning, there was paper. Paper is pretty cool for many reasons - it is cheap, made from renewable resources, can be recycled. People are fettered only by artistic ability. Anything that can be represented in imagery or text can be captured.

However, there are as many drawbacks. When gathering data, paper forms can be used and often people using forms do not think outside the square(s). More importantly, additional tools are required in order to make the paper truly useful - a clipboard or other portable writing surface, a pen/pencil or blood. If the user is in an out-door situation the paper can be rendered useless by weather conditions - wind, rain, or other environmental conditions.

Exceptions to these situations abound - at some significant cost. Use of plastics and specialist pens or pencils were briefly popular. Waterproof notepads or folders are also available. But all are cumbersome and inconvenient in many instances.

Take for instance our intrepid roof maintenance experts. When climbing a roof, the expert has to carry with them a folder or clipboard, writing implement, perhaps a tape measure. On a windy or slightly wet day this becomes somewhat dangerous.

Now picture a different situation. The expert climbs the ladder with both hands free. They visit all corners of the roof, gathering point data using a GPS. Points of interest are recorded, and notes written directly onto the surface of a device the same size as the experts’ hand. This device, when finished with, can be slipped back into the pocket from whence it sprang.

Science fiction? A little Trekkie? Well, no actually. In fact, just this sort of scene is played out in many corners of the world now. People everywhere are beginning to use tools that are personalised for their work, work well, and don't pretend to do everything. In fact the basic premise behind the Palm platform is simplicity. Jeff Hawkins, the motivating force behind the Palm phenomenon, recognised two prime forces - people are smarter than computers and that a palm sized device will be treated as an extension of the desktop, not a replacement (Dillon 1998).

How does this relate to our friendly roofing man? Well, if a Palm sized computer has the ability to take notes, allow pictures to be drawn, can be connected to a GPS, and maybe even intelligent enough to do some of the work, then there is no reason why the small device cannot replace the increasingly dangerous and inconvenient paper. Back at the office the data collected can be sent to the desktop computer and any reports can be printed - more professional than handwritten reports!
The small screen can be a problem for data input and display - 160 pixels square is small but not insurmountable.

Given that a palm sized device is appropriate for our task, and that users will be happy to do some learning, let us look at usability issues in general and their bearing in this environment.

Nielsen provides 10 usability heuristics that can be used to measure subjectively, the usability of a tool (Nielsen 1993, Page 10). The ten heuristics are: visibility of system status, match between the system and the real world, user control and freedom, consistency and standards, error prevention, recognition rather than recall, flexibility and efficiency, aesthetics, recovery from errors, and the availability of help. Hoh and Thomas (2000) have revisited these heuristics, determining that most can be used without modification against Palm applications.

There are some background considerations that are inescapable. In the Palm environment the need for some heuristics have been partially avoided or mixed together because many things found on a desktop computer have been simplified in the search for small size and simplicity of design. The most obvious heuristics (as defined by Nielson) that are combined are Error prevention, Visibility of system status and Help users recover from errors. In fact, the commonly provided means of dealing with these issues are often combined, less powerful, functions that users would expect to find on their desktop computer. This is, of course, in keeping with the keep it simple paradigm espoused by Hawkins. Matching the system to the real world and recognition and recall are also intermixed when considering the Palm devices currently available. Many of the cues provided are taken not so much from the real world as from the world of science fiction. For example, 'beaming' information between devices is similar to what you may find in any episode of Star Trek, as is the appearance of the device.
Review prototype software.

Given the task, and the assumption that we are working with Artificial Neural Networks (ANNs), the next step to be taken is the design of an appropriate interface, ensuring that what can be built on a computer will actually give the results desired (indeed, expected) by industry experts.

A prototype was designed and built using Borland C++ builder, as this allows for the construction of an interface with a structure that closely resembles a PalmOS application.

The interface consists of three panels with changes dependant upon individual roof types. An entry, or main, panel gives the user the option to choose a location, then a roof type (PMT, Concrete, Iron or ColourSteel). There is also an exit button, which closes the program (useful for Windows, but not implemented on a PalmOS device).

Each roof type has a new panel that allows the expert to choose the particular roof profile (refer Figure 6). For instance, a PMT roof may have a Harvey, DecraMastic, DecraBond, DecraBond II, or some other profile. There are also 'sliders' that allow the expert to position their estimation of wear (or other consideration) based on what is seen. This allows the expert to reflect the 'analogue' nature of wear characteristics - time, and therefore deterioration, is a continuous process.

Below these inputs, the results returned by the ANN are displayed. This is done in real time - as changes are made the ANN is queried and results displayed.

A reset option is included should the expert wish to begin again. This resets values to those expected by the software, along with results. Each ANN is instantiated directly as part of the code attached to the panel.

The 'next' button opens a new panel, with more input options available. This was built to demonstrate the further developments that are available - with modification these inputs may be used in the quote process. This will not be implemented in the first version for Palm, rather a later development.

Finally, the exit button closes the current panel.

![Concrete Roof](image)

**Figure 6 – Desktop prototype.**

The software design itself was built with simplicity of design in mind, to be read by people rather than speed or compactness of code. The structure was as follows:

<table>
<thead>
<tr>
<th>Panel</th>
<th>Global Variables</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>LocationAdjuster</td>
<td>PMTProfileAdjuster</td>
</tr>
<tr>
<td>------</td>
<td>------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td>PMTLocationAdjuster</td>
<td>btnClick()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cspinChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProfileChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DisplayResult()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reset()</td>
</tr>
<tr>
<td>CONC</td>
<td>CONCLocationAdjuster</td>
<td>btnClick()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>concProfileAdjuster</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scrollChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cspinChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DisplayResult()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reset()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProfileChange()</td>
</tr>
<tr>
<td>IRON</td>
<td>IronLocationAdjuster</td>
<td>btnClick()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scrollChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cspinChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DisplayResult()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reset()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProfileChange()</td>
</tr>
<tr>
<td>FCS</td>
<td>FCSLocationAdjuster</td>
<td>btnClick()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>scrollChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cspinChange()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DisplayResult()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reset()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ProfileChange()</td>
</tr>
</tbody>
</table>

The PMT roof stands out as being the one roof type that performs better using fuzzy rules than an artificial neural network. This is because conflicting input values tend to confuse the NN when training, a situation that is overcome through using a Fuzzy Neural Network.
(FuNN). As a result, all decisions are run through FuNN rather than a back-propagation ANN.

Each of the other roof type panels have associated with it a FuNN that is instantiated when the DisplayResult() method is called. Each FuNN is destroyed when the method is closed. It is possible in this way to avoid having all FuNN's instantiated and operational concurrently saving demand on memory. However, this places much more load on the CPU as each FuNN object must be instantiated every time the scroll change event is processed. It is better to instantiate the FuNN when the form is loaded and remove it when the form is closed.

In practice concurrent operation will never occur because an expert will only ever be on one roof at a time.

PalmOS, being designed using C, has some limitations not catered for in the prototype design. To this end, the design shown in Figure 7 was put forward for the underlying engine component:

![Figure 7 Layers of abstraction](image)

This design adds a layer of abstraction that makes memory use more efficient in a restricted environment. Having fixed memory addressing for common elements should result in some performance gains. This added layer of abstraction also allows easy creation of new classes and maintenance of existing classes.

![Figure 8 – Layers allowing independent update of the FuNN](image)
Instantiation of an abstract class in this manner also makes the application interface consistent. Overriding methods creates an easy way to modify the software in any manner required during development (Figure 8).

A description of the key methods and global variables follows:

**ANN**

The virtual ANN class has a `Recall()` method, where the input values are received (as an array of floating point values) and Output values are returned (also as an array of floating point values).

**ANN Roof Type Classes**

The `Recall()` method is inherited from the ANN base class.

Each ANN class will have the same design, the only real difference being the number of and values contained within the weight matrices. Each can be pre-compiled, based on the virtual class above. The matrices can be stored as arrays so some database structure may be implemented for this to further save room. This would mean only one ANN needs to be built. The downside of this approach is speed so a compromise will have to be made at implementation.

The `RoofType()` constructor and `~RoofType()` destructor are used for memory management, in an attempt to keep control of the restricted resources available in the Palm platform. The appropriate ANN is created at this time, the destructor removing it from memory when `RoofType()` is closed.

The `DataProcess()` and `Decision()` methods are abstract, ready to be implemented in the derived roof type methods.

The `Process()` method passes the input parameters to the NN, and catches the returned values. This acts as follows:

- Run the `DataProcess()` method
- Pass those values through the ANN
- Pass ANN values through the `Decision()` method
- Return the result in string form for display

The `DataProcess()` method (a derived method) takes care of collecting inputs and collating them ready for the neural network to process. The output from this method is an array of values that are passed by the `Process()` method into the ANN. The ANN returns numeric values that are then passed to the `Decision()` method.

The `Decision()` method (also a derived method) takes the results and converts them from numeric values to a sentence form - something the expert can readily understand. These sentences are then returned for display.

Most of the mechanics for these methods can be taken from the working prototype already created, as they are successful in operation. It is expected that the `Decision()` method will be called whenever a form control has been manipulated, querying the neural network in real time. The ANN is created when the form is opened, and destroyed when the form is closed, helping to speed real-time operations.
This section describes some of the constructs used in the building of a PalmOS user interface in the CodeWarrior IDE. Although many of the constructs are the same when using GCC, there may be some minor differences.

Also, when using some of the GUI tools, much of the background code is hidden, and cannot easily be accessed by developers. When building a front-end to a database this approach is fine but it can be difficult to attach specialist code.

This material is not intended to be comprehensive - rather this is a 'how to' covering the essentials for the Roof Maintenance project. The use of Fields is described, along with Scrollbars, SpinControls, Pop-Ups and List boxes. All of these are used in various ways by the interface for input and output purposes. The means of implementation for each of these resources can be replicated or adapted in any way necessary. In some instances code has been modularised - not the most speed efficient but much easier to develop further and maintain.

When reading the following sections, the following details must be considered:

- Due to the constraints of memory, everything is accessed by way of pointers or handles. When an object is to be manipulated, it must first be locked (the memory chunk prevented from being moved by the dynamic memory handler) so it can be read.

- Some functions require the handle, some the pointer. Be careful. If in doubt, get both... A pointer 'points' directly to a memory address, a handle holds a pointer.

An overview of how all the pieces go together is described last.

Resources:

Constructor is a program that allows the interactive building of form resources, to which may be attached code. Any sort of form, interface tool or global constant used may be built. The result used by attached code is known as a resource file. Names given to items are related to a unique index - which can then be given a handle and pointer (refer to the description of field or scrollbar resources that follow).

This is done in an interactive manner - a 'form' that has the appearance of a Palm screen, drag an item onto the form to create it then alter any properties.

A screen shot is included (Figure 9), showing the commonly used components. These are the constructor menu bar, the list of current resources for the project and the catalogue of user interface components the installed SDK will support.
Figure 9 Constructor, with common interface elements displayed.

Note that each screen element is given a numeric label - the index. Also, the form itself has an index, with items appearing in a hierarchy within it (1000 being a form index, 1001, 1002 indices of objects on the form). The properties panel allows the developer to set initial properties for the resource, which can then be modified in the code.

Fields:

Fields store a text string for display. Fields may be active (able to be manipulated by the user) or inactive (visible, but not able to be adapted by the user).

When attempting to display numbers, this can be tricky. There is an in-built function to convert integers to strings (and the reverse), but not float or double data types. However, there are examples of code to do this in the texts currently available (Mykland 2000, Ch 12 or Mann and Rischpater 2001, Ch 3).

As writing numbers to fields was a task completed regularly, separate methods to handle this were created. For example, it is common for numeric values to be displayed, or the contents of such a numeric field manipulated. The following code block describes the method developed to handle this task simply.

```java
/*
This method allows numeric field to be altered or valueFormUtils set. If
showvalueFormUtils is 0, then the indicator valueFormUtils will be either
a negative valueFormUtils (subtract 1) or positive (add 1) to the existing
valueFormUtils

If the existing valueFormUtils is null, then it will be defaulted to 0

If showvalueFormUtils has a valueFormUtils (not 0), then the field will be
set to that valueFormUtils.
*/
```
If both showvalueFormUtils and showIndicator are 0 then field will be set to 0

*/

Boolean alterFieldValue(FormPtr showForm, UInt16 showField, int showvalue, int showIndicator)
{
    // set up variables
    UInt16 index; // general purpose index - local
    FieldPtr field; // Used for manipulating fields
    CharPtr psrc; // Points to the text in //the resource
    CharPtr textPtr; // used to point to the //text of a field
    char localText[3];
    MemHandle localHText; // Handle to the text in //our edit field
    // now do it
    // get an index to the field
    index = FrmGetObjectIndex( showForm, showField );
    field = (FieldType *)FrmGetObjectPtr( showForm, index );
    // get pointer to the field object
    FldEraseField( field ); // clear the display
    // get pointer to the field text area
    textPtr = FldGetTextPtr( field );
    //get current valueFormUtils
    StrCopy( localText, textPtr );
    localHText = MemHandleNew( HTEXT_SIZE );
    //create a handle for the text so it can be manipulated
    if( localHText == NULL_ )
    {
        showvalue = 0;
        return( 0 );
    }
    if (showvalue ==0)
    {
        if (showIndicator<0)
        {
            //convert to int
            showvalue = StrAToI( localText );
            //subtract 1 from the valueFormUtils
            showvalue--;
        }
        if (showIndicator>0)
        {
            //convert to int
            showvalue = StrAToI( localText );
            //subtract 1 from the valueFormUtils
            showvalue++;
        }
        else if (showIndicator==0)
        {
            showvalue = 0;    //set to 0
        }
    }
    // ensure negatives don't happen
    if (showvalue < 0) showvalue = 0;
    //convert to text
    StrIToA( localText, showvalue );
    // point to the new handle
    psrc=(CharPtr)lockTheHandle( localHText );
    // put the new valueFormUtils in
    StrCopy( psrc, localText );
    // point the field to the new text handle
So what does this do?

First, a series of local variables are defined. These will contain the field name or designated description (pointing to the field resource on the form itself). \texttt{textPtr} is a pointer to the current text in the field, and \texttt{psrc} points to the new text (once defined).

All other values necessary are passed into the method. \texttt{showForm} points to the form on which the field resides, \texttt{showField} is an index to the field itself, \texttt{showValue} is a new value to be displayed (or is 0 when the current value is to be manipulated), and \texttt{showIndicator} indicates what is to happen to the current value (a negative value indicates subtract 1, a positive value indicates add 1, and 0 indicates no change). If both \texttt{showIndicator} and \texttt{showValue} are 0, then the field text value is set to 0.

Some preliminary work is done. The field index is established, then a pointer to the field obtained. This is done here as it is easier to obtain the designation than pointers, reducing work elsewhere in code.

\texttt{FldEraseField()} clears the display. Now we can get a pointer to the current text value for the field. Finally we can set up a new memory chunk, using a handle in this case, so we can manipulate text.

The next section of code does the conditional work, establishing what value is to be shown dependant upon the current text value of the field, \texttt{showValue} and \texttt{showIndicator} (as described above).

We can now convert the numeric result back to text, get a pointer to the locked handle, copy the new value to it, set the fields' text handle to the new value, unlock the memory, and finally display the new value.

As stated above, this may not be the most efficient way to achieve this, but it is easy to read and can be extended in any way desired (for example, altering the increment amount to being the value of \texttt{showIndicator}).

\textbf{ScrollBar resource:}

The following code is modified from that suggested by the SDK documentation. Normally scrollbars are associated with a form or list and are used to alter the position the window views (Mykland 2000, Ch 8). However the resource can be used to determine a value on a continuous scale - reflecting the analogue or continuous nature of objects in the real world.

This meant a slight alteration in how the scrollbar resource is accessed. Lots of pointers and handles are required which is kind of messy. However, once these have been set it is quick and painless to get the current value or position (which is what we are interested in). The current value can then be displayed or used in calculations.

\begin{verbatim}
Int genericScrollGet(ScrollBarPtr scrollPtr){
    //ScrollBarPtr scrPtr;    // scroll bar pointer
    Short* valuep;
    Short* minp; // not used
    Short* maxp; // not used

    // access the scrollbar
    valuep = (Short*) MemHandleLock( scrollPtr);
    *valuep = ...;
    MemHandleUnlock( scrollPtr);
    return(*valuep);
}
\end{verbatim}
This is written as a generic means of getting the current value (position) of the carriage in a sliding bar. By making this generic, it is possible to get any value we need at any time by simply passing the unique ID of the scrollbar.

The index value is used to get a pointer to the scrollbar. We need to set handles and pointers for each of the components within the scrollbar structure - a pointer to the current value (or position), the minimum value, maximum value and page size (the number of steps taken when page scrolling).

With these set we can call the `SclGetScrollBar()` method. Although we have had to set up pointers to all of the values, we are only interested in the current value that can now be put into a variable.

Having done all we want to at this time we can unlock all the memory handles, freeing memory.

**SpinControl:**

The spin control is made up of three items; an 'up' button, a 'down' button and a display field. Having set up the elements there is only a small amount of handler code required. The buttons are set as repeat buttons. This allows the user to alter the value by as few or as many as they wish - allowing a high degree of control over the value displayed in the field.
case ctlRepeatEvent:
{
    // set up pointers etc
    UInt16        buttonID;
    buttonID= event->data.ctlSelect.controlID;

    // set value
    PMTSpinUpRepeatingIndex = PMTSpinUpRepeating;
    PMTSpinDownRepeatingIndex = PMTSpinDownRepeating;

    // now we have to differentiate between items
    if (buttonID == PMTSpinUpRepeating)
    {
        // call change value method
        alterFieldValue(currentForm, PMTAgeField, 0, 1);
        // get result
        PMTCalc( currentForm, PMTAgeField, LocationAdjuster,
            ProfileAdjuster );
        break;
    } // end SpinUpRepeating

    if (buttonID == PMTSpinDownRepeating)
    {
        // call change value method
        alterFieldValue(currentForm, PMTAgeField, 0, -1);
        // get result
        PMTCalc( currentForm, PMTAgeField, LocationAdjuster,
            ProfileAdjuster );
        break;
    } // end SpinDownRepeating

    return(false);
} // end ctlSelectHandler

Listing 3 – Spin Control

A number of variables need to be set. The index variables are convenient to be set as locals for the event handling method with the buttonID index local to the repeat event.

The if-statements differentiate between up and down buttons and field value changes are made accordingly. Because we are only catching indices there is no requirement to set up pointers or handles here - all is set within the alterFieldValue() method.

Finally, this is related to the PMT roof type, so there is a call to get the result having changed the value of the spin control. We don't have to worry about displaying results – this is taken care of by the PMTCalc method.

List boxes and pop-ups:

Pop-up list boxes are quite different in operation to list boxes found in other developments. They are treated as separate items by the OS not as a single entity as would be found were you developing in MS Access, C++ builder etc.

Firstly, there are 2 resources to be built - the 'pop-up' list and the list box itself. Constructor will not allow a list box to be created that does not contain at least as many items as lines in the list box. It is also recommended that the box be placed in such a way as it covers all of the pop-up resource (Figure 10).

case popSelectEvent: //check value for profile
{  
    switch (event->data.popSelect.selection){
    case (PROFILE_DECRAMASTIC):
        ProfileAdjuster = 1;
        break;
    case (PROFILE_FISHER):
        ProfileAdjuster = 1.4;
    
} // end popSelectHandler
```c
break;
case (PROFILE_HARVEY):
    ProfileAdjuster = 1.2;
    break;
case (PROFILE_DECRABOND1):
    ProfileAdjuster = 0.8;
    break;
case (PROFILE_DECRABOND2):
    ProfileAdjuster = 0.6;
    break;
case (PROFILE_OTHER):
    ProfileAdjuster = 1;
    break;
}//end switch

/*
Each value should match the globals defined - Each must match the list correctly. I personally don't like this, but it is much easier to code for the purposes of a prototype than using database. Also, altering the order is not so difficult, if manual, task. Ultimately, however, the database is to be preferred when it can be complemented with some update application on desktop. Then the update can be done when the hand-held is hot-synched the next time.
Sam
*/
```

**Listing 4 – List Boxes and pop-ups**

Once the resources are built, there are two pieces of code that need to exist. The first is a global definition for each list item that is available for choice. This helps when actually doing work with the value chosen, particularly for decision trees.

![Figure 10 Pop-up box](image)

The handler code becomes more complex
More memory is made available for the application
Independent file updates can occur (i.e. new items can be added at the desktop database and/or provision made on the Palm device for file maintenance)

Other useful things:

As previously mentioned, modular code is particularly effective when developing in PalmOS given the limited resources available to the user. Modular code allows smaller run-time applications to be built.

Modular code also allows developers to make improvements over time, without affecting code in multiple locations. Fixes in one place are preferable to fixes in multiple locations.
It is particularly important to release resource handles when finished. There are two schools of thought on this - setting and releasing resource handles takes time, therefore slowing your application. Global variables are faster but take memory, a condition that must be balanced against the limited memory available in the tool.

Ultimately the developer must make some judgement about what is best for the application. In the examples above, anything locked is released as soon as possible after work has finished. Depending upon the number of form resources, it may be better to create all handles on form entry and close all on exit - a successful method where a form is set as modal and there are limited numbers of controls.

**CBIS Code Changes**

This section describes alterations required to the CBIS code-base in order that CodeWarrior could compile and run the ANN software.

One good thing about this code-base is that the use of many separate classes enables many small memory chunks to be used. This is useful as PalmOS allocates small memory chunks. However, there are minor changes required since the code was originally compiled using the GCC tool chain. The CodeWarrior tools, although ANSI compliant, still have a small number of type-safe components that cause problems within the CBIS code. Once these changes have been affected, compilation can be achieved.

**Friend**

In C/C++, the friend keyword is used to include classes that have common methods, and where other variables may be used interchangeably. While this is implemented in the PalmOS environment, the keyword class must be included - the command changed from

```
friend <classname>()
```

to

```
frend class <classname>()
```

**Int and Index**

The Int data type is accepted as a keyword, but causes problems when used as an Index. In order to alleviate this problem, Index instances of Int were replaced with a new type Index. This was done as the two had been type-classed identically. This ensures that a single definition is used for all data of the type Index.

**Void vs. NULL**

Null is generally assumed to have the value void. However, this causes problems when initialising pointers - null is not a permitted value. Instead, the keyword NULL_ was defined to have the value zero, thereby bypassing this problem.

**NIL vs. 0 (zero)**

Similar to the void/null issue above, nil and zero have a similar issue in practice. Using NIL points to a specific memory address, where zero is a numeric value. In compilers with less stringent type-safe checking this is a commonly used means of initialising or checking for no current value assigned. However, within the CodeWarrior environment it was necessary to replace NIL with Zero.
Assert.h

Assert.h is a file used widely in the error-handling process, particularly in design and test phases of C/C++ development. This caused problems when developing the original prototype in Borland C++ Builder - the solution was to create a new dummy version of Assert.h.

However, overwriting assert.h caused deeper problems in CodeWarrior, so a version was written that returned the new NULL_ value instead of a Boolean. This new version was named new_assert.h, to overcome issues regarding using any other (usable) versions of assert.h.

A note - when converting code from CW6 Lite to CW7 full, the duplication of Assert.h caused other difficulties. Reverting to the Assert.h supplied with the MetroWerks Standard Library (MSL) overcame many of the original and subsequent issues.

Extern handling

The extern() statement is usually associated with the ability to call methods in pre-compiled code. In PalmOS, this caused code-sharing problem. This was resolved by explicitly stating any type conversions (casting) that needed to take place.

Macros

There was a necessity to turn off some macros programmatically, rather than through the IDE settings. In most C/C++ IDE's there is the option to do this globally, so that it is applied to all programs compiled. However, with CodeWarrior there is no clear means to achieve this. Instead, macros need to be turned off programmatically. To this end, the following lines of code were included in a file named 'CWUtils.h'.

#define CORBA_DISABLE
#define NEURALNET_PARTIAL
#define FUNN_ENABLE
#define FUNN_PARTIAL
#define DATAOBJ_DISABLE
#define NULL_ 0

This example caters for the situation described above - where a 'null' default value is needed for things like pointers (where the Index data type is not appropriate).

There are other conversion issues. These are covered well in Mykland (2000, Ch 14) which discusses the porting of existing application code.

Install on handheld

Once the Palm prototype was deemed to be satisfactory, it was given to experts for comment. In particular we were interested in the UI as being appropriate for their needs and that the results provided by the ANN were both timely and accurate.

The program itself was installed on the handheld in exactly the same manner as any other Palm application. That is, a .prc file was created, which was included by the Palm desktop program. The application is then installed and made available at the next hot-synch.
Feedback

In order to gauge the effectiveness of the tool, a questionnaire was created to gather feedback.

Two roof maintenance experts were available in Dunedin at the time this was made available, so there is insufficient basis for statistical measures to be made; however there are some common themes noted.

The first is that the interface and how the tool worked took but a few minutes to learn and try. This is because the tool is designed following existing paper forms, the paper forms having been refined over several years.

Some refinement of results can be expected. Particularly once a database of existing roofs is available then the process of updating the ANN's can be automated, improving the accuracy of results.

Our experts agreed that the use of the Palm tools simplifies their work; mostly by making it safer to climb onto the roof and that results are immediate.
As there has been some input by experts, there was no problem with decision making being hidden. The input/result mapping is also obvious to the Roof Maintenance Expert, so underlying processes are not considered at all while using the tool.

Completed questionnaires are included with this document as Appendix B – Completed Questionnaires.

The experts were presented with a list of potential improvements (presented with a user perspective). They were asked to rank these suggestions, and propose any other suggested improvements. The ranked results were:

<table>
<thead>
<tr>
<th>Suggestion</th>
<th>Expert 1</th>
<th>Expert 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatically calculate price of work recommended</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Automatically calculate the roof size (area)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>The ability to draw and save a picture of the roof</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>The ability to add and save comments</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Store and recover information for future visits</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Take pictures of the roof</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

So the ranked list would appear:

1. Automatically calculate the roof size (area)
2. Automatically calculate price of work recommended
3. Store and recover information for future visits
4. The ability to draw and save a picture of the roof
5. The ability to add and save comments
6. Take pictures of the roof

A discussion of how these could be achieved (in a perfect world) follows.
Section 3 - Future work

As a result of development, and the provision of a questionnaire, a number of possible improvements or extensions were considered. These were presented to the experts as a part of the questionnaire

1. Automatically calculate the roof size (area)
2. Automatically calculate price of work recommended
3. Store and recover information for future visits
4. The ability to draw and save a picture of the roof
5. The ability to add and save comments
6. Take pictures of the roof

Some of these items are related. For example, the first two have similar functions. First, to find the area (square meterage) of the roof can be achieved through the use of GPS. By this means the expert can get the exact location of each corner (means are available to generate high degrees of accuracy). By including other points of interest (ridge-lines etc) the exact shape can be established – the roof pitch is one of great importance to the maintenance staff, as particular equipment is used on some roofs, not others. There should also be means to manually enter roof pitch, as some roofs cannot be climbed on easily, A-Frame homes being one example.

Once the square meterage is known, then pricing can be computed. Other factors of consideration include roof pitch, tile breakage and other repair work. With all factors included, the palm can then calculate the exact cost of work recommended.

One possible (indeed desirable) development to be considered is the recording of settings and results from a roof inspection for presentation to the client, for future training of the ANN, and for comparison with conditions should the roof be inspected again at some later date. This would also allow changes in condition to be noted and used in discussion with the property owner, and the charting of change to be used for training staff.

It is also useful to demonstrate change in condition over time to a client. In later years change accelerates and it is not uncommon for considerable deterioration to occur in a very short time for some roof types and profiles.

Once a roof shape has been generated, it is useful to add other features of interest. For example, identifying regions with greater than average wear or loose/broken components can be very useful when discussing roof condition with the owner of a property. Being able to do this graphically can be very powerful. Currently experts sketch the roof and add notes and comments to this.

Other comments may also be useful, particularly if a roof is in abnormal condition – in surprisingly good or bad condition given its location. Usually, these comments form part of a written report on the condition of the roof, and are expanded on when discussed with the owner of the property. By recording such comments directly in the Palm, they can be incorporated with a report created automatically and printed out. Such a report is also more professional in appearance than a hand-written report, as commonly prepared in the field.

Finally, photos can be used to further illustrate the condition of a roof. Currently a Polaroid can be taken onto a roof (when necessary) to capture items of particular interest. This can be a particularly powerful attention grabber where the property owner is unable to get onto the roof or where they live in a different location (investment property managers for instance). Although useful in some instances, our experts agree that it is not an essential at this time.
References
Appendix A – Published Papers

The first extract was published in ANNES2001. The full reference is:

The material has also been presented at the AAAI 2002 Conference in Edmonton, Alberta. This material was streamed for the AI Intelligent Systems Demonstrations thread, where leading edge applications using AI techniques are showcase AI research and real-world applications.

The second extract was published in AAAI-2002 Proceedings. The related poster is also included for your information. The full reference is:
Abstract

Roof maintenance is an issue that has ongoing ramifications for property owners, all over the world. Determining accurately what maintenance should be taken is difficult, with often-conflicting evidence further complicating the task.

A system has been developed for use by Roof Maintenance Experts. The expert can input information about the condition of the roof then a Fuzzy Neural Network makes an assessment, returning probable roof maintenance options.

This is a non-trivial problem from the real-world domain. Often, many combinations of possible maintenance can occur and, as individual parameters change, so does the prime (or most important) option.

A fuzzy neural network system was developed for assessment, running on a handheld device that could be taken into the field by a roof maintenance expert.

Keywords: fuzzy systems, fuzzy neural network, GUI, integration, roof maintenance, expert system

Introduction

The application of Artificial Intelligence systems to real-life problems is not new. Many people are familiar with fuzzy controllers of lifts, automobile transmissions and air conditioners. However, these are seen as being 'high technology', only for the elite, or as a novelty.

Before widespread acceptance of Artificial Intelligence systems by mainstream computer users or system developers will occur, they must be:
- robust
- accessible, and
- seen to have more than novelty value. That is, be useful in solving a problem that would otherwise be considered a difficult problem to solve.

In addition to these issues, it is desirable to simplify a task that is technically complex.

Often the use of technology needs to be transparent or ubiquitous – users don't care about how the job is done, only that it is done. The advent of the palm-top device has meant that this type of computing power is now relatively portable, inexpensive, and accessible.

Any system developed for palm-top devices needs to be fast and compact, the aim being to provide a portable, real time, software 'tool' for daily use. In order to meet these combined conditions, a novel, real world, problem was approached.

Why a Palmtop device?

In the beginning, there was paper. Paper is useful for many reasons - it is cheap, made from renewable resources, can be recycled. People are fettered only by artistic ability. Anything that can be represented in imagery or text can be captured.

However, there are as many drawbacks. When gathering data, forms can be used, and often people using forms do not think outside the square(s). More importantly, additional tools are required in order to make the paper truly useful - a clipboard or other portable writing surface, a pen or pencil. If the user is in an out-door situation the paper can be rendered useless by weather conditions like wind or rain.

Exceptions to these situations abound - at some significant cost. Use of plastics and specialist pens or pencils were popular briefly. Waterproof notepads or folders are also common. But all are cumbersome and inconvenient in many instances.

Take for instance our intrepid roof maintenance experts. When climbing a roof, the expert has to carry a folder or clipboard, a writing implement, perhaps even a tape measure. On a windy or slightly wet day this becomes somewhat dangerous.

Now picture a different situation. The expert climbs the ladder with both hands free. Points
of interest are recorded and notes written directly onto the surface of a device the same size as the experts' hand. They visit all corners of the roof, gathering point data using an integrated GPS system. This device, when finished with, can be slipped back into the pocket from whence it sprang.

Science fiction? A little Trekkie? Well, no actually. In fact, just this sort of scene is played out in many corners of the world now. People everywhere are beginning to use tools that are specialised for their work that work well, and don't pretend to do everything. In fact the basic premise behind the Palm platform is simplicity. Jeff Hawkins, the motivating force behind the Palm phenomenon, recognised two prime forces - people are smarter than computers and that a palm sized device will be treated as an extension of your desktop, not a replacement [1].

How does this relate to our friendly roofing man? Well, if a Palm sized computer has the ability to take notes, allow pictures to be drawn, can be connected to a GPS, and maybe even intelligent enough to do some of the work, then there is no reason why the small device cannot replace the increasingly dangerous and inconvenient pen and paper. When back at the office, the data collected can be sent to the desktop computer and any reports can be printed - more professional than handwritten reports!

**The Problem:**

Determining the condition of a roof can be difficult, especially for those not having a technical background. Deciding the most appropriate course of maintenance for a roof is dependent on a number of factors, a combination influencing the final decision. Often there is not a sufficiently high correlation for any one condition to be used, making the final decision complex.

A Fuzzy Neural Network system has been developed that returns appropriate maintenance options. This system builds on work begun in 1999 [4]. The FuNN used is that available in FuzzyCOPE 3, as developed by the University of Otago [3].

**Method**

Neural networks are excellent tools for divining rules from a data set [2], as was the case in this instance. As there was no rule-set to begin with, training of the network was imperative for the success of the project. To this end, an expert was engaged to determine expected input properties and build a suitable test data set.

Random input values were created, based on the expected inputs. The expert then assessed these inputs to give a determination of the expected output. The resulting data sets were used as the training and test values for the FuNN. Having formatted the results, training data (75% of the total data set) was passed through the FuNN. The remaining 25% was used to test the FuNN.

Once training was complete, rules were extracted from the FuNN. These rules were then formatted, coded and inserted into a prototype desktop system.

**Expected results – maintenance options and conditions**

Each roof type has different properties, maintenance requirements and wear characteristics. Possible input parameters identified by the expert are listed as in Table 1, along with the possible maintenance options for the roof.
In real life, it is possible that more than one result may be appropriate for a particular roof. Fuzzy systems have the ability to determine when more than one result may be appropriate, making this a useful technology to use for this problem.

It is also possible that some inputs may appear contradictory – another condition for which Neural Networks can be trained.

It is possible that input and result values may have a high or low degree of membership. Input values can be represented by the position of a 'slider' (refer 'Growth' or 'Wear' sliders in Figure 1). The possibility for multiple output options is also displayed in Figure 1.

Input and output values tend to fall on a non-linear continuum. For example, a roof may be described as being 'very worn' rather than 'worn'. The result could be 'Definitely Resurface the roof', falling between 'Re-surface the roof' and 'Replace the roof'. Taking time as one axis, the action of weather will affect the alternative axis (whether that be the amount of wear exhibited, amount of fungal growth etc).

This system also takes into account two other parameters – 'Location' of the roof and its 'Profile'.

Location (refer Figure 2) plays an important role in the life span of a roof. For example, a roof in Invercargill has a dramatically shorter life than the same roof in Dunedin.

The profile (Figure 3) also plays a part (although generally of lesser importance than location). There are, however, some notable exceptions that need to be taken into account.

A desktop-based prototype was built using Borland C++ builder. The prototype, in incomplete form, was then given to a group of roof maintenance experts for comment. Most of the comments received were expected given the incomplete nature of the system, but one logic error was noted. After completion, the prototype was again submitted for expert assessment, with favourable comment.

<table>
<thead>
<tr>
<th>Concrete Tile</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Result</td>
</tr>
<tr>
<td>Wear</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>Growth</td>
<td>Clean</td>
</tr>
<tr>
<td></td>
<td>Re-surface</td>
</tr>
<tr>
<td></td>
<td>Replace</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pressed Metal Tile</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Result</td>
</tr>
<tr>
<td>Chip Loss</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>Base Damage</td>
<td>Clean</td>
</tr>
<tr>
<td>Rust – Pans</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Rust – Valleys</td>
<td>Replace</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unpainted Iron</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Result</td>
</tr>
<tr>
<td>Surface Oxidisation</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>Point rust – white</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Point Rust – Red</td>
<td>Replace</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Faded ColourSteel</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Result</td>
</tr>
<tr>
<td>Oxidisation – Colour</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>Oxidisation – Base</td>
<td>Re-surface</td>
</tr>
<tr>
<td>Rust – white</td>
<td>Replace</td>
</tr>
<tr>
<td>Rust – Red</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 – Possible Input and Output values. Note: Input and result possibilities are not related in any way.

In order for a software tool to be used successfully in the Palmtop environment, suitable consideration needs to be given to:
the small screen size (a PalmOS device is limited to 160x160 pixels),
- limited ability to input text on the handheld,
- seamlessly synchronise to a program on a desktop computer,
- be small, and
- be fast. [5]

PalmOS has been developed with the use of the tool in mind. It complements the need for applications that are fast to load, simple to use, and can be considered an extension of the desktop computer rather than a replacement for it.

Language and built-in control limitations also exist. This means that compact, accurate, code is necessary. Taking these factors into account, the desktop-based prototype was used as the basis for a further prototype built for the PalmOS tool. Palm-top devices hold great promise for work improvement over time, in particular for those gathering or generating data in real time, as in this instance.

Code size limitations in the Palm Environment

A major limitation to be overcome relates to the actual memory size found in a Palm device. Many devices have 2MB of memory, while newer, more powerful, examples have 8MB or more.

In order to control the use of memory, strict code limitations have been implemented. In most instances, 32kb code chunks limit is imposed (although later devices can explicitly use larger sizes). This is implemented through a process known as 'code chunking'. This is a means by which an entire project is split into several pieces. Those components, or chunks, then pass messages between them, as described by Object Oriented design specifications.

In this project, it is expected that more powerful machines would be used, as further development work is expected to take place. However, it is prudent to maintain tight code in the interests of maximising use on any PalmOS machine.

The desktop prototype took up 4mb with 4 Neural Networks running concurrently. Considering the GUI overhead, a not inconsiderable 3.2mb, this was considered acceptable.

The PalmOS front-end resulted in two 30k code chunks, with the balance of memory allocation devoted to the Neural Network processes.

There was also a conscious decision made to limit operations to a single Neural Network at any given time. This allowed two things to be done:

- a re-use of common code meant that there was a much smaller stored memory overhead, and
- faster processing as fewer processes will be run at any given time.

The resultant code runs in as small a memory space and as quickly as possible while still giving accurate results.

Results

The completed prototype confirms that Artificial Intelligence systems can solve real world problems. Combined with advances in hardware, a truly useful tool was built.

As a result of this development, roof maintenance experts (most of whom are self-confessed technophobes) are now becoming more excited about the possibilities
technology can bring to their work environment.
Further developments should eliminate the need for clipboards and paper having to be taken onto a roof. The integration of a GPS device would limit the need for tape measures. The option to save results and roof details in a database for the next visit is potentially very useful and deserving of further development work.
Automated pricing of maintenance options, based on results, is also possible. This can save the expert time, effort, and be more accurate with calculations.

**Conclusions**

Roof maintenance rules are not easily determined by industry experts, or comprehended by trainees. The creation of an 'automated' tool has the potential to greatly decrease the time required to train new staff members.
Ultimately, Artificial Intelligence systems can become an integral part of work processes and overall work systems. In particular, the ability to save-to and draw-from databases appeals to data integrators.
The tool developed simplifies the process involved in assessing the condition of a roof, a process currently undertaken by a roof maintenance expert. By providing useful hardware and software tools, people become used to small, fast, robust and accurate Neural Network systems. Familiarity is likely to lead to greater acceptance of all Artificial Intelligence systems by general computer users.
The creation of an Artificial Neural Network that can be implemented on a Palm-top device also opens possibilities for more complex developments in the future. This particular model, where the Neural Network is built separately then 'attached' to a front-end, means that updating the Neural Network can be done independently of front-end requirements.

**References**


Fuzzy Neural Networks in a Palm Environment

Samuel Moyle and Michael Watts

Department of Information Science
University of Otago School of Business
PO Box 56
Dunedin, New Zealand
smoyle@infoscience.otago.ac.nz, mike@kel.otago.ac.nz

Abstract
This paper outlines the achievements made in the area of small expert systems, in particular the use of multiple Fuzzy Neural Networks (FuNN) within a single application implemented on a PDA. Also discussed is the opportunity for using the architecture as a generic problem solving method – if a Neural Network is an appropriate solution to a problem then a PDA based implementation becomes possible.

Introduction
This paper commences with an overview of the problem addressed. The proposed solution is outlined, as is the use of Fuzzy Neural Networks in the building of a small expert system. The generic architecture implemented, and the reasons for using such a structure, is discussed. Finally, the options for future development are outlined, as proposed by expert evaluators and system developers.

The Problem
Determining the condition of a roof can be difficult, especially for those not having a technical background. Deciding the most appropriate course of maintenance for a roof is dependent on a number of factors, dependent upon the roof type under assessment. The combination and severity of factors is used to determine what maintenance is recommended to the property owner. Often there is not a sufficiently high correlation for any one condition to be used, making the final decision complex.

In some instances input values may be contradictory to the result expected. This complexity makes training new staff difficult, especially where the staff member has little building, technical or engineering background. It is likely that more than one result is valid and it is often difficult to establish which is to be the primary decision.

Currently, maintenance experts need to climb onto the roof to be assessed. The expert must have a folder, pen, and often a tape measure. This makes the act of climbing onto the roof hazardous, an issue identified by OSH\(^1\) (2000). Ideally, an expert need only take a palm-sized device able to accept input. This device can be stored in a pocket until needed thus freeing hands for more important purposes (like climbing ladders).

The proposed solution
To enable novice maintenance staff to quickly become trained, it is useful for the tool to make decisions given the available inputs. Before climbing onto the roof, the expert knows:

- Where the roof is (location)
- How old the roof is (age)
- The roof profile

With this information the system should be able to 'guess' what the roof maintenance expert will see. The expert should also have opportunity to alter the parameters to reflect what is actually found, should they be different from that expected by the system. The system should be able to take the new parameters and use them to make an assessment of preferred maintenance.

Determining Suitable Results – the underlying expert system
Having determined that a palm-sized tool is appropriate, and that the user interface can be created in a compact manner, the underlying expert system needed to be developed.

Fuzzy Neural Networks (FuNN) (Kasabov et al, 1997) are excellent tools for divining rules from a data set. As there were no initial rules, accurate training of the network was imperative. Rules were extracted and assessed by an expert to confirm suitability. Another advantage is the FuNN is capable of providing clear secondary results. Priority is established by mathematical ranking. For this work the FuNN found in FuzzyCOPE 3 as developed by the University of Otago (Watts, Woodford and Kasabov, 1999) was used.

An expert was engaged to determine expected input and results. The test values consisted of a set of randomly generated numbers that were assigned results by the expert. This data set (inputs and expected results) was used for training and testing

\(^1\) OSH is the Occupational Safety and Health division of the Department of Labour. They are charged with ensuring that New Zealand businesses comply with workplace safety standards.
of the FuNN. The total set was divided into two sets – 75% used as for training, 25% for testing.

Why a generic structure?
An important factor in this development is that the use of new technologies should simplify work done. In this instance we wish to take advantage of Palm technologies in creating a small, real-time, expert system.

Palm devices are small, not just in physical size, but processor power and screen size. Memory is a constraint, as memory code chunks are 32k in size. This means that, where possible, work is broken into small pieces and allowed to reside in separate memory areas. There is a limitation in the relative distance that memory calls can be made within dynamic memory – also 32k. Note that devices with more than 2mb RAM may be implicitly instructed to use a larger memory chunk size.

This memory allocation problem was overcome through allocating the generic work to layers - allowing calls to be made between chunks. The system developed uses 4 layers. The visible outer layer is the User Interface (UI). Any PDA User Interface should reflect the work being done and, wherever possible, be simple, intuitive and fast.

The next layer is a translation layer. The translation layer has two tasks. The first is to convert raw values from the UI layer into numeric values appropriate for the neural network. The values returned from the FuNN are translated into sentence form, which are returned to the UI layer for display.

The next layer, the neural network layer, is responsible for taking the neural network input values, bundling them with the relevant connection weight information and passing them on to the generic neural network layer.

The generic FuNN Structure layer is based on the CBIS code-base (Ward et al, 1997), as utilised by FuzzyCOPE3. This accepts the connection weight information and input values, processes them and returns the numeric results to the neural network layer.

In the PalmOS prototype constructed, these layers are in the form of code classes placed in separate code chunks.

This widens the appeal and possible applications that can be built using neural networks. The generic use of components becomes advantageous when there are a number of small, highly specialised, applications sharing resources in a single PDA.

Further Development
After evaluation of the PalmOS prototype, experts have identified areas of development that would make the tool even more useful. These include:

• Saving data for a specific property to a database for future visits.

• Combining the existing tool with a Global Positioning System (GPS) tool. Gathering of data points should enable automatic calculation of the surface area of the roof. This would also eliminate the need for the location to be entered, resulting in better location specific training of the Neural Network.

• Linking in of costing components – automating the creation of quotes.

From a development perspective, it would be beneficial to further separate the underlying expert system from the front end. PalmOS allows this through the ability to store the underlying FuNN structure in the form of a pre-compiled library. The use of libraries enables many applications to use a single library. Code optimisation can further reduce memory use and improve application speed. Only the recall components need be included in the PDA as training can be undertaken on the desktop computer.

The connection weight files can be stored in an independent PDA database. There are a number of advantages in doing this. The desktop computer can update weight files, the Palm database updated when next connected to the desktop. Connection weight files can be updated on the palm device without having to re-compile any application(s) using them. As many connection weight files as are needed can be stored on the palm – enhancing memory efficiency.

For small applications, the translation layer may be combined in the main program component as methods or classes. In order to maintain a generic structure separate classes are used in the PalmOS prototype. Separation provides the advantage that classes can be modified, added, or removed with minimal programming effort.

References


The Problems, then:

How do property owners know what maintenance should be taken when a roof is showing signs of deterioration? Which signs are significant, and which roof? What are the signs of deterioration for a given roof type? What maintenance options are recommended and when should they take place?

These are questions that, usually, a roof maintenance expert is called upon to answer. For those roof types that respond well to maintenance, decisions to be made are often difficult and the costs sometimes contradictory. Training a new expert is time consuming and relies on current work examples, as heuristics will only go so far.

Secondary issues, related to the decisions made in this work, are safety factors. Currently, an expert climbs onto a roof to make an assessment carrying a clipboard, and tape measure. Carrying items is particularly dangerous at the transition between ladder and roof.

Possible Solution? The Working Article!

Consider this situation: The expert climbs onto the roof. Once on the roof, the expert views all corners of the roof, gather point information by GPS. The point information is recorded on a handheld computer, which then plans the roof dimensions, calculating dimensions and pitch. The expert enters the roof type, profile and age, the computer determines an expected condition. Where there is variance, the expert makes changes. These values are used to determine the optimum maintenance plan for the roof, with recommendations passed on to the property owner by the expert once safely down from the roof.

Science fiction? Well, no. The underlying expert system for just such a scenario has been built using fuzzy neural networks (FuNN) running in real-time, on a Palm device.

Why use a FuNN?

The determination to use a FuNN engine came as a result of testing and comparison between a number of contenders:

- Crisp Rule based systems
- Fuzzy Rule based system
- ANNs (Back Propagation Artificial Neural Network)
- FuNN (Fuzzy Neural Network, Back Propagation based)
- SOM (Self-Organising Maps)

The FuNN allowed known heuristics to be checked against the fuzzy rules generated. The ability to train by example, so much the same way as training a movie, was also seen as a significant advantage. However, the FuNN has a great advantage because specific, representative, examples across the possible range (from new to detail) can be used for training. It was also decided that multiple, small, task specific, FuNN's would allow maximum accuracy for each roof type. This also means that any FuNN called upon will require only a small amount of computing resource – necessary in a palm sized device.

PMT

- Chip Loss
- Bitumin Wear
- Corrosion
- Moss and Lichen Growth

Concrete Tile

- Surface Wear/Coating Loss
- Water and Wind damage
- Porosity
- Occlusions

Weight Damage
- Moss and Lichen Growth

Iron

- Corrosion (Chalking, White Rust, Rust)

ColourSteel

- UV Coating Damage
- Micro-Cracks leading to oxidisation (Chalking, basecoat exposure)

Corrosion (White Rust, Rust)
Appendix B – Questionnaire

This questionnaire was designed to gauge the effectiveness of the tool, while finding out what could be done in the future as far as integration with other tools is concerned. Rather than a static questionnaire, this one was designed to have comments made with each question – making the entire set of questions as relevant as possible to the current state of development.

The following instructions were included with the questionnaire:

The purpose of this questionnaire is to help us improve the quality and usefulness of this iteration of the roof maintenance project for PalmOS.

This tool was developed with two objectives in mind. The first is to showcase the usefulness of Fuzzy Neural Networks in the workplace, the second to help roof maintenance experts in their daily work.

Comments are welcomed in respect to all questions, although it is suggested that you read and answer all questions before making comments. It is expected that you would spend about 5 minutes completing this questionnaire.

Again, my thanks for your help,

Sam Moyle
Questions:

1. How long did it take for you to learn the interface of this tool? ............ minutes

2. Do the 'wear bars' suitably represent your understanding of the important roof maintenance factors? ...........................................................................................................................(Yes / Maybe / No)

3. Would these same attributes be better described in the following manner? .......
   (Yes / No)
   
   Example: Concrete Tile - Roof Wear
   
<table>
<thead>
<tr>
<th></th>
<th>Low</th>
<th>Medium</th>
<th>Severe</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Are the results presented as expected? ............................................ (Yes / No)
   [Please record errors/alterations so that they can be updated in future revisions]

5. Rank the following additions to this project in order of importance to you – (1 being most important, 6 being least important)

   - [ ] Automatically calculate price of work recommended
   - [ ] Automatically calculate the roof size (area)
   - [ ] The ability to draw and save a picture of the roof
   - [ ] The ability to add and save comments
   - [ ] Store and recover information for future visits
   - [ ] Take pictures of the roof

6. Given that these implementations were made, do you believe that a Palm sized device will simplify your work? ................................................................. (Yes / No)

7. Will such a tool help you to work more safely? ................................. (Yes / No)

8. Does it matter to you that the tools’ decision-making process is hidden? (Yes / No)
Appendix B – Completed Questionnaires
The following pages are the completed questionnaires from our experts.
Questions:

1. How long did it take for you to learn the interface of this tool? ...... minutes

2. Do the 'wear bars' suitably represent your understanding of the important roof maintenance factors? ................................................................. (Yes / Maybe / No)

3. Would these same attributes be better described in the following manner?

Example: Concrete Tile - Roof Wear

<table>
<thead>
<tr>
<th>Low</th>
<th>Medium</th>
<th>Severe</th>
</tr>
</thead>
</table>

4. Are the results presented as expected? ................................................................. (Yes / No)

[Please record errors/alterations so that they can be updated in future revisions]

5. Rank the following additions to this project in order of importance to you – (1 being most important, 6 being least important)

1. Automatically calculate price of work recommended
2. Automatically calculate the roof size (area)
3. The ability to draw and save a picture of the roof
4. The ability to add and save comments
5. Store and recover information for future visits
6. Take pictures of the roof

6. Given that these implementations were made, do you believe that a Palm sized device will simplify your work? ................................................................. (Yes / No)

7. Will such a tool help you to work more safely? ......................... (Yes / No)

8. Does it matter to you that the tools' decision-making process is hidden? (Yes / No)
Questions:

1. How long did it take for you to learn the interface of this tool? 2 minutes

2. Do the 'wear bars' suitably represent your understanding of the important roof maintenance factors? (Yes / Maybe / No)
   I think there should be some wording to whether tiles are cracking or not.

3. Would these same attributes be better described in the following manner? (Yes / No)
   Example: Concrete Tile - Roof Wear
<table>
<thead>
<tr>
<th>Low</th>
<th>Medium</th>
<th>Severe</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
   |with the above addition - i.e. some tiles cracked?

4. Are the results presented as expected? (Yes / No)
   [Please record errors/alterations so that they can be updated in future revisions]
   Have trouble adjusting age of roof. Also feels stuck in replace roof at low deterioration.

5. Rank the following additions to this project in order of importance to you – (1 being most important, 6 being least important)
   1. Automatically calculate price of work recommended
   2. Automatically calculate the roof size (area)
   3. The ability to draw and save a picture of the roof
   4. The ability to add and save comments
   5. Store and recover information for future visits
   6. Take pictures of the roof

6. Given that these implementations were made, do you believe that a Palm sized device will simplify your work? (Yes / No)

7. Will such a tool help you to work more safely? (Yes / No)

8. Does it matter to you that the tools' decision-making process is hidden? (Yes / No)

26/11/2001 10:25