Enhancing Daily Planning Using Artificial Intelligence

Can a machine be solely responsible for a person's daily routine?

Jonathan Tremelling 08003098

Supervisor: Jim Smith

April 2, 2012

Abstract

Due to modernisation and the increasing capabilities of technology, society has become more dependent on intelligent devices. Automation systems are used a lot within industry and large corporations, and now with the use of mobile technologies, the general public.

All software is designed to fulfil some purpose and continue carrying out the same tasks. Some would say that due to autonomous programs, this is not the case. Some software has the ability to learn new things and adapt its functionality slightly, maybe with Amazon's recommended purchases or Google's alternative searches. Although these programs learn to a certain extent, they are still fulfilling one requirement: to find alternatives.

This document describes the uses of such autonomous devices and whether a mobile application can learn to manage a person's daily routine using Artificial Intelligence. This subset to autonomous technologies provides an insight into the possibilities of having a program that is completely autonomous, much like a human being. Having a piece of software that can learn all aspects of a person's life doesn't necessarily mean it is autonomous in every sense of the word. The system would need to be *self-healing* to be truly autonomous. Having a mobile application download other services specifically for an individual, or a machine that can automatically schedule meetings, order transport and book accommodation shouldn't be out of our grasp. When

a machine can learn to not only adapt to its environment, but also fix defects in its own design, the technological world we live in will never be the same again.

Contents

Chapter 1

Introduction

1.1 Background

The modern word is full of tasks and schedules, which dominate the society of today. Everything is planned and structured, diaries are used, notes written and dates logged. Our lives have sped up and we rely on technology more than ever, to get us through our daily routines. Whether it be the smart functionality of a mobile phone, or the convenience of an MP3 player, we have allowed ourselves to let technology run our lives. We have become technology addicts, obsessed with hand held marvels managing our every needs, eagerly awaiting the next quick fix.

Mobile phones have become the dominant species of technology over the last few years managing the functionality of MP3 players, personal organisers and much more. They seem to be the ultimate device, managing almost every

need you could think of. Some would argue this is mainly due to mobile applications, having the view that

if you don't have it now, it will be available to download later

. People like to have one piece of technology to manage everything. It reduces the need for lots of devices and enables you to concentrate your efforts on the one, personalised piece of equipment.

With the development of wireless connections, 3G, and cloud computing (Hayes, 2008), devices can be synced over the network, allowing files to be accessed from anywhere with an internet connection. This reduces the need for devices with large memory capacity and processing power can be shared. With files being stored externally, data can be shared among users, saving time and money, due to the need for only one storage area.

1.2 The Problem

Autonomous systems still need human interaction and are not completely independent. Due to the demanding nature of today's society, technology has become more complex to cater for tasks a human being could not manage themselves. A large amount of trust goes into such systems, because a lot of the time they are trying to keep us safe. Such technology is used to manage rail travel and diagnose medical symptoms, as well as in public and military aircraft. A lot of modern fighter plans simply wouldn't stay in the

sky if it weren't for a highly advanced technological system constantly making adjustments to the aircraft's flight.

There are many mobile applications available, managing different tasks and aiding people with their daily lives. Some applications combine functions and can be tailored to a certain person, however their functionality is still limited. Due to the limitation of an application only being useful as long as its purpose is needed, new applications keep being created. I propose the idea of autonomous applications, that can adapt to a person's lifestyle, initially looking at the idea of shopping and task management. This product aims to learn a person's shopping habits and provide warnings of empty stock and over spending. It will also learn repeating tasks and help keep track of events. If the idea of autonomous shopping and task management is possible, other autonomous processing such as learning habits and interests could be achieved in the not so distant future.

Creating one piece of software for each scenario seems quite primitive these days and with technology continuing to grow, more systems will need to be designed. Research into machines that can self-adapt has been developed for years and the recent solutions currently being worked on give hope for designing one system that can be trained for different scenarios, instead of hard-coding for each specific problem.

1.3 Product Expectations

The purpose of this report is to discover aspects of artificial learning and ways it has been applied to computing. The mobile application that will be produced is meant as a small example of an autonomous system, discovering user information for itself. It will be a basic example of a self-learning machine, processing user shopping habits and evolve stock usage rates. Other features will include storing meal recipes and shopping lists, and the ability to plan these events. The application will learn which items are needed the most and provide warnings of low stock.

The aim of this experiment is to discover whether systems can be designed to learn on a larger scale, such as a person's hobbies and interests. Continuing with the example of mobile applications, one could design a system to discover user interests, such as sport, and download useful services from the android marketplace. Expanding on this idea, a system could be designed to maintain business schedules, arranging work events and booking/cancelling meetings.

There is a lot of research going into the field of autonomous systems and this report highlights different ways of applying this to computing. It is meant to provide an insight into this area of artificial intelligence and how the modern world is being shaped by such efforts.

Chapter 2

Background Research

2.1 Literature Review

2.1.1 Will Wright - Hivemind

Will Wright (Gaudiosi, 2012) is an American game designer who co-founded the game development company *Maxis* (Bhd, 2011) and was the original designer or the successful video game *The Sims* (The Sims, n.d.). The Sims is a strategic life simulation game based around human life. It contains a number of virtual individuals that live in a community emulating the real world. The idea is to build a life for these "Sims" in order to gain money to buy luxuries and move up the property ladder. It is not in real time but everything is in proportion, meaning the player of the game has to learn to save their money, sleep at relevant times and eat accordingly. You can also train your Sims to learn routines and other skills such as cooking and playing

a musical instrument.

Will Wright decided to build on this idea of autonomous play by inventing *Hivemind*, which is a game that incorporates the real world and the player's daily routine. It gathers information about the gamer by

> *tapping into streams of personal information on phones, tablets, social networks and computers(Gaudiosi, 2012)*

building upon Alternative Reality Games (ARG).

The interesting features of this game are the convergent abilities that allow it to adapt to different routines, not being held back by general rules, building upon user experiences. My application aims to give a similar insight into the power of autonomous products, showing that one product can adapt for many different uses. This game proves the technology to achieve this is available.

Wright also developed a television programme called *Bar Karma* that used crowd sourcing to feed development and decide upon certain tasks. Using the community as a development tool like this means that the idea of a product being designed to carry out a specific task may soon be in the past. Autonomous technologies allow their designers to be more creative and not produce something that will become useless in a few years time.

Some criticisms(Rose, 2011) of this game say that it is too ambitious and would need enormous amounts of resources:

> *the resources needed for a project like this would be enormous. I guess he could integrate with Google/Google+ and perhaps Facebook? That's really the only way I can see something like this happening.(Rose, 2011)*

Other people think it is a waste of time an should be used as more of a platform than a game. However this is not the first time a new technology has triggered controversy. After all, nobody thought the internet would grow to such a powerful tool that is used by almost every technological device today, and there are others who think that this sort of idea is already in production:

> *This is already happening. Life is more like a game because our phones have added a game-like interface to life. I don't think it's a far cry to add more game like experiences that fit within the same framework as social networking or location based games like foursquare. I think many people are thinking along these lines already. If you're not then prepare to get left behind again (a la iPhone/mobile).(Rose, 2011)*

2.1.2 Jeff Howe - Crowd Sourcing

> *Crowd sourcing is the act of taking a job traditionally performed by a designated employee and outsourcing it to an undefined, generally large group of people in the form of an open call.(Howe, 2008)*

Due to the development of the internet and new technologies, small businesses and members of the community can compete with much larger enterprise companies. Some early examples of crowd sourcing are *wikipedia*, allowing the general public to upload facts and figures, and share information with other users and the development of Unix. Other websites such as *Ebay* and *Amazon* allow people to buy and sell products at prices competitive with other businesses.

Unix was invented by Dennis Ritchie, Ken Thompson and others in Bell Labs in 1969. It was initially a re-write of another operating system and the first edition was released in 1971, free of charge. Twenty years passed, with 9 further editions being released and new programming languages such as C being produced (developed by Ritchie). In 1991, Linux was introduced by Linus Torvalds, a student in Finland. Over the years this has developed into a highly customisable, open source distribution used and developed by many around the world (Hope, 2012). It has sparked a group of individuals dedicated to maintaining and improving its functionality, and created a divide in the computing industry. Its importance and reason for being so

popular is the fact that there is a community working together to keep and maintain the whole system. Improvements are governed by crowd sourcing (Whitla, 2009) and the operating system is a mixture of societies needs.

Jeff Howe says that it is not always beneficial to employ one person who is perfect for a job role. Sometimes it is more advantageous if a large group of freelancers with different skills manage many tasks, because there is more versatility that way. The group can work as a team to evolve through each new technological era. Autonomous products such as mine and Will Wright's *Hivemind* depend on the community to keep them up-to-date and useful with modern lives. If we allow large communities to teach products, helping them evolve, they can become more tailored to our needs and less resilient to change.

A debate on one website (Crowdsourcing Debate, 2009) highlighted a few problems with crowd sourcing, such as the quality of the results and the sometimes manic confusion generated by such a large volume of participants. There is no guarantee that the crowd will have anyone with any expertise in the field, which will mean the quality of the results will not be of such a high standard as an expert. It can also be hard to manage such a large number of results, making sure the best is picked.

A reply to this article suggested that although the mass crowd agree upon one idea, it doesn't stop the experts within the company overriding that decision with their own professional opinions. This agrees with the view that crowd sourcing is effective, but only if it is managed properly, by experts.

Another point made was the idea of having an entire crowd containing no expert knowledge at all. The reply pointed out that a crowd this size is likely to have expert contributions and therefore keep the end result fairly accurate.

Condorcets jury theorem(J. Austen-Smith, 1996) states that, given a situation with two answers (one right, one wrong), if the probability that each voter in a group will choose correctly, is less that 1/2, adding more members will simply worsen the outcome. So having a large group may not always average to the best decision, unless the probability of an individual choosing the best decision was greater than if they were not.

2.2 Machine Intelligence

Artificial Intelligence has been around for many years, but it has only been in the last 70 years that it could be applied to computing (The History of Artificial Intelligence, n.d.). Ever since Norbert Wiener's contribution to feedback theory in the early 1950's (McGarry, 2009), experts have been developing artificial intelligence to automate increasingly difficult tasks. Weiner's research into the work of the thermostat realised the theory of all intelligent behaviour being the result of feedback mechanisms.

In 1955, two theorists named Newell and Simon developed *The Logic Theorist* (The Logic Theorist and Its Children: AI in Action, n.d.), which was arguably the first program to show intelligence. Its aim was to codify

the principles of pure mathematical logic. The program proved thirty-eight of the first fifty-two theorems and even found a proof for one theorem, which was more effective than one provided by Russel and Whitehead.

In 1957, Newell and Simon developed a new program called *The General Problem Solver*(GPS) (A. Newell, 1961), which was an extension on Wiener's feedback principle, with more advancements in general problem solving. As the years progressed, more intelligent programs were developed, leading to the advent of the Expert System in the 1970's and new theories such as *Minksy's Frame Theory* (P.N.Johnson-Laird and P.C.Wason, 1977) were tested.

As the machine learning theory progressed, it gave ideas for more intelligent agent-based models, such as neural networks and genetic programming, becoming more widespread in the 1990's. Nowadays it is used in many different areas of research, such as the financial market and space travel. One in particular was *Deep Space 1* (Deep Space 1, n.d.) - A spacecraft developed in the late 1990's for investigating high risk areas in space, particularly the comet Borrelly in 1998.

The term *Moore's Law* (Moore's Law, 2012) predicts that the number of transistors that can be placed on an integrated circuit doubles approximately every two years, and has so far been correct. This has led to smaller, faster and more efficient electronic devices capable of performing computation tasks of high complexity. This along with many other advances in hardware and software has shaped the technical world we live in, and meant the capabilities

of systems is increasing much faster than their programmers. Using Artificial Intelligence allows these machines to learn for themselves, at their own rate, completing tasks much faster than a human being could.

2.3 Self-Healing Machines

When designing an autonomous system, dedicated to learning more about its environment, making our lives easier, it shouldn't be expected to need updates or extra features added at a later date. A truly autonomous system needs to learn these things for itself and more importantly fix any defects it may be carrying (Murch, 2004). After all, this is what a human being is capable of.

Google's autonomous vehicle project has had great success and their Toyota Prius hybrids have been released on public roads. On the streets of California, Google's cars have been travelling hundreds of thousands of miles by themselves, however they need some sort of human intervention every thousand miles to prevent a crash. Some would argue that this isn't complete autonomy since they are not thinking for themselves one hundred percent of the time.

> *Think of supposedly automated aeroplanes. I havent seen the situation where the pilot and copilot come back to the cabin and say, Hey get a me a brandy, where the whole thing is on autonomous mode. You have two highly trained professionals monitoring the system, reading out loud to each what theyre going to do next, very diligently checking the mobility apparatus before and after each flight. How often do we even check tire pressure? (Vanderbilt, 2012)*

It is clear from existing vehicles and air traffic control systems that although technology has advanced, human interaction is still not rendered obsolete. Google have said however that they are planning on increasing the time the cars can stay on the road for, before needing to be checked. True autonomy seems unlikely to make an appearance in the immediate future, especially in the public's hands, however it is not out of reach.

DARPA have also been successful in creating completely autonomous vehicles to drive large distances, without the need for human interaction. The *DARPA Grand Challenge* (The DARPA Challenge, n.d.) is a competition for driverless vehicles to travel an off-road course in the shortest time. The vehicles have built in sensors and cameras capable of detecting the shortest route possible through deserted terrain. It is a competition for aiding research into autonomous vehicles and has been running since the early nineties.

Combining machine hardware with biological organisms could be one solution to creating self-learning and self-adapting systems. Scientists at

the U.S. Department of Energy's (DOW) Argonne National Laboratory and North-western University have been developing biological machines powered by bacteria.

> *Researchers have discovered that common bacteria suspended in a solution can be made to turn microgears. This opens up the possibility of building hybrid biological machines at the microscopic scale. (Quick, 2009)*

Although this is only on a small scale and the functionality is basic, this proves that self-adaptive machines are possible and a hybrid system is just one solution. Machines are more commonly modelled on biology and natural evolution because it is the most successful, but complicated theory we know.

2.4 Agent Communication Languages

Intelligent systems such as stock control and online auction agents, capable of predicting complex outcomes and procedures much faster than a human can, are already used today. They do this through BDI (Beliefs, Desires, Intentions) and subsumption architectures (Brooks, 1985) and communication ontologies, such as KIF (Knowledge Interchange Format), which is used by KQML and FIPA (Foundation for Intelligent Physical Agents). KQML is a language for defining performatives that can be used by agents to communicate with each other. Example performatives are "broadcast" (send

information over all connections) and "ready" (ready to respond to previous performative) (Wooldridge, 2009, p.138). FIPA also started work on a program to standardise agents (FIPA, 2009). To aid in agent cooperation, Environments such as JADE (Java Agent Development Environment) (Italia, 2003), which is open source, allow these agents to communicate with such ontologies.

With new ontologies being designed and created, bigger and more complex systems can communicate with each other, achieving goals beyond our reach. This means complex negotiations can be automated and run continuously without the need for human interaction, generating complex solutions that can only be understood using an advanced agent world. Bidding services such as Ebay (Omidyar, 1995) and Amazon (Bezos, 1994) are only a stepping stone towards the capabilities of tomorrows agent-based applications.

2.5 Mobile networking and Applications

The use of mobile phones has increased dramatically since the introduction of smartphones in 2002. The *Treo* smartphone developed by Palm was one of the first to have wireless web-browsing, email access and other personalised applications. This gave its users freedom to perform look-ups, manage meetings and plan tasks away from their home and opened up a window of opportunity for smart phone developers such as Apple and Google. The ability to download applications straight to your phone has meant users can

personalise their own device and developers can now create their own applications to publicise.

The idea of personalising your own device like this has meant that consumers no longer need to wait for their perfect product to be released. The fact that users can rate an application online gives developers instant feedback, as opposed to companies having to create surveys and other research. This also benefits phone manufacturers because they only need to work with one phone platform and not worry about application content.

Personalised applications are great for growing businesses and young artists, because they can be tailored to any design. Useful facilities can be combined and unwanted features discarded, leaving a simple, yet useful product for a niche market. A lot of mobile applications are also free, which attract consumers and are a great way of advertising a product.

Due to the advancement of cloud computing, advertisement has never been easier.

2.6 Stock Control Systems

Managing stock levels and weekly usage rates of shopping items can increase production time and reduce the chance of having insufficient ingredients for meals etc. Stock control systems manage product levels and ensure a minimal amount is stored at any one time, reducing the chance of wasting items and the amount of money tied up in produce.

Stock control, otherwise known as inventory control, is used for tracking stock and determining how much is available at any one time. It ensures the right amount is stored at the right time, maximising sales and minimising cost tied up in unused stock. In order to determine how much of each item is stored, the system needs to assign them a value of importance. The higher the importance the more of that item needs to be stored.

An advantage of using computerised stock control systems is that *Sales Order Processing* (SOP) and *Purchase Order Processing* (POP) can be integrated into the system and stock can be updated as orders arrive. When stock runs out, new orders can automatically be generated and batch orders can be controlled to avoid over spending. Computerised systems can also be used to find the cheapest and fastest supplier, as well as speeding up sales with bar-code scanning and tracking items with *Radio Frequency Identification* (RFID).

InFlow (InFlow, 2006-2011) is a free stock control system for small businesses. It is compatible with most bar code readers. It is multi-language compatible and lets you use multiple units of measurement to price stock. This could be in crates, dozens or individual pieces. InFlow's simplicity means that stock can be ordered with ease and orders can be managed with one motion. The easy-to-use user interface makes it easy to navigate around the system, which will save time and reduce management errors. When items are low, the Reorder Stock button can automatically generate purchase orders for low stock and prices can be controlled using a moving average if they

change. To understand results better, graphs can be generated to represent sales, profits and inventory levels, and reports can be used to show detailed information such as total sales, best selling products, and how long stock will last. This information can then be shared across the network, giving multiple users access at the same time. If needed, data can be exported as CSV spreadsheets, allowing for offline analysis.

There are many other Stock Control Systems such as Inventory Droid (Inventory Droid, n.d.), which is Android's only full Inventory Management App, and HanDBase for all mobile platforms, which provide similar features to that of *InFlow*. A good Stock Control System supports the following (What Makes a Good Inventory Control System?, 2011):

- Counting and monitoring of inventory items
- Recording and retrieval of item storage location
- Recording changes to an inventory
- Anticipating inventory needs, including inventory handling requirements.

(See Appendix A.12)

In order for Inventory Control Systems to be of any use, they need to learn the relevant company's stock levels and how they are affected by consumers. The more they are tuned, the more accurately they can predict changes in stock. The company can then use this information to make important decisions about improving cash flow consumer traffic.

2.7 Learning Methods

Since this project is heavily based on some autonomous behavioural aspect, it is important to choose the right learning strategy. One that requires a lot of past data will not be sufficient, since the application initially has no knowledge about the user's shopping habits. This is why *Neural Networks* and *Genetic Programming* cannot be used, since they are very inaccurate with small amounts of data. If they have too little information, they could get stuck in a local minima or not even find a solution at all. Since all the data is stored on a database and the only assumptions being made are how long items may last and how often they may be used, a simpler learning algorithm would suffice. This approach would be better for debugging, as it could be written in a human-readable format, making it easier to find any problems with the code. Using Android's built-in alarm manager would also be helpful for keeping track of dates, such as planning a meal.

2.7.1 Inference

Inference is the act of generating assumptions based on existing facts. Normally involving IF-THEN rules, it can be used to discover new conclusions from premises assumed to be true.

2.7.2 Deductive Reasoning

Deductive reasoning takes what is know to be true and attempts to prove new theories based on the initial knowledge. It is usually in the form of IF-Then rules and assumes new facts based on the initial discovery. An example of deductive reasoning would be

> "The name of a fruit is determined by its colour and shape."
>
> "If the name of a fruit is determined by its shape and colour, then all round, green fruits should be the same."
>
> "However a grape is not the same as a granny smith apple, therefore the statement is not supported."

Deductive reasoning relies on the initial premise being correct and can generate a very closed environment (Deductive And Inductive Reasoning, n.d.).

2.7.3 Inductive Reasoning

Inductive reasoning is the opposite of deductive reasoning. It generates general rules from many specific instances. An example of inductive reasoning would be if you were investigating the sum of the interior angles in triangles. When a sufficient amount of different triangles prove that the sum of the interior angles total 180, a general rule could be generated stating that the interior angles of all triangles add up to 180.

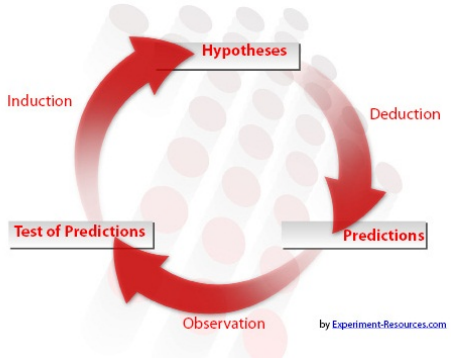


Figure 2.1: Inductive and deductive reasoning

Inductive reasoning is good for when a system needs to gradually build up a knowledge-base of different instances and create general rules as it learns. This is different to deductive reasoning, where the system starts with a set of base rules to infer new rules from.

2.7.4 C&RT

C&RT (Chou, 1991) modelling is

> an exploratory data analysis method used to study the relationships between a dependent measure and a large series of possible predictor variables that themselves may interact. (CART, n.d.)

It is used for designing classification and regression tree's based on different data, optimising each tree's depth to ensure an efficient categorisation process. At each Node, the sample data would be analysed and a decision would be made into how effective it would be to split it into smaller, more precise groups. An example would be classifying living things. The first "branch" may split the dataset into "plants" and "Animals", which may not be very useful. If this is the case, the "Animals" dataset may be split into "Amphibians" and "Insects", which is more specific.

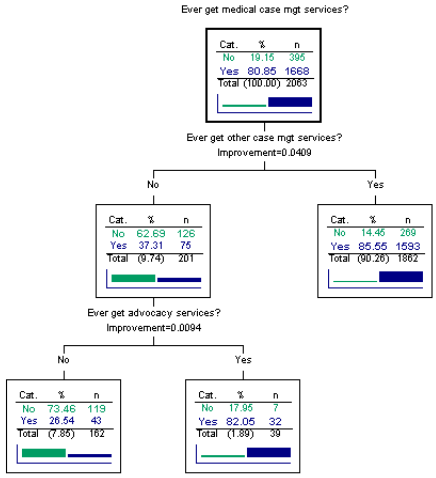


Figure 2.2: An example of C&RT

Classification Tree's are used when the response is discrete and only involves a set number of outcomes. An example of a Classification Tree would

be a Decision Tree. Regression Tree's are used when the number of available responses is continuous/infinite. This would be used for working out varying prices of consumer goods, or predicting the stock market.

2.7.5 Decision Trees

Decision Tree's (Magerman, n.d.) are formed using a collection of variables and associative rules and can be used to categorise certain features of a problem, such as types of species. Decision Tree Learning is used with statistics and data mining for prediction monitoring. It is represented with Nodes and Branches. The internal Nodes hold attributes for decisions to be made upon. The branches can hold features that the internal Nodes may display. These then connect onto leaf Nodes which hold similar information to internal Nodes. The idea is that you would traverse down these branches, passing through Nodes, eventually reaching an end Node, through which a decision can be made.

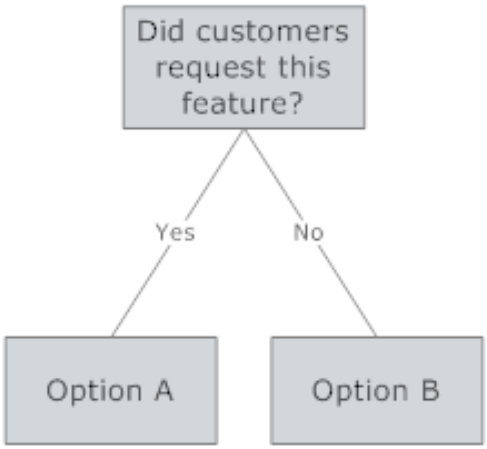


Figure 2.3: An example of a Decision Tree

Decision Trees are normally used when instances can be described by

attribute-value pairs, or on noisy training data. This could be for medical diagnosis or credit risk analysis. Search methods such at depth-first and breadth-first can be used to find better solutions once the tree has been made.

Applying decision trees to a shopping manager could be useful for categorising items to be used when suggesting alternatives. If the user information was shared with others, the search space would increase and some management system would need to be implemented for keeping items ordered. This could then be used to find new stock items for cooking or eating as a snack. An example of this kind of decision tree is displayed in Figure 2.4.

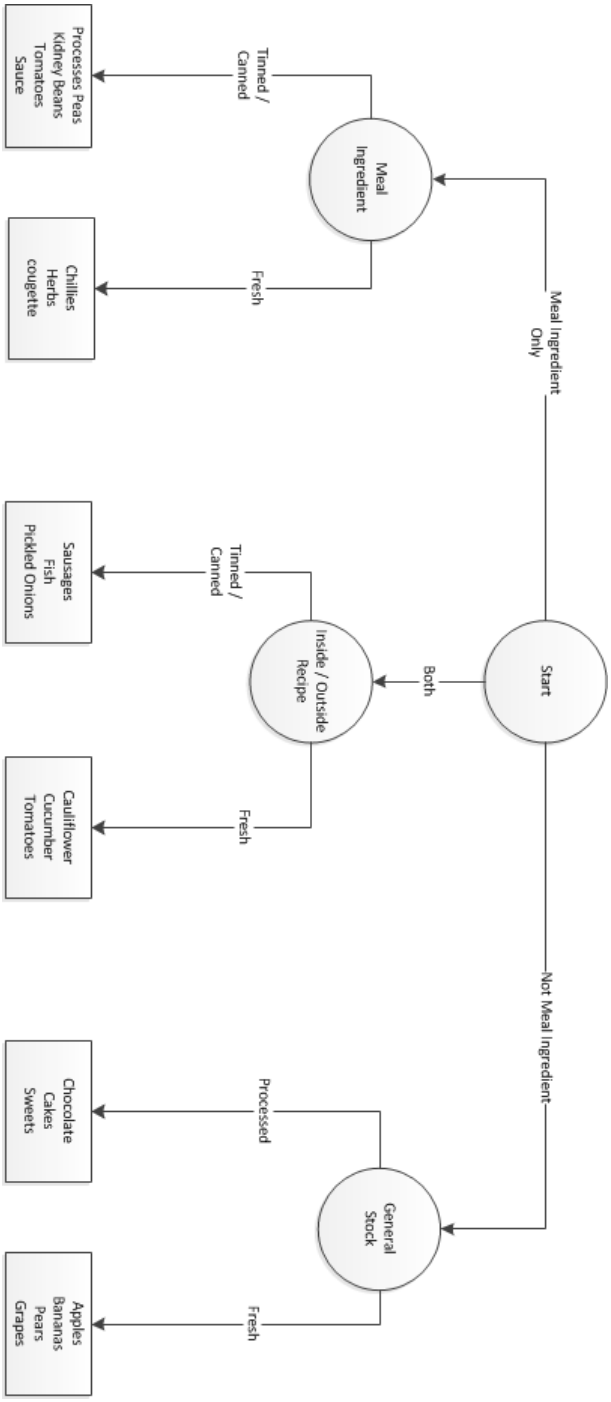


Figure 2.4: A decision tree for a shopping manager

This would also be useful for working out how often items may last. The system could search a shared tree finding the specified item and observe how often others use that item.

2.7.6 Associative Rule Learning

Associative Rule Learning is a way of finding useful patterns between data in transaction databases. This is usually of the form X->Y, where X and Y are two subsets of the data stored. X is called the antecedent, or Left hand side (LHS) and Y is called the consequent or Right hand side (RHS). The stronger the relationship between X and Y, the more satisfying the associative rule is. It is a very useful way of finding strong links between datasets and discovering new relationships.

Associative learning could be useful for finding links between shopping items and meals or shopping lists. It would be a good way of discovering popular items and working out how often they need to be stocked. There will be larger consequences for some items running out of stock compared to others and the system needs to prepare for that.

2.7.7 Learning The Parameters

Learning methods depend on useful parameters and informative premises. Inferred information is only as accurate as the knowledge this information is based on. Useful parameters for deciding on item degradation rates would be

time and quantity, because from that a gradient can be decided upon using quantity over time.

2.8 Applicable Learning Methods

Artificial Intelligence is a huge area of research and the idea of an autonomous system needs to be applied to a smaller scope. A shopping manager will have enough detail for a sufficient learning method to be used, as well as explain the need for autonomous systems. The general concept will involve learning a person's daily shopping habits and how often certain items degrade over time. The system will initially have no concept of what routine it is to learn, having to rely on user input to populate its knowledge-base. The more it is trained, the more accurate its estimates will be. Using this method, warnings can be generated about empty stock and the system can infer other warnings such as whether meals are available based on current ingredients etc.

2.8.1 Linear Regression

Linear regression is a way of modelling the relationship between two variables by applying a linear equation. This could be a graph of people's heights and weights for example, showing the taller the person, the heavier they are generally. Applying this to a shopping manager, two variables could be quantity over time, or usage rate over time. The first graph would show that the larger the quantity, the longer the item would last before needing a refill.

$$r = \frac{1}{n-1} \Sigma \left(\frac{x - \overline{x}}{s_x} \right) \left(\frac{y - \overline{y}}{s_y} \right)$$

Figure 2.5: Correlation Coefficient Formula

The second graph would show that the larger the usage rate, the less time the item would last. There is not always a relationship between two variables, the items price does not correlate to how long it lasts for example. However, it may correlate well to how often it is used. A scatterplot can be used to determine this correlation, gathering a significant amount of data over time and applying a trend line to the plotted points. A trend line can only be applied to variables that have a sufficient correlation. The strength of this correlation can be decided using the correlation coefficient, which is a value between -1 and 1 (Linear Regression, n.d.).

2.8.2 Regression Trees

> *A classication or regression tree is a prediction model that can be represented as a decision tree* (Classication and Regression Tree Methods, n.d.)

Regression trees are similar to decision trees, in the sense that they help categorise datasets, using nodes and branches. However, there is an additional partitioning tool applied to the former. Each regression tree is monitored for correlations between node values.

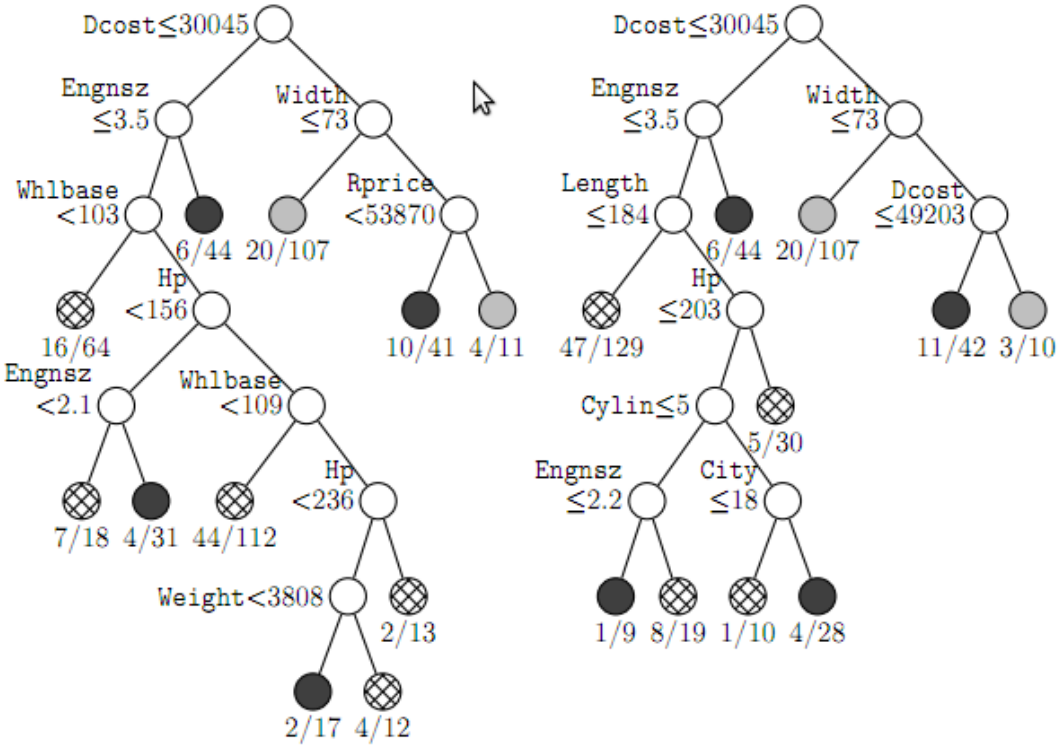


Figure 2.6: Some example regression trees.

Take the regression trees in Figure 2.6. These trees hold attributes of a number of vehicles, such as cost, length and engine size. To form correlation coefficients between these attributes, each node is monitored and related to the others in the tree. If for example a route that traversed right for both Dcost and Width, often ended with a sports car, the correlation between price and type of car would be high.

2.8.3 Hebbian Learning

Hebbian learning is where weights on connections between nodes are adjusted, so that each weight better represents the relationship between the nodes. Nodes that tend to be both positive or both negative, will have strong positive weights, whereas nodes that tend to be opposites, will have strong negative weights. The strength between node A and B increases the more A participates in firing B. The general form of the Hebbian learning rule is defined in section 2.7.

$$\Delta W_{ij}(t) = F(x_j, x_i, \gamma, t, \theta)$$

Figure 2.7: A Hebbian learning rule.

Where Xj is the output of the presynaptic neuron, Xi is the output of the postsynaptic neuron, Wij is the strength of the connection between them and y is the learning rate. This formula takes time t and the learning thresholds into account.

For a shopping manager, Hebbian learning could be used to discover the strength between types of food items, to provide alternative meals or items. It could be a case of working out the strengths between items to form alternative shopping ideas, or new meals based on ingredients commonly purchased.

2.8.4 The Perceptron Learning Rule

The perceptron learning rule is

> *a procedure for modifying the weights and biases of a network.*
>
> (Perceptron Learning Rule, n.d.)

This learning rule is designed to train a neural network to solve different tasks, by adjusting the weights of each node and generate different outcomes. It is implemented so that the mutation range of the weights is initially fairly high, gradually decreasing as the network reaches an optimum fitness. A general implementation of the perceptron rule is as follows.

```
1 t = 1/n ^ (1/2)
2
3 o = o * exp(t * N(0, 1))
4
5 Function N(mean, dist){
6    return (nextGaussian() * dist) + mean;
7 }
```

Where t is the mutation range and o is the learning rate.

2.9 Item Degradation Learning Algorithm

In order to track different items and how often they are used, one 'unit' needed to be defined. It can't be a simple case of weight or number of occurrences, because people don't use whole loaves of bread at a time and can't weigh each food item before consumption. It seemed reasonable to

suggest that one unit is a measurement of the quantity of produce someone may normally buy. For example if a person normally buys a 2 pint bottle of milk each week, that is one unit. If they then use roughly one eighth of that for their coffee in the morning, then each coffee uses one eighth of a unit. Of course working all of this out can be very time consuming and mean a lot of data, so instead a daily consumption use is calculated. Daily consumption is acquired by calculating the time between shopping trips. Once a trip has been logged, the application assumes a fresh quantity has been registered. This could however get confusing when spontaneous shopping occurs, so the degradation rate begins at one per day for each item. The more information is fed to the application, the more it learns and hence the more accurate it becomes.

It is important to note that the whole system is based on assumptions about food consumption. The degradation rate is a general idea of how long items will last, based on how long they have been in the system for. This is fine for items that get used periodically every week such as bread or milk, but it is more difficult to generalise items that are sparsely used. Items such as tins of red kidney beans, may only be used in chilli which may get cooked once a month. This would mean that it would take a lot longer to gather enough information to generalise its use. It could be argued however that these items are less important and therefore wouldn't matter so much if it needed manual overrides for a while. It could also be argued that some items may only ever be used once or twice, as an experiment for example, so the

usage rate would never properly learn. This would only really be a problem if these items were important, but if they are rarely used, then it wouldn't matter if their usage rates take a long time to adjust correctly. It could also be said that most people only cook from a set number of meals, so these items would be repeatedly used over time. This could be because of time constraints, meaning people don't have the time to get creative with food and may only cook new things at weekends or on rare free days.

As mentioned above, Hebbian learning could be used to generalise this usage rate more, and even start with a more appropriate initial rate. Having weights on the connections between items, size of household and usage rates, a profile could be generated to suggest a family of four would use 2 loaves of bread per week. The initial usage rate could become more accurate through past experiences with other users, and better assumptions can be made using data from a large group of users. Using the perceptron learning rule, these weights could be altered to produce more accurate learning rates, firing items with low stock or popular meal ingredients.

As well as Hebbian learning, regression trees could be used to work out initial usage rates of items. This could take in a number of parameters such as the number of household members and whether the item was consumed whole like fruit or partially like bread. Once all the parameters have been fed into the tree, a usage rate would be given to estimate how long the household would take to consume the item.

The learning algorithm is the main component of the product and as

well as being accurate in predicting item usage rates, in needs to be able to adapt to changing environments. People don't consume food at a constant rate. They don't cook the same meals every week for the rest of their lives. Families grow and this changes what and how much is eaten. This is why a characterisation method is needed and why comparing data with other users helps the application learn new recipes. It also helps determine how large meals need to be for a set number of people. Autonomous systems need to collaborate and learn, and using tools such as hebbian learning and regression trees can accommodate that.

This regression tree (Figure 2.8), is a modification of the previous decision tree and shows how this could be implemented. Categorising how popular items are and how they get used, helps generate a rough estimate as to how often they would be consumed. Finally, these usage rates would be categorised as decaying partially, or as a whole number.

Another example where artificial neural networks have been used to learn parameters are predicting the stock market, learning when to buy and sell shares. These weights are constantly being evolved to adapt to changing share prices and there are no fixed usage rates, because these values are changing all the time.

It could be argued that unlike with stock market predictions, the shopping manager could generalise item usage rates over a period of time, instead of change with every new buy. In a perfect world, where a person uses the same items at a fairly constant rate each week, this would be more accurate. This

method however doesn't adapt to change very well, since it would take a long time to adapt when usage rates change suddenly. Since an average rate would be stored per item, if a person changed their shopping habits, it may take a long time to change that average, if the data set was fairly large.

It may sometimes be required that the usage rate be reset, maybe due to members of the household leaving or arriving for example. This is a feature that would need to be implemented if the usage rate was acquired from a cumulative set of data over time, as this would solve the issue of adjusting to change. Since the usage rate is decided from recent shopping visits, meaning the usage rate effectively get reset each time, this will not be an issue.

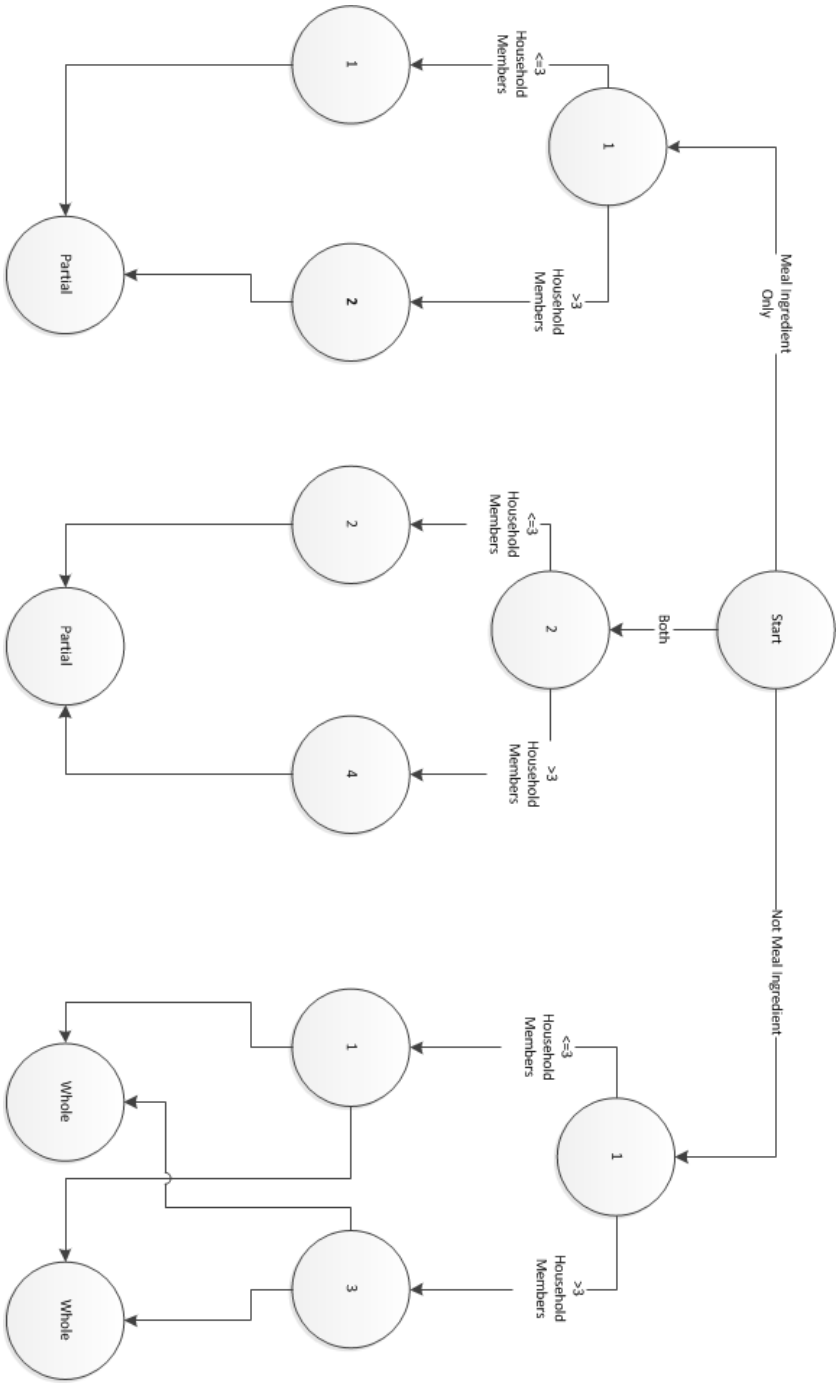


Figure 2.8: A regression tree for a shopping manager.

Chapter 3

Creating A Useful Application

3.1 Key Features

A useful mobile application needs to be appealing to the consumer. It needs to make their lives easier in some way. In order for a product to be of any use, it needs to be simple and easy to understand, otherwise people will lose interest with it. The Unix philosophy describes a program as doing one thing well, stating that each program should only have one purpose within a program. This increases simplicity and makes a program easier to use/maintain. The type of application I am interested in is task management systems, helping to maintain a busy routine. The main features being shopping list management and planning meals.

The idea spawned after realising how organised some people have to be, maybe because they have a family to feed or just need to keep on top of

their busy life. Most people like to plan their weekly shopping, because they don't want to buy food every day, meaning some sort of list needs to be worked out, in order not to run out of shopping items. It would be useful and more accurate to allow some form of artificial intelligence to work this out, providing weekly shopping lists and low stock warnings, making it easier to manage meals. The system will have the following capabilities:

- A pantry to display all items
- Shopping lists containing items to buy
- A meals list showing current and past meals, with information on ingredients and serving suggestions
- A calendar system to log important dates and tasks.

Initially the system will be completely uninformed, imitating a human personal assistant. The more you train it, the more it can generalise, creating warnings of items low on stock and shopping lists containing empty items. Obviously this is only a prototype and in order to actually start off with similar intellect to that of a human being, basic initial knowledge is needed. As mentioned previously, this can be achieved by combining user data, providing some lookup system for deciding on initial item usage rates and prices etc.

3.2 Current Products

3.2.1 Grocery List Manager

This is an online piece of software, useful for storing grocery lists and keeping track of shopping items (Grovery List Manager, n.d.). It links with your google account and allows you to create a list of shopping items. It isn't particularly intelligent because it doesn't learn. It is simply used as a database to store your favourite items. It does however have great reminders such as flagging items that have coupons for money off, which helps the user save money on their weekly shop. It also has the ability to copy recipes straight from other websites and you can even add those ingredients to your grocery list.

Although this is not a phone application, it still has the underlying principles that define a good task management system: ease of use, adaptability and usability. This is important for keeping users interested in the product, since it is hard to get used to complicated software. It also allows for new changes to be made easily, enabling convergence and interoperability.

3.2.2 GeoTasks Task Manager

Geotasks is a clever piece of software that detects your location and alerts you of tasks that could be completed in that area (Geotasks Task Manager, n.d.). For example, if you were near a shopping centre, you may get reminded to buy milk. You can also set an alert range to notify you when you are within a certain distance of the shop, or a few days of needing to get milk. Its simplicity means that the user doesn't need to spend much time learning how to navigate through the application, and it helps keep the whole design looking minimal and basic. This is probably because it is all based around the phone clock and uses applications already available, such as google maps and phone widgets. A key feature of any phone application is the fact that it is customisable to the user's needs. Geotasks let you customise the background, alert time and range, and latitude/longitude.

3.2.3 Mister Lister

This is an application designed for the iPhone, iPod touch and iPad. It is very similar to the grocery list manager but it is developed by apple for their portable range. It can run on your computer and synced between devices, which eliminates the need to transfer multiple copies (Mister Lister Grocery List Manager, n.d.).

Some negative reviews of this product mention wanting to order shopping lists by alphabetical order and attach simple push alerts. Having push

alerts would remind users of stocks running out and storing list alphabetically would help to keep some order when searching. Another main problem with this application is the fact that items are priced automatically, which is a pain for people who live in a state with no tax on food. If the items could be edited and personalised, this could be avoided.

3.3 Gap In The Market

There are many features of these products and others that are really important for keeping track of tasks and shopping items. The main noticeable features are usability, simplicity and flexibility. If a product is to succeed, it needs to be easy to understand, without too many options to work through. It needs to be customisable, so that the user can mould it around their daily routine. If possible, extra features would include synchronicity between other applications such as google calendar across the internet.

Having one application that can mould many services together is useful and it proves that an autonomous system could be popular if it could learn these things for itself. Most of these applications have been designed through ratings and comments on the android marketplace. They are then released with extra patches satisfying these comments, which can be a slow process. If an application could learn these features through crowd sourcing and personal suggestions, it would decrease the time and effort spent releasing new versions.

It can still be hard work managing software that is meant to be handling tasks for you. It doesn't seem to be any different to simple pen and paper. If the software could think for itself, making its own decisions, the user would benefit hugely. Initially there would still need to be some sort of maintenance and the user would need to monitor the software's decisions, but at least the responsibility could be shared, and eventually handed over completely. Allowing this service to be used with multiple interfaces such as a website, or Java applet similar to Astrid (Astrid Android Task Manager, n.d.), also brings more freedom to the user.

Chapter 4

The Product

4.1 Aims

- To develop and evaluate an automation shopping service for predicting a person's daily routine
- To discover the effects of applying an autonomous computer system to a person's daily schedule and the limitations of such a task

4.2 Objectives

- Complete literature research of current task management systems and mobile applications
- Develop a suitable way of storing and linking personal data for use with a lookup system

- Complete a basic application for storing food items, building shopping lists, creating recipes, and organising tasks and chores.
- Link prices of items to the internet to improve accuracy and allow the product to rely less on user input.
- Evaluate ways of providing autonomous learning to the product, researching past and present theories
- Apply chosen method to enable the product to predict when items run out and tasks need doing
- Extend the product to supply alternative meals and shopping lists.

4.2.1 Product Design

The design of the product is based on the Facade pattern, which separates each individual component, allowing all communication to go through one class. The makes everything more readable and reduces coupling, which often implies high cohesion.

Since Android likes to include the user interface within the logic, using a Facade pattern was fairly difficult. The UML diagram in Figure 4.1 shows the "MainMenu" class effectively being the Facade, controlling which operation is implemented. All the classes that don't require a dynamic layout, reference an XML file, which contains the view of that page. These XML files however don't control any operations on buttons etc. so the user interface is effectively split into two, which is why all the processing has to be performed within the "MainMenu" class.

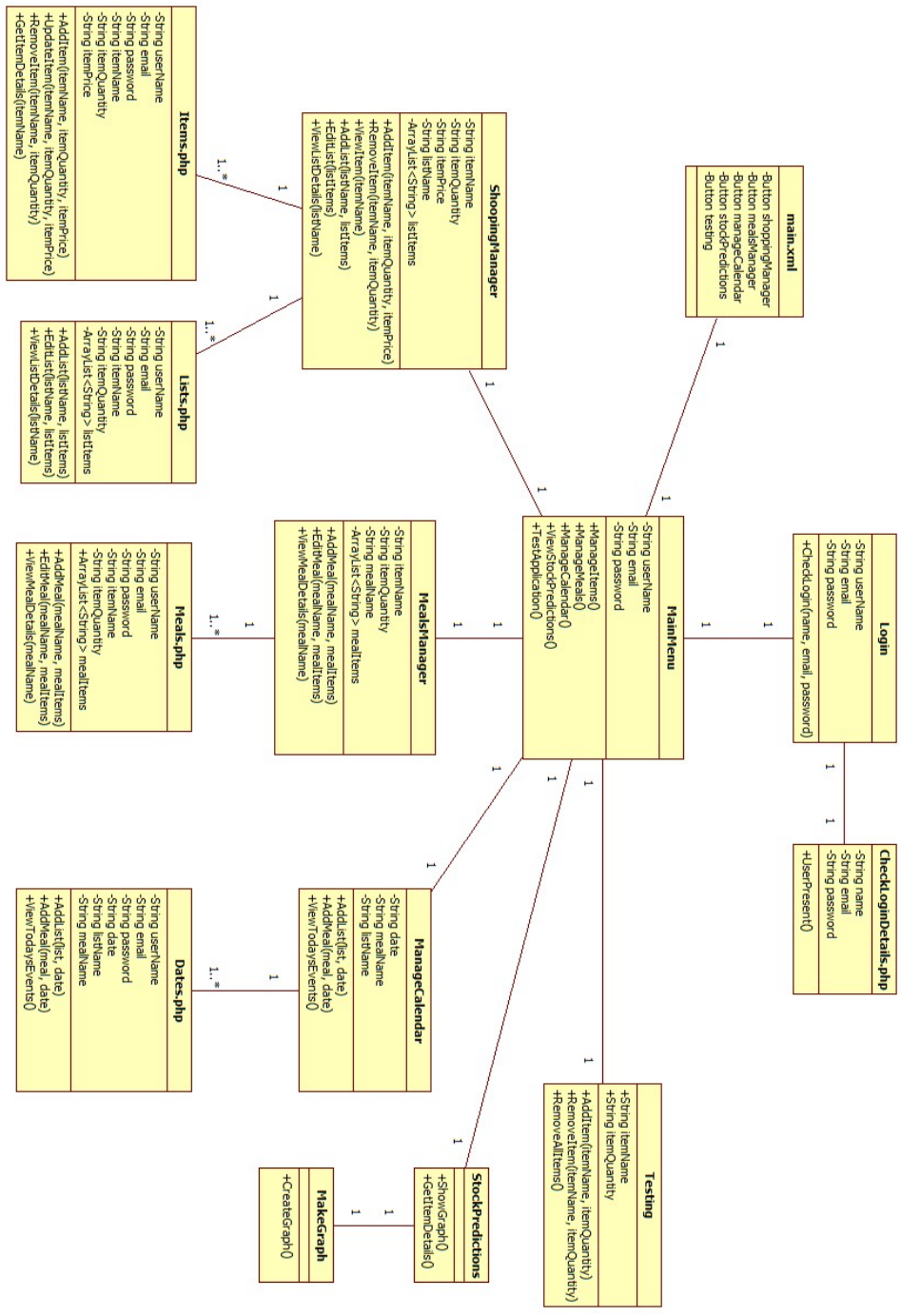


Figure 4.1: Product UML Diagram

4.2.2 Requirements Analysis

The aim of this report is to highlight the possibility of combining services to create autonomous learning. Although this application will not be self-healing, it will be able to learn basic user choices on its own, using an inference-ruled approach.

The items manager will allow the user to add, edit and remove items from the database. It will soon learn how often items run out based on how often they are bought. It is this information that supports the rest of the application. The shopping list manager will be used for creating and editing shopping lists, for future visits to the supermarket. These lists can be planned on a certain date, which will cause an alarm to be set. On the day, the contents of the list can be viewed and each item can be confirmed as bought, updating the database. The same functionality applies to storing recipes, as well as an option for displaying available meals based on available stock. On the day the meal needs to be cooked, the application will remind the user to buy the required ingredients.

Based on the MoSCoW prioritisation technique, used for ordering requirements, the product requirements can be found in Table A.3.

This application will be able to predict when items run out, how often they are used, if meal ingredients are present, and other links that could be of importance. The idea is to make the system learn with a more human attitude, generating ideas behind the data and spotting important information based on past experiences. Diagram A.9 explains each use-case further,

Requirement	MUST	SHOULD	COULD	WONT
Adding Items	x			
Editing Items		x		
Adding Lists	x			
Editing Lists		x		
Adding Meals	x			
Editing Meals		x		
Planning Lists			x	
Planning Meals			x	
Viewing Dates			x	
Providing warnings for upcoming dates			x	
Self-Generated Lists			x	
Link With Website			x	
Warnings about upcoming dates and empty stock			x	
Provide Alternatives				x
Link With Other Users				x

Figure 4.2: MoSCoW Requirments Table

providing a description of the product capabilities, whereas Figure 4.2 displays a complete overview.

Chapter 5

Product Development

5.1 Test Driven Development

Test-driven development (Beck, 2002) is a way of keeping control of production, by imposing tests on each new component. It ensures that each task is finished and working before starting the next, and protects tests from being forgotten, if left until the end.

The idea is to write test code before functional code, so that it is clear what to expect. This prevents small changes being made throughout the implementation.

There are two levels of test-driven development: Acceptance and Developer (Introduction To Test Driven Development (TDD), n.d.). Acceptance testing includes writing an acceptance test, or behavioural specification, along with enough code to satisfy that requirement. Developer testing

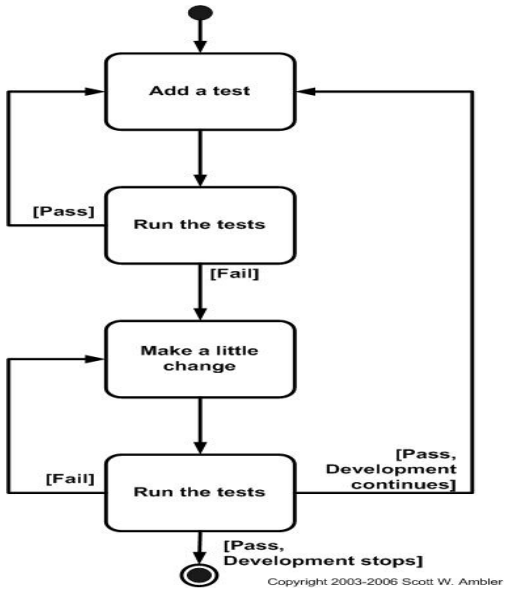


Figure 5.1: Test-Driven Development

includes writing a developer test, or unit test, along with enough code to fulfil that test requirement. This is the most common form of test-driven development. Unit tests will be applied to the learning algorithm, as one single unit, and functional tests will be introduced to test the quality of the components within the application.

5.2 Development Tools

There are many mobile applications on the market, the main competitors being Android, iPhone, and Blackberry, each needing different development environments. The most popular IDE for Android is eclipse, mainly because of the support available and the fact that it is recommended on their website, with detail instructions on how to set everything up. Like Android, Blackberry applications are also written in Java, the most common IDE being Netbeans with the mobility pack. iPhone suggest a different development environment called Xcode and require all applications to be written in objective C.

5.3 PhoneGap

PhoneGap (Ivings, 2011), is a way of creating mobile applications using web-based technologies, rather than complex, native languages such as Java or Objective-C. According to *TMCnet*, the mobile application downloads market is expected to reach $58 billion by 2014 (Campbell, 2011). With more and more applications being developed, and different companies bringing out new platforms with their own architecture, cross-platform compatibility becomes a big issue. Using PhoneGap allows one application to be produced for multiple platforms.

With modern HTML and JavaScript engines we can create a web-

> site that is virtually indistinguishable from a native application. (PhoneGap, 2012), Slide 4

Since all modern phones have web browsers, it is possible to write one application for all platforms using web-based languages. These languages are much simple than the native languages, meaning development will be quicker. Other Internet tools such as jQuery even allow such applications to be developed online.

> These combine HTML, CSS and Javascript, allowing us to design a webpage tailored specifically for mobiles. (PhoneGap, 2012), Slide 5

Below is some example code using jQuery.

```
<!-- Login Page -->
<div data-role="page" data-theme="a" id="login" >

   <div data-role="header" data-backbtn="false">
      <h1>Inbound Mobile</h1>
   </div>
   <div data-role="content">

      <form id="login_form">
         <fieldset data-role="fieldcontain" id="login_content">
            <label for="username">Username: </label>
            <input type="text" id="username" />

            <label for="password">Password: </label>
            <input type="password" id="password" />

            <label style="display:inline-block;" for="remember">
               Remember</label>
            <input type="checkbox" data-inline="true" name="
               remember" id="remember" />
```

5.3.1 Disadvantages

Although this seems like a much easier option, there are a few complications. JQuery can only be responsible for the design of mobile applications and cannot access the native device's API, such as the camera or network. It also means that although these applications can reach a larger audience, they cannot be distributed in the normal way, such as the Android and Apple markets. There is however a solution, all mobile operating systems use a service called WebView. This allows web pages to be rendered within native applications. Along with *rhomobile* and *titanium*, PhoneGap can be used to utilise this service.

5.3.2 Signing Applications

Once an application has been developed, they need to be signed in order for them to run on a device, or sold on the market. Apple require a developer's account (costing £60) and a Mac, and involve a fairly long-winded process. Android only require an Android SDK and a plugin for eclipse, if that is your chosen IDE. Applications can be self-signed, for debugging and a key can be generated using *keytool* (for Windows) or an Eclipse plugin, when they are ready to publicise. An Android developer account is needed for this, costing around £15. Blackberry and Nokia use slightly different methods, but the concept is still the same. Applications need to be signed in a certain way to meet different company requirements.

PhoneGap makes this whole process a lot easier, allowing all applications to be signed off using the same method, regardless of the product manufacturer.

> PhoneGap Build allows us to compile our Application in the cloud, creating an App for each Device from one source folder. (PhoneGap, 2012), Slide 30

5.4 Why Android?

Android is a relatively new operating system designed for smartphones and more recently tablets. It was originally developed by Android Inc. until Google purchased them in 2005. Since then Android has grown and the Android marketplace has introduced young developers from all over the world. It is the basis of phones such as the Galaxy S, Nexus S, and the Desire S, due to its ease of use and development. Google provide the basic framework behind Android and anyone can develop this further in their own unique way. There is so much freedom when developing in Android, since nearly everything can be integrated with other services, such as the built in camera or messaging service.

Other areas for concern are programming language and helpful sources available. Java is a widely used programming language and is a good starting point for mobile development, since it can be used to create applications for Blackberry mobile phones as well as Android. There are a lot of helpful

sources on the internet and it is relatively easy to progress towards more complicated Java applications. The development is performed over the top of the Android API, which is used to interact with the underlying Android system.

5.5 Development Environment - Eclipse

There are many IDE's available for Android development such as Netbeans and intelliJ. The most popular of the two being Eclipse since there is a lot of help available on the android-developers website regarding set-up/use. There are also more plug-ins available as Eclipse is used by Android developers for creating their own software.

5.6 Android Architecture

> *Android is a software stack for mobile devices that includes an operating system, middleware and key applications (Android-Developers, 2012)*

The Android architecture consists of four main layers. These are the applications layer, application framework, libraries and the Linux kernel. The applications layer consists of the general features that can be seen by the user, such as contacts, Web browsers and other applications. This is the top level of any Android application. This layer sits on the application

framework, which contains all the services used by the application, such as window managers and notification managers. It is used every time an application calls a service, for example viewing their contacts or setting an alarm. The next layer down contains all the libraries that the application might use. This could be anything from SQLite to a simple surface manager. These layers sit on the main Linux kernel that contains all the drivers needed for using the phone's built in hardware. This could be for using the camera, WiFi or Audio drivers. Please see Diagram A.4 for more detail.

5.7 Android Integration With Java

Due to the layering approach between the android architecture and Java layer, the programming style changes slightly. Instead of one main Java class, Android has many runnable Activities, which dictate a part of the program. However, other Java classes can still be used in the same way. Activities are simply an extension of a Java class:

```
1 public class SomeClass extends Activity {
2 }
```

All Activities need to be declared in the Android manifest file, typically in the form of XML. This is used to locate the correct classes and declare the main Activity which the application starts with. The manifest file also contains the android version number, as well as all the permissions needed

for starting certain services, such as using the internet or the vibrate functionality of the phone.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest>
3      <uses-sdk android:minSdkVersion="9" />
4      <uses-permission "permission"></uses-permission>
5      <application>
6        <activity android:name=".name"
7                  android:label="@string/app_name">
8              <intent-filter>
9                  <action android:name="name.MAIN" />
10                 <category android:name="name.LAUNCHER" />
11             </intent-filter>
12         </activity>
13         <activity android:name=".MainMenu"
14                 android:label="@string/app_name">
15             <intent-filter>
16                 <action android:name="name.MainMenu" />
17                 <category android:name="name.DEFAULT" />
18             </intent-filter>
19         </activity>
20     </application>
21 <manifest>
```

All Activities need a way of displaying their content, using a textfield, buttons or forms etc. Their layout can either be produced dynamically, creating the relevant components within the code, or created with an XML file. These files are similar to Web style sheets, in the sense that they define the page (or Activity's) layout. The XML files are then referenced inside each Activity.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
>
    <EditText
      android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:inputType="textPersonName"
      android:id="@+id/personName"
      android:text="Name"
    ></EditText>
</LinearLayout>
```

Android links these Activities together using Intents. These usually have the form

```
<currentActivity, destinationActivity>
```

which specifies the connecting Activity and next destination should the event happen. These Intents can then be called when needed, for example if a button is pressed or form submitted.

```
1  // Create an intent for going back a page
2  final Intent back = new Intent();
3  back.setClass(AddItemToDatabase.this, ShoppingList.class);
4
5  // Set a listener for the back button
6  backButton.setOnClickListener(new View.OnClickListener() {
7    @Override
8    public void onClick(View v) {
9      // Go the previous page
10     startActivity(back);
11   }
12 });
```

5.8 Development Technique

In order to get a working model early, so that any problems can be detected with time to spare, a rapid prototyping approach will be used. Identifying problems early means that a more achievable time plan can be formed, allowing for adjustments to be made to the application. An *agile* production cycle will also be used with each prototype, so that each feature can be tested before moving on to more complex tasks. It is important to get a stable prototype before any advanced features are added. A time plan can be viewed in Table A.1, as well as diagram A.13, showing this in more detail. A monthly plan in section A.14 also explains this further.

5.9 Development Ideas

Throughout the development process and after receiving user feedback some features have appeared technically difficult. It was important to maintain an environment that is easy to use and make sure that all the relevant data was stored in the correct way, otherwise it would be more difficult for the application to maintain links between stock and provide clear information.

5.9.1 Storing User Information

In order to manage large amounts of data securely, with ease of access from multiple sources it was necessary to have a database that could satisfy the following requirements:

- Data needs to be stored externally (for back-up and restore, and the ability to access data through multiple devices)
- Data can be represented in a more manageable way, such as XML or JSON

The simplest and most versatile way of storing user data seemed to be using a database such as MySQL (The Advantages and Disadvantages of MySQL, n.d.) because it is open source, free and robust. It is important to have a system that is reliable because there is a lot of sensitive information being generated that is personal to the users. There is also a lot of help available for performing queries or simply setting everything up. It is an

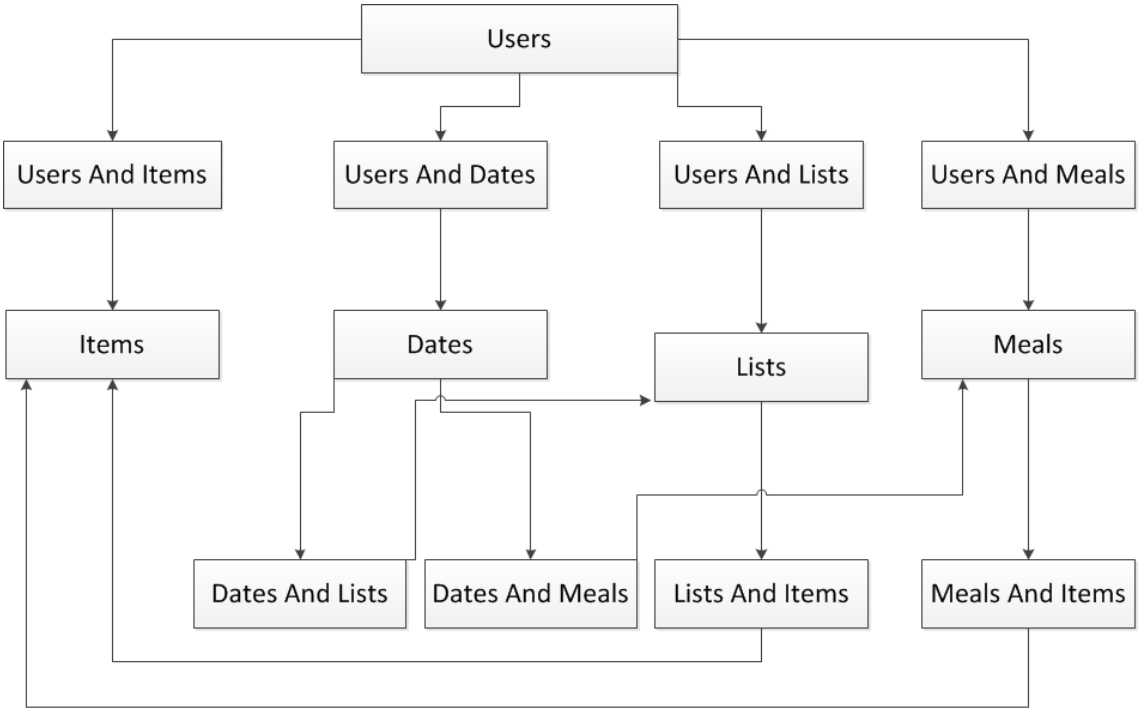


Figure 5.2: The Product Database Design

easy-to-use, multi-platform system that provides the essential features of a more expensive database such as Oracle (Oracle - Databases, n.d.).

In order to ensure that further developments can be made, there will be a number of specific tables, connected using primary keys.

Item names will be stored in the "Items" table, and they will be referenced in tables "Users And Items", "Lists And Items", and "Meals And Items". This is so that different users can reference the same item in the "Items" table, but other information such as item quantity and price will be stored in the "Users And Items" table. This is to allow for differing item details.

Users table

ID	Username	Password	Email Address
Alex	1	alex123	alex@yahoo.com

Items table

ItemID	ItemName
1	Onion

Users And Items table

ItemID	price	quantity	UserID
1	1.00	3	1

Users And Lists table

UserID	ListID
1	1

Lists And Items table

ListID	ItemID	ItemQuantity
1	1	2

Figure 5.3: The Product Database Structure

The ingredient quantities will also be stored in the "Meals And Items" table and the same applies for the "Lists And Items" table. This prevents duplicated entries into the database and means data can be shared easily, as well as being more secure. User information is kept separate but item details can be shared anonymously. This approach is similar with the "Dates" table. If new users are added, a new entry will be added to the "Users" table and that primary key will be used to reference the items, lists, meals and dates. For a more detailed representation, see the tables in Figure 7.1 and Appendix A.2.

5.9.2 Representing User Information

Storing data externally is only useful to an application if it can be represented in a meaningful way. Two implementations that are useful within a Java program as well as a Web interface are JSON and XML.

XML was designed as a markup language for storing large amounts of data, such as documents, in a human-readable format. The most common XML platform for this is DocBook, which is an XML specification for documentation. XML is robust and easy to understand, which is why it can handle such large amounts of information. XML can also work well with databases since data can be displayed in a machine readable-format. Some common uses of XML are for RSS feeds in Web development and database management systems. These could be used for storing descriptive information for websites needing an image gallery, somewhere to store product details, or services such as weather systems and banking services.

JSON is a lightweight version of XML derived from JavaScript. It is less verbose than XML, which makes it more human-readable, but JSON can be more limiting, due to JavaScript's reserved words. JSON is faster than XML because of its simplicity and size, but XML can handle large documents easier. If this application would benefit from the speeds that JSON offers, there would be no contest, but since the sluggish nature of Java would contradict this and the fact that XML can manage large amounts of data, as well as provide extensive error checking, XML seems the more appealing choice. XML has also been around for longer, meaning there are

more resources and better support available.

> *Virtually anyone who has done programming for the enterprise, web, or mobile markets in recent years has encountered XML. It is just about everywhere you look. (Ableson, 2010)*

5.9.3 Dynamic Item Check lists

If a shopping lists or meal needs editing or their contents displayed, the user can view a check list of the items concerned. This is dynamically generated depending on how many items need displaying and allows them to select a subset of the contents available. It is necessary for when a shopping list is re-used, but only certain items actually bought, or specific meal ingredients used. Without the option of selecting only a few items, the entire list would be altered instead of only those items that were changed. This is a more advanced way of generating a layout because the Activity doesn't reference a fixed XML file. Each layer of the screen layout needs to be added manually and finalised at the end of the code.

5.9.4 Setting Alarms For Calendar Events

In order for reminders to be set for various shopping trips and meals, Android's Alarm Manager was used. One alarm would be scheduled for the morning of that day detailing the upcoming events, and another would be set for the evening asking for confirmation of items used. To notify the user of these events, a notification will be created in the notifications bar directing them to the shopping lists or meals of the day.

5.9.5 Submitting Shopping Lists and Meals

When a meal is cooked, or a shopping list has been created, it would be helpful if the stock levels were altered to accommodate this, rather than altering them all manually. This is why a submit button will be applied, which alters stock levels accordingly when clicked. Stock levels will increase when a shopping list is submitted and decreased when a meal is submitted, saving the user a lot of time and effort.

5.9.6 Viewing Available Meals And Booking Events

Sometimes meals are not able to be cooked, because certain ingredients are not available. To facilitate this, the application will have a page to view available meals, instead of the user having to look at each item manually. Items are presented using a dynamic check-list mentioned in section 5.9.3, in-case not all of them are bought/used.

There will also be reminders when shopping visits and meals are booked using the dates page, which warns the user to get the required ingredients if they are out of stock. This allows them to be more organised and prevents last minute shopping visits. A warning will appear in the morning and evening, so they can submit the event if they forgot to do so, using the submit button on the notification page.

5.9.7 Creating Shopping Lists Automatically

Since the application will be able to notice empty stock, it would make sense to allow shopping lists to be created automatically based on stock levels. The user will have to option to create lists manually or click a button to build one automatically, allowing items to be added and removed after.

5.9.8 Synchronising Local And Global Databases

If the website is used to edit stock items instead of the user's phone, the local and global database will be slightly different. This is why the date each database is changed will be stored, allowing a comparison to be made when the phone application is used. If the global edit date is more recent than the local date, its contents will be duplicated locally, and vice versa. For efficiency, if the mobile phone has access to the global and local database, both will be changed, to save the need to synchronise the two.

5.9.9 Security

So that personal details remain hidden and only permissive data is shared, each user must register. This can be done using the website or the mobile application and stores a user-name, password and email address. Structuring the databases so that users are separate from items, allows personal details to remain private, and information about stock to be shared

Chapter 6

Prototype 1

6.1 Product purpose

The first prototype had minimal learning capabilities. Its main purpose was for getting valuable feedback about usability. The product allowed the user to manage shopping items, as well as create shopping lists. It could also manage meals and notify the user of important dates.

In order for the system to learn efficiently, all the information needed to be stored correctly, as well as important calculations made, such as working out the prices of meals and lists based on the prices of the items. It was a good idea to get some feedback from the alarm system, such as whether it was useful having two alarms being set on the day of the event, so that a confirmation form could be submitted and if the alarm times were appropriate.

Once the required information was gathered and feedback acquired, it was easier to design the learning functionality and incorporate it into the main application. There would be no need for added functionality if the basic uses were not understood.

6.2 Requirements

Since usability was important for the first prototype, so that the basic features could be tested appropriately, it was important to take a minimalistic approach. It would be a huge set back if the results learnt from user data were impaired by incorrect use. The main features are detailed below.

- Adding and removing a shopping item, with details of its price and quantity.
- Adding and editing shopping lists, changing items and their quantities.
- Setting reminders for shopping trips and meals, where the user gets notified to update items used or simply submit the entire list/recipe.

6.3 Successful Features

The most popular features from the first prototype were allowing items to be individually selected from shopping lists and meals, and the fact that new items could be added when creating such lists and meals, instead of adding them beforehand.

When the user decides to view a shopping list or meal, they have the chance to select only the items used, instead of submitting the entire list. This gives flexibility for changing habits every now and again. There is also this option when editing meals. The user can select more than one item to be removed from the list, so it is faster to change the details.

Some feedback gained from this prototype highlighted the ability to view upcoming meals and shopping trips and cancel any that were unwanted. This would allow the user to plan ahead, which prevents any surprising contradictions to their routine. Extra features such as this are less important at this stage, but will be thought about once basic learning capabilities are achieved.

6.4 Required Improvements

Some general requirements needed were mainly additions to the basic design, allowing for more freedom with meals and shopping lists. It is necessary to store all data submitted to the application in order to generate accurate predictions. However this could mean there are a lot of items that are no longer bought still being displayed. To resolve this, a filter could be used and a flag system to stop warnings being generated about out of stock, unused items. New ideas received from the trial of this prototype are:

- Finding meals available based on the current stock.
- Updating the user via email when events occur.
- Provide warnings if items are missing for a meal being planned.

Chapter 7

Prototype 2

7.1 Product purpose

The second prototype included basic learning functionality, mainly adapting item degradation levels. It was a test of the applications implemented learning algorithms, trying to discover any flaws with the design. As discussed in section 2.9 on page 37, the application assigns daily degradation rates of stock items. These are calculated using the formula:

Degradation Rate = $(Q \ / \ (D_{bought} - D_{previous})$

Where Q stands for item quantity and D stands for date.

In other words, the degradation rate is how much of an item is used over a period of time. Once an item's quantity reaches zero, the user will get a prompt asking for confirmation. If this isn't the case, they can update the quantity and the item's degradation rate is worked out as before. To avoid

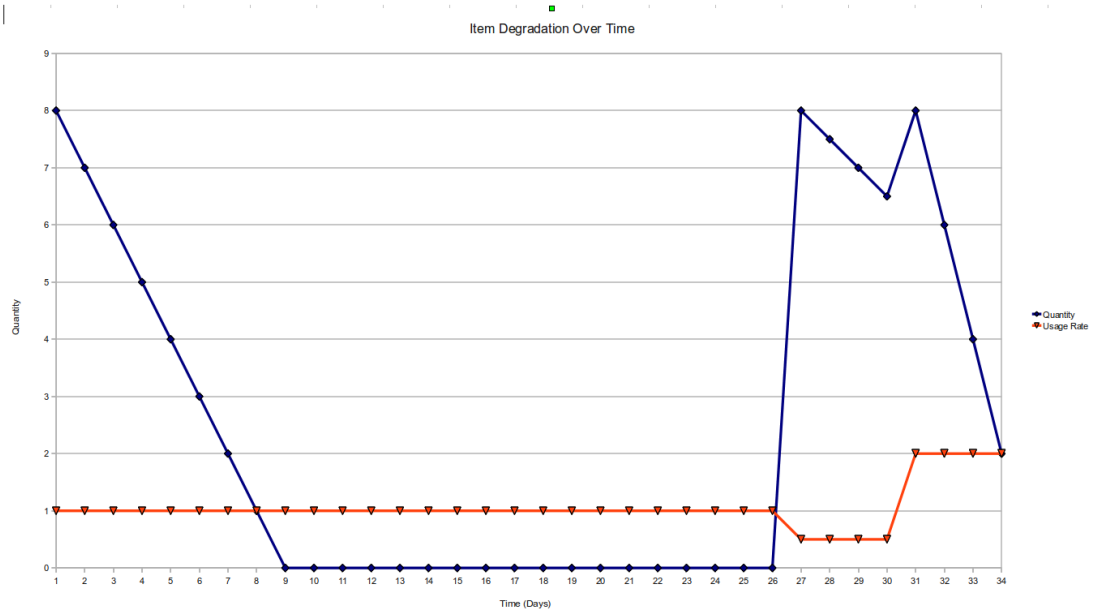


Figure 7.1: A Graph Showing The Degradation Rate Adapt To User Habit

an overload of prompts, each day one prompt will display all stock that the application believes are empty and using a check list, the user can edit each item accordingly.

The graph below shows how the usage rate alters after each item quantity is changed. The blue line is the item quantity degrading and the red line is the usage rate being adjusted every time that item is bought. The spikes in the blue line indicate an item being bought and its stock replenished.

A more sophisticated approach would be to predict these peaks and build a better picture of how much is bought each week. Section 9.0.1 shows a more advanced learning rule, taking into account predicted item quantities.

7.2 Requirements

This prototype included all the main features of the application. The main differences were:

- Finding meals available based on the current stock.
- Creating shopping Lists based on empty stock.
- Providing a four day forecast predicting stock levels.

This was the penultimate prototype, before the testing scenarios were implemented. This was the last version to get user feedback on, however any feedback on the final product could be included in future releases, if this were to be sold.

7.3 Product Feedback

The feedback gained was mainly positive, commenting on the ease of use and informative pages, as well as accurate stock level predictions. The drop down list of existing items was helpful for adding shopping lists and meals, but they still need adding sequentially, as opposed to a check-list structure such as the one used for submitting shopping items bought.

Other comments were about the presentation of item details. The information page is not very appealing and could be displayed in a more friendly way. The application's capabilities seem to be accepted by the user, but the whole product could benefit from having a slight re-design. This is not of much importance, compared to its purpose, but if this product were to be released officially, this could be a concern.

Chapter 8

Testing

There are many ways to test software, be it through unit tests or simply releasing it to a focus group for feedback. According to *Jiantao Pan* at Carnegie Mellon University, software testing is:

> *the process of executing a program or system with the intent of finding errors* (CMU Testing, n.d.)

Bearing this in mind, it would be wise to test all aspects of the design functionality, using multiple methods, in the hope of getting different views on the application. The main methods used were:

- Unit Testing
- Functional Testing
- Releasing the application to a focus group

8.1 Unit Testing

Unit testing is a way of monitoring individual parts (or units) of your software for errors. It is often done automatically but can also be performed manually. Testing individual units of code helps discover where errors occur and ensures that a wider range of test cases are tried. This is because a test can be placed on each method of code, which can then run sequentially with the others, to check if that method is working correctly. If this wasn't implemented, certain methods may rarely be used, allowing errors to be hidden.

According to *Roy Osherove* (Osherove, 2009), a unit test must be:

- Trustworthy - A test needs to be reliable to ensure trusted results
- Maintainable - To ensure they last
- Readable - For others to maintain easily

8.1.1 Unit Testing Tools

There are many tools available for performing unit tests on software. Depending on the development environment, these are sometimes integrated within the program.

In Eclipse, within the Android package, there is the option to create an Android Test Project (Android Unit Testing, 2012), which allows you to monitor a specific class and utilise its components. A basic test class utilises the *ActivityInstrumentation* toolkit and the following code accesses the class *AddItemToDatabase*.

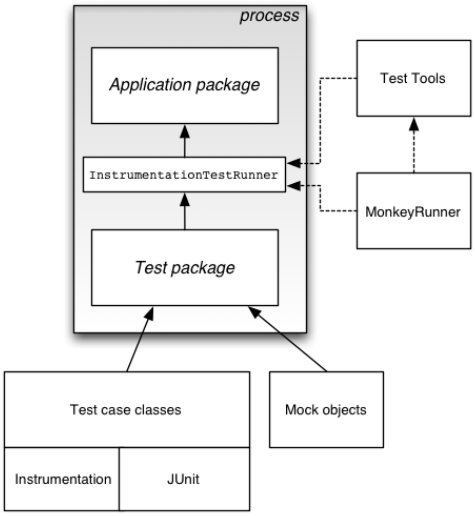


Figure 8.1: Android Testing Framework

```
1  package com.jonathan.addItem.test;
2
3  import java.text.DecimalFormat;
4
5  import android.test.ActivityInstrumentationTestCase2;
6  import android.widget.TextView;
7
8  import com.jonathan.addItem.*;
9
10 public class TestAddItem extends
       ActivityInstrumentationTestCase2<AddItemToDatabase> {
11     public TestAddItem() {
12         super("com.jonathan.addItem", AddItemToDatabase.class);
13     }
14     @Override
15     protected void setUp() throws Exception {
16         super.setUp();
17         mActivity = this.getActivity();
18         itemName = (TextView) mActivity.findViewById(com.jonathan.
               addItem.R.id.addItemName);
19         itemPrice = (TextView) mActivity.findViewById(com.jonathan
               .addItem.R.id.addItemPrice);
20         itemQuantity = (TextView) mActivity.findViewById(com.
               jonathan.addItem.R.id.addItemQuantity);
21      }
22
23     public void testPreconditions() {
24         // Test a section of the class here
25     }
26 }
```

Other unit testing tools include *JUnit Testing* (JUnit Unit Testing, 2012), which is very similar to Android testing, *FoneMonkey* (FoneMonkey, n.d.) and *Robotium* (Robotium, n.d.). FoneMonkey allows users to record scenarios and test their functionality. Once a scenario is recorded, it can be passed through a script to validate the input. This software is installed onto the phone and is used to interact with its applications. It is good for keep-

ing a record of user activity during the testing phase and testing individual components.

Robotium is essentially an Android version of *Selenium* (Selenium, n.d.), which is for automating web browsers. It acts as a symbolic user, clicking buttons and populating forms. In Robotium, scripts can be written to send test data to an application, automating user interaction and decreasing usage time.

8.1.2 Applying Unit Tests Within The Application

To test whether the application can alter the usage rates of items, a separate page was added, allowing total control over what data is added to the database. This meant dates could be specified for additional items, allowing the decay rate to be trained quickly, imitating future dates.

Another test for evaluating the effectiveness of the learning algorithm was carried out by sending multiple XML files containing item details to the database. Each file contained a different date and simulated adding items over a period of time. Graph A.5 shows this simulation adding a number of apples on three different dates. The blue jagged line represents the degrading quantity, with the spikes showing more apples being added. The lower orange line shows how the usage rate changes each time more apples are added.

It is virtually impossible to write completely error-free code, due to human error and most programs managing many tasks. A software developer can design and write a piece of code, testing individual parts along the way, but

do they know it performs every requirement once it is finished? It is hard not to be suggestive when testing your own code, as one person cannot think of every possible situation. This is where unit testing can be used to test all major pieces of code, allowing smaller errors to be found with *Functional Testing*.

8.2 Functional Testing

Functional Testing is implemented to test individual requirements of a piece of code. This could be ensuring pages link together properly, text is displayed correctly or information is added to a database. One function could be multiple units of code. Functional testing is less specific than unit testing, giving an overview of the capabilities of a piece of software, without getting caught up in small errors.

To test the usability of the application, a number of test cases were exercised and presented to a focus group. Table A.6 shows the different test cases and how the user found them. Robotium was then used to test these scenarios, creating a step-by-step execution, including all the components of the application. Some sample code below shows a test to add items to the database.

```
1 // Add item to database
2 solo.clickOnButton("Shopping List");
3 solo.clickOnButton("Add/Edit Item");
4 solo.enterText(0, "BREAD");
5 solo.enterText(1, "1.20");
6 solo.enterText(2, "1");
7 solo.clickOnButton("Add/Edit");
```

In order for Robotium to be used, the application had to be installed onto the device in debug mode, for testing. An APK file can still be signed in debug mode within Eclipse for easy release to mobile devices.

8.3 Gathering User Feedback

As well as functional and unit tests, a survey was carried out to get a better idea about what was expected of an autonomous shopping manager and how it would perform. Since the application depends on user data to become effective, the usability of the application is critical if people are expected to populate the database accurately, or even stay interested at all. This is the most effective way of determining if the new data predicted by the application is accurate and determining if there will simply be too much to learn.

The questionnaire comprised of ten questions about usability and what features are important, and five questions about user responses to the product. These last questions were only aimed at those that had tried the product a sufficient amount. The product was given to a small group to experiment with other a period of a few days. Since the intention of this questionnaire was not to test the learning algorithm, but mainly how the product used that information, it was not necessary to let them use it for any longer. After the group had become familiar with the application and utilised all the options, they were more able to provide insights into how it could be better. These comments were included at the end of the questionnaire. Highlights of this questionnaire can be found in table A.8.

Chapter 9

Evaluation

The aim of this application was to decide whether it was possible to create a system that could learn for itself and generate conclusions without the need for user interaction. This was tested by applying unit tests on its learning ability, using functional testing to test the usability of the application and consulting a focus group to determine the effectiveness of its functionality. The results from the unit and functional tests were used to analyse the possibility of a completely autonomous system, by analysing the limits of a shopping manager which inhibits some autonomous learning. The results from the questionnaire were used to decide whether this type of application would be useful to others and if it were made to be completely autonomous, could it handle all the usage data, in order to make accurate assumptions.

The results from unit tests show that it is possible to create autonomous learning after each new entry to the database. However, in order to become

fully autonomous, there needs to be some sort of memory and lookup system, analysing the error and taking past decisions into account. An example of this could be to use a multi-layer perceptron.

9.0.1 Alternative Learning Algorithm - Smoothing

It would be more effective to have some way of basing future decisions on past experiences, generating more accurate predictions. Just like the original learning algorithm, this would work out predicted next purchases based on the last date bought and the usage rate, but also take into account meals cooked. This would be more effective than just basing assumptions on shopping visits, not recording how often meal ingredients are used. This way a predicted purchase level could be generated, based on the target level and predicted level.

```
1 *Assume target stock level = level after last purchase*
2
3 predictedPurchase = targetLevel - predictedLevel
4 predictedLevel = timeSinceLastShop * usageRate - mealsCooked
```

Using this algorithm, an error can be generated for when the application needs to be corrected. This is simply the difference in stock levels. The new usage rate can then be formed taking into account the previous rate, the error and some learning rate.

```
1 Error = actualBuy - predictedBuy
2 newDecay = oldDecay + (error * LR)
```

This method takes into account previous data and also learns around the user's meal consumption, achknowledging items being used in recipes and preventing quantities being degraded more than once each day. Initially item quantities will degrade based on shopping visits alone, which takes into account meals cooked each week, however it doesn't plan for any changes in routine, cooking something completely different. This will cause the user to submit the meal they had one week. Say they did this 3 times, causing the predicted target and the error to decrease. The predicted buy of the ingredients would then decrease by 3, causing the error to also decrease by 3. The new decay rate would then decrease by 3 times the learning rate, since it is assumed that this item can be included in meals and part of its quantity decay rate is taken care of manually by the user.

```
1 predictedBuy = targetLevel - predictedLevel
2 predictedLevel = timeSinceLastShop * usageRate - 3
3
4 Error = actualBuy - predictedBuy
5 newDecay = oldDecay + (-3 * LR)
```

The new usage rate decreases, allowing the user to submit meals and not affect item stock levels. If this changes, and stock levels need to degrade because meals were not submitted, the decay rate will be too high, causing the predicted stock level to decrease, the predicted buy to increase and the error to increase.

```
predictedBuy = targetLevel - predictedLevel
predictedLevel = timeSinceLastShop * 3 - 0

Error = actualBuy - predictedBuy
newDecay = oldDecay + (3 * LR)
```

9.0.2 Unit Test Results

The unit tests to analyse the limits of the learning algorithm, were very successful in proving it is possible for it to be implemented within a shopping manager application. Manually adding items to the database using the test page inside the application was a useful way of assuring the local database was storing the data properly, as well as altering the usage rates accordingly. Adjusting the usage rate this way however doesn't account for any previous decisions and only the most recent item details are monitored. This isn't a very sophisticated method since the user still has to apply a lot of thought to make sure the application hasn't missed anything out. Say for example, if one of the children in a household goes to University. When they are absent, the usage rate has to be trained to adapt to a smaller household. When

the child returns home, the application has no memory of consumption rates before they left. This means that the application has to be trained with every change and only works effectively with a consistent user.

The new learning method mentioned above is much better at handling changes in meals cooked and the frequency of items bought, but even this has problems. Suppose the meal needs to be cooked for more people. The recipe will need to be edited so that ingredient quantities can be changed. Using some method of sharing user data, this could be solved, allowing the number of people dining to be taken into account. This would also mean that the user wouldn't have to create recipes from scratch and could choose from a list. Shopping lists could also act in this way and through the collaboration of users, default meals and shopping lists can be created online, to be downloaded when they are needed. To prevent the application becoming useless in cases where there is no signal, some default options could already be installed.

9.0.3 Functional Test Results

With all this data being stored, it is clear that there are many options available, from editing recipes and shopping lists, booking and view events. This prototype incorporates all the important features and caters for basic needs, however future products will need more flexibility. Table A.7 shows a list of autonomic requirements that transform this application from a prototype, proving autonomous behaviour, to an effective product able to integrate with a person's shopping routine with ease.

The functional tests were designed to prove each component of the application worked as they were supposed to, and determine whether they were robust enough to handle large amounts of data. The concept of sharing data to assist with organising shopping items, lists and recipes, is only useful if the software can handle this flexibility. It is also hard to fit everything on a small mobile phone screen, especially when it comes to editing shopping lists and meals. It takes just as much thought to make simple options to manage complicated tasks, as it is to implement an effective learning algorithm.

All functional test cases passed, however there were a few important features missing in order for the application to be helpful and require a minimal level of intervention. These include making certain components more flexible, such as allowing upcoming events to be viewed and removed. This first release only allows events of the day to be viewed, but this doesn't give the user much time to prepare. It can be said however that this is not an electronic calendar and all the capabilities of one shouldn't have to be included.

Another issue is the amount of information needing to be displayed in order for shopping lists and recipes to be edited. It was very hard to include everything on one small screen due to the flexibility needed and it may be easier to split this into multiple pages. Any solution will have its complications however and they will need to be compared in order to find an optimal solution.

Using Robotium helped discover any broken components that were rarely used. It also ensured that current code was not disrupted when writing new code, since a set sequence of events were written and run after each new change. This was also helpful for finding small problems with the usability of the application, since it is hard to remember what has been tested when actually using the application.

9.0.4 User Feedback

From the public questionnaire, it is apparent that this application is needed and would be useful for many people. Most people mentioned that they went shopping at least twice a week and monitored their shopping item stock levels daily. as well as needing to manage their own meals, they said it would be helpful to share information with others, in order to gain more options for cooking. The questionnaire revealed that a lot of people only have around 12 meals that they cook from each week, which suggests that having other ideas could widen this scope. It was also discovered however that most people don't have time to cook adventurous meals due to there not being

enough time. If everybody shared this view, ideas could be passed around for quick meals to make, as opposed to simply interesting meals. The idea of sharing information is to pass on solutions to common problems, such as short amounts of time to prepare meals and isn't just a way of becoming a better cook.

The focus group used to test the application said it was easy to use and fairly accurate in predicting stock levels. This was expected and as mentioned in the feedback for the functional and unit tests, there are methods to make this application more effective and easier to use. It was clear from the results that a flexible application is very important because everybody wants one application to do everything. This doesn't come easily and doing too much can impair the functionality of certain components, such as synchronising local and global databases. With more experimentation, better ways could be found to make sure all data is up to date, maybe a cloud-based approach combing users could be the answer.

Positive comments mentioned the 4 day forecast showing degrading stock levels helped users prepare, and the shopping check-lists provided more freedom in choosing items to buy. Users also mention that the event alerts were useful, however it would be more effective if events could be viewed for future dates.

9.0.5 Project Reflections

The project as a whole was well timed and helped highlight a lot of insights into how this could be an effective idea. Using a rapid prototyping approach was a good tool for test-driven development, because it helped find problems with each prototype early on. Spacing the releases of each prototype out by a sufficient amount also aided in finding early problems before each new release, which allowed the prototypes to be more robust. Designing the product in stages, whilst producing each product component ensured that the completion date would be met and the design could be adapted if necessary. When production starts, small problems usually arise and allowing the design to change helped prevent this happening with newer prototypes.

Developing this product as an Android application saved a lot of time, because it is basically Java, so there was no need to learn a new programming language. All the resources, such as an Android mobile phone were accessible and installing the Android SDK was very straight-forward. There were a lot of resources on Android programming, as opposed to Apple and Blackberry, which meant it was a lot easier to create each component. One of the more complicated components of the product was connecting to the global database. Since this could be programmed in Java, it didn't take long to find tools for sending data to a PHP file and through that, connecting the the database. After that it was then a case of putting the data together to be sent.

Using a Facade design pattern, made the product easier to read and

debug for errors. Having a single point of access reduced the complexity of the product because less links were needed between the classes, causing less possible errors. If this product were to be developed further, this design facilitates any additional functions and maintains a cohesive structure.

9.0.6 Overall Product Performance

The overall performance was a success, proving that an autonomous shopping manager is possible in a fairly open world. Using the suggested modifications and solutions to some restricting problems, this application could be capable of learning new recipes by itself and provide alternative suggestions through collaborations with others. This however would involve some cloud-based system to maintain different shopping items, lists, and recipes, as well as some way of altering the quantities depending on how many are in a household. This could be done using regression trees. Other classification systems, such as decision trees, could be used to categorise foods into sections such as vegetarian, gluten free and dairy. The general database design would not need to change, since all tables are independent of each other, however another table may need to be added to link shopping items with categories, as well as some processing algorithm to categorise this. There are already products that can combine user data and provide alternatives, such as the grocery list managers mention previously, so to make this application stand out, it needs to at least incorporate all the common features of a shopping manager application.

9.0.7 Future Work

Although the product is a fully working application, capable of performing all of the initial requirements, there are a few additions that would make it more useful. one idea gained from the prototypes, that didn't get implemented, was notifying the user via email about empty stock. This could be very beneficial if their phone was inaccessible. The option of showing available meals, was very popular, however it doesn't prevent these meals being planned and no longer being available at the time. Small changes such as these would be a useful addition to future releases, which should attract more users. Releasing additional features like this should keep the public interested and allow the application to stay up-to-date. Since a website is also available to use, updates and other information could be displayed there, allowing fans to sign up, keeping in touch with future updates. New APK (Application Package) files could be downloaded and feedback can be received using discussion boards etc. Having many sources like this keeps people involved and allows them to feel like they are contributing to future work.

Chapter 10

Conclusion

For a system to be completely autonomous, it needs to be able to adapt to new scenarios, without the need for human interaction. This is usually easier if it could collaborate with other systems and learn from its own mistakes, fixing itself when it is broken. To determine whether assistive technologies such as personal organisers and email managers can become truly autonomous, a shopping manager was created that could learn a person's shopping habits. This had various components that could assist with managing shopping lists and recipes, as well as plan these in the future. The outcome of this experiment was to predict whether the idea of an autonomous system could be expanded to cover personal assistants and other autonomous creations. It was a way of acknowledging their limits, finding usages and attempting to change the way we see assistive technologies today. Eventually these limits could be expanded further, altering cookers to learn how long different foods

take to cook, coffee machines, brewing coffee at busy times in the day, or even motor vehicles that remember routines and drive you there automatically. The question is, how far can technology go in creating a world where everything is taken care of, and everyone just goes along for the ride? More importantly, how far are we willing to go before things go too far and we have to take some control back?

The results from the test application had shown a lot of positive conclusions, however there is a lot of work to be done in order to create a system to answer the question posed. It is reasonable to say that with enough work, it is possible to create something that can learn almost anything we can think of, but what about the tasks we cannot think of? Although the real world is dynamic and continuous, it can be argued that it is a closed world, because it is limited by the thoughts of those who live in it. Feats of engineering such as flight were not thought possible until the Wright brothers' successful experiment in 1900, creating a world with only a subset of modern day possibilities.

From the results of the unit tests and the amount of work that went into perfecting a simple learning algorithm for working out item decay rates, it is clear that there needs to be a more sophisticated way of learning. There were a lot of exceptions that became apparent throughout the project, suggesting just how unpredictable people are. In order to maintain an open system, it needs to be more aware of events happening around it, working with other systems to solve complex problems and learn new skills. Much

like the anatomy of the human brain, there needs to be a network of decisional components contributing to related solutions or remaining neutral if unrelated.

Artificial Neural Networks are a promising model for the human brain, allowing neurons to fire for different decisions, using a synapse with a weighted threshold. In respect to the shopping manager, it would be more effective if it were a neural network that evolved around a person's routine, because every component can change radically, adapting to completely new situations.

Putting autonomous systems inside their own world, allowing them to communicate via ontologies is one way of getting them to collaborate. However, these worlds need to be designed for some purpose, posing the question of whether it's inhabitants are really acting autonomously after all. Using a subsumtion architecture is one way of allowing these intelligent agents to make their own decisions, using groups of beliefs, desires and intentions.

The feedback from the questionnaire suggested that the idea of an autonomous technology would be of great use, if it was free to make its own assumptions, without the need for extensive interaction. However some responses suggested that it would not be feasible to live in a world where machines acted on their own, because we like to be in control of situations. A system that can be trained is desirable, however one that can think for itself is unnerving. Maybe due to this limit of being in control of one's self, completely autonomous systems have to live in a closed world.

The question *Can a machine be solely responsible for a person's daily*

routine?, is one with many complications, beyond technological capabilities, and extending thoughts about what makes us human. Limitations such as ideas from human thought, restrictions on privacy and the right for control over mechanical minds, suggest that a completely autonomous system is a possibility, but is not desirable within a political society.

Acknowledgements

With thanks to Jim Smith for helpful feedback throughout the project, as well as James Ivings and Ben Simpson for using the application. This dissertation would not have been possible without their support and the valuable feedback gained from the focus group.

Appendix A

Diagrams, Tables, Information

A.1 Milestone Identification

Activity	Estimated Completion
Literature Review	4 weeks
Research Machine Intelligence	6 weeks
Research Mobile Applications	8 weeks
Investigate Stock Control Systems	10 week
Review Current Products	13 weeks
Design Features Of Application	15 week
Develop Shopping List Feature Of Application	19 weeks
Develop Meals List Feature Of Application	23 weeks
Develop Calendar Feature Of Application	25 weeks
Complete First Prototype	26 weeks
Complete Second Prototype With Learning	30 weeks
Analyse And Evaluate	34 weeks
Complete Report	36 weeks

Table A.1 shows the milestones that need to be met, as well as how long each stage should take. Any additional features will be added at the end, depending on time restraints.

A.2 Database Design

Users table

ID	Username	Password	Email Address
Alex	1	alex123	alex@yahoo.com

Items table

ItemID	ItemName
1	Onion

Users And Items table

ItemID	price	quantity	UserID
1	1.00	3	1

Users And Lists table

UserID	ListID
1	1

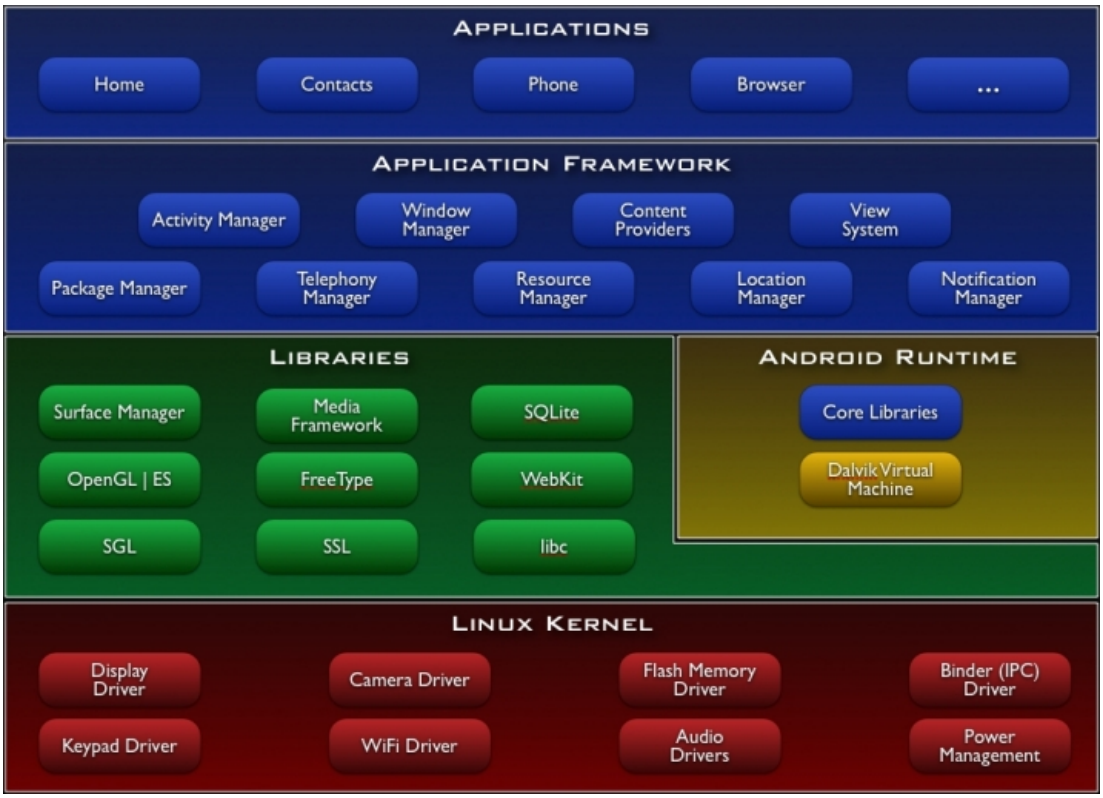
Lists And Items table

ListID	ItemID	ItemQuantity
1	1	2

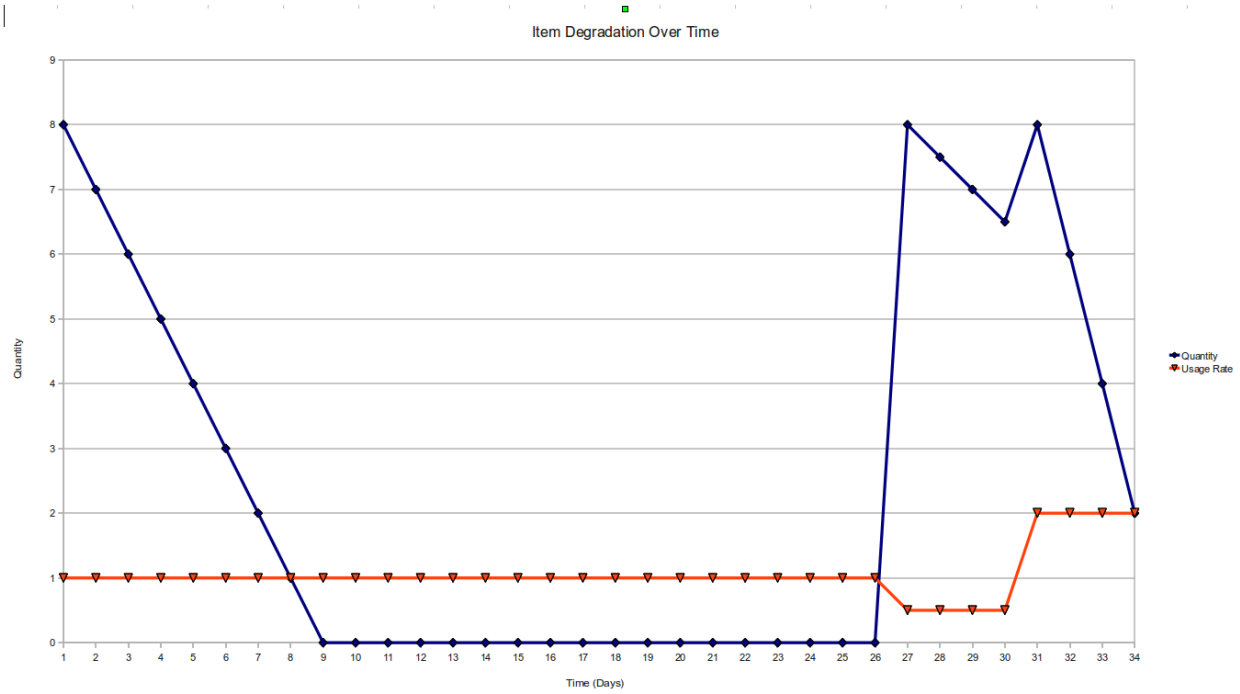
A.3 Product Requirements

Requirement	MUST	SHOULD	COULD	WONT
Adding Items	x			
Editing Items		x		
Adding Lists	x			
Editing Lists		x		
Adding Meals	x			
Editing Meals		x		
Planning Lists			x	
Planning Meals			x	
Viewing Dates			x	
Providing warnings for upcoming dates			x	
Self-Generated Lists			x	
Link With Website			x	
Warnings about upcoming dates and empty stock			x	
Provide Alternatives				x
Link With Other Users				x

A.4 Android Architecture



A.5 Testing The Learning Algorithm



A.6 Functional Test Cases

Test Case	Pass Rate	User Rating
Adding Items	100%	10/10
Editing Items	100%	9/10
Adding Lists	100%	8/10
Editing Lists	100%	7/10
Adding Meals	100%	8/10
Editing Meals	100%	7/10
Planning Lists	100%	10/10
Planning Meals	100%	10/10
Viewing Dates	50%	5/10

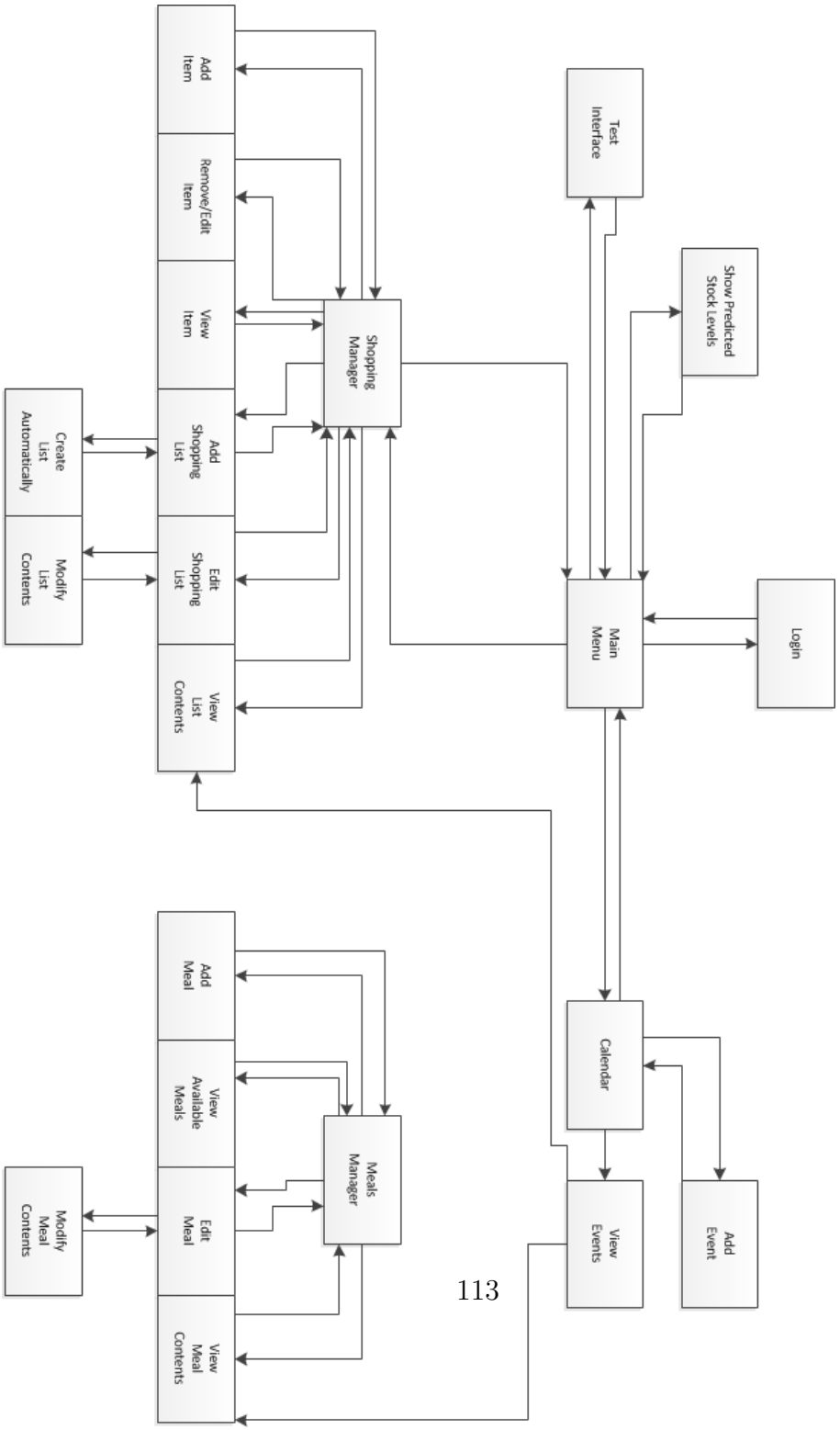
A.7 Autonomic Requirements

Requirement	Implemented/Future Work
Create shopping list based on empty stock	IMPLEMENTED
Provide warnings of empty stock	IMPLEMENTED
Provide warnings of upcoming dates	IMPLEMENTED
Alter item quantities after meals and shopping trips	IMPLEMENTED
Alter usage rates after each shopping trip	IMPLEMENTED
Alter usage rates over a period of time	FUTURE
Suggest alternatives based on other users	FUTURE
Provide initial usage rates using regression trees	FUTURE

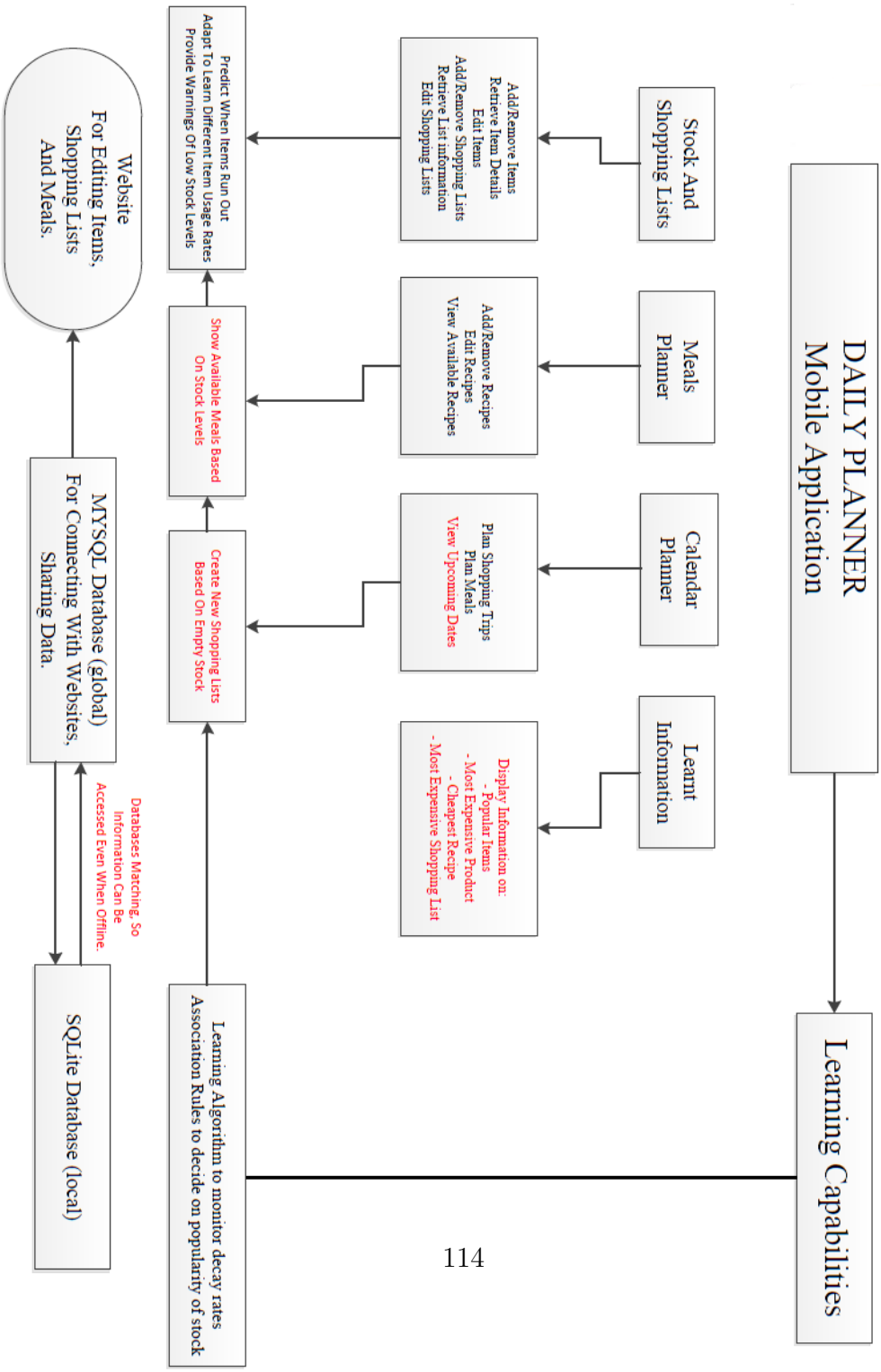
A.8 Public Questionnaire Results

Question	Majority Comments
How often do you need to go food shopping?	Twice a week
How often do you monitor item stock levels?	Daily
How much time do you spend working out meals to cook?	1 hour per week
Could you benefit from a mobile application assistant?	Yes
Would you be happy sharing recipes with others?	Yes
Would you be happy sharing item usage statistics with others?	Yes
How much time do you get to try new recipes?	1 day per week
Do you stick to a set number of meals?	Mostly
How many different meals do you cook regularly?	12 - 15
Would a meals suggestion option be useful in a mobile application?	Yes
How easy could you traverse through pages?	very easily
How accurate was the predicted stock levels?	fairly accurate
How useful is it to have warnings for events and low stock levels?	Very useful
Do both the global and local databases synch properly?	Mostly
How flexible is the application?	Very
Other Comments:	
Application could allow for products to be removed	
The 4 day stock level forcast was very useful	
Being able to check items off with shopping lists made things easier	
Event alerts was great in the form of notifications	
However it would be useful to be able to view upcoming events	

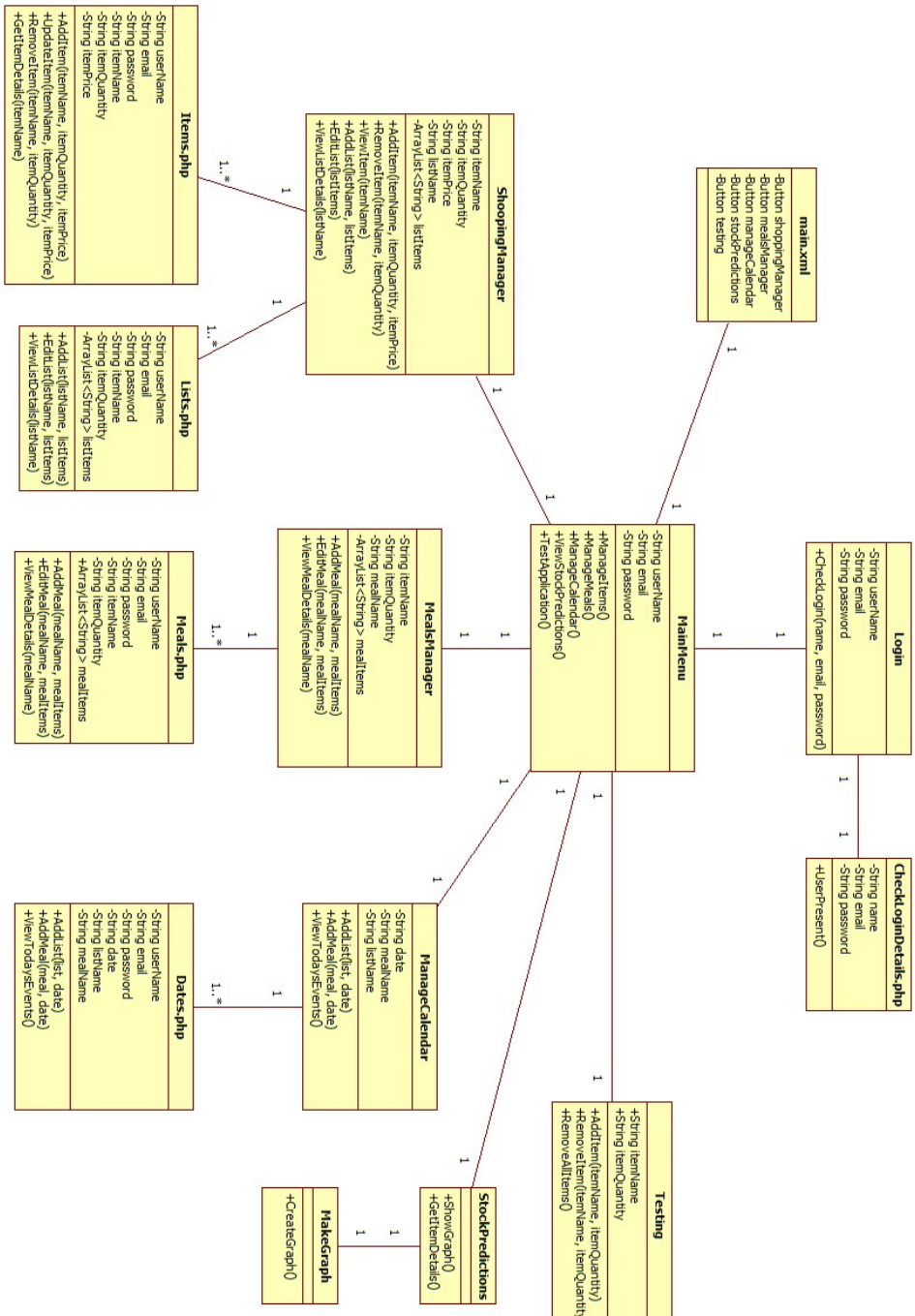
A.9 Use-Case Diagram



A.10 Product Components Overview



A.11 Application UML Diagram



A.12 Inventory Management

A.12.1 Counting and Monitoring of Inventory Items

In order for stock to be managed properly, the system needs to keep track of how much is being stored and how much is being dispatched. This is relatively easy when managing a business that only handles finished goods, but it is much more complex when there is a conversion between materials and finished products. The most common use of a Stock Control System is continuously counting finished products for sale. This is because items valued at retail price have greater market and offer a more accurate representation of a company's profit margins.

A.12.2 Recording and Retrieval of Item Storage Location

In order for stock to be managed accurately, there must be a good database and management system. Items need to be stored efficiently and categorised accordingly. This speeds up database queries and reduces errors. If the database is designed correctly, it should be able to integrate with other applications without having to use complex decoding algorithms. This goes hand-in-hand with a well designed user interface, which provides simple instructions for performing complex database look-ups. This is helpful for business clerks who are unfamiliar with the stock system.

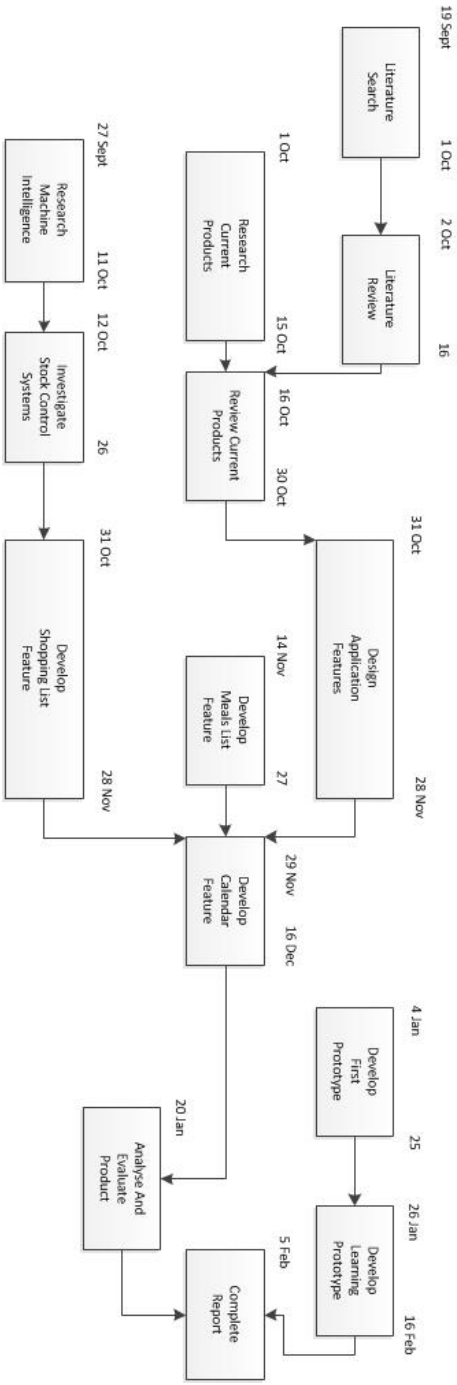
A.12.3 Recording Changes to Inventory

For an Inventory Control System to learn changes in stock and update itself, it needs to process some input. A database is the common approach to storing the relevant information, but it is important for the system to incorporate all its functionality, such as purchasing items, handling returns and managing prices. For the database to update effectively, the system needs to provide some sort of triggering mechanism, publicising any changes in stock. There also needs to be some sort of feedback from the database to the system, confirming the changes that have been made correctly. This should be part of a data quality control system, making sure all stock being managed by the system is stored correctly in the database.

A.12.4 Anticipating Inventory Needs

One of the most important features of an Inventory Control System is that it keeps inventory levels at a minimum. The system should only keep enough stock to prevent a stock-out situation, where the company has nothing left to sell. In order to achieve this, the system needs to anticipate stock levels and predict when things will run out. It also needs to adapt to changes accordingly, without overstocking. If the system is sound, it should be able to predict rises and falls in stock levels, producing graphs and other representations for businesses to analyse. This helps with making purchase volume decisions and setting aside funds for new purchases, as there are fewer risks.

A.13 Work Breakdown



A.14 Monthly Time Plan

A.14.1 September

This will be the design phase, deciding on key features of the first prototype. Once these features have been decided, the first prototype can be created, designing more advanced features as later on. This will highlight any problems that may arise and allow changes to be made to the design, based on what is realistically achievable.

A.14.2 October

This is where the shopping list and item manager will be developed. This will manage all the items and their details, as well as handling shopping orders. All the data will be stored in a MySQL database online. This will be synchronised with a website showing similar information.

A.14.3 November

The second part of the product will be managing meals and ingredients. The user will be able to store recipes on their phone. This will also link with their stored items, providing information such as the price of meals and whether ingredients are in stock.

A.14.4 December

The second and third parts of the product will be completed: managing tasks and budgets. The user will be able to plan shopping trips, meal times and other such tasks, keeping track of their daily routine. They can also work to a weekly budget, keeping track of their finances.

A.14.5 January

After Christmas some learning will be added to the product. It will be able to predict when items run out, notify the user of shopping trips and meals, and warn them if of future stock levels. Other intelligent features will include notifying the user via email, creating shopping lists automatically and providing lists for special occasions.

As well as working on the product, valuable feedback from my prototypes will be gathered, so the design can be adapted to fit with the customer's preferences. This will involve distributing the application to as many phones as possible and letting lots of users try it out for themselves.

A.14.6 February

This will be the test phase, where the final product will be distributed to the public for them to try out. A test scenario will also be created, where test data will be sent to the device over a period of time. This will test out the application's learning functionality, to make sure item quantities are being formulated correctly. Any new features as a result of the feedback gained, will be designed and worked on if there is time.

A.14.7 March

The last month will involve finalising the latest prototype and ironing out any small problems turning it into a robust piece of code. Final feedback will also be gathered, so that a conclusion can be drawn based on the original intentions of the report.

Bibliography

A. Newell, A. S. (1961) *Gps, a program that simulates human thought*, http://circas.asu.edu/cogsys/papers/newell-simon.GPS.pdf. [Online; accessed 26-November-2011].

Ableson, F. (2010) *Using xml with android*, http://www.ibm.com/developerworks/xml/library/x-andbene1/. [Online; accessed 28-January-2012].

The Advantages and Disadvantages of MySQL (n.d.) http://ezinearticles.com/?Advantages-and-Disadvantages-to-Using-MySQL-Vs-MS-SQL&id=1559158. [Online; accessed 14-December-2011].

Android-Developers (2012) *The android architecture*, http://developer.android.com/guide/basics/what-is-android.html. [Online; accessed 29-January-2012].

Android Unit Testing (2012) http://developer.android.com/guide/topics/testing/testing_android.html. [Online; accessed 4-March-2012].

Astrid Android Task Manager (n.d.) `https://market.android.com/details?id=com.timsu.astrid&hl=en`. [Online; accessed 15-November-2011].

Beck, K. (2002), Test-Driven Development By Example, Addison Wesley. [Online; accessed 25-March-2012].

Bezos, J. (1994) *Amazon*, `http://www.amazon.co.uk/`. [Online; accessed 29-March-2012].

Bhd, M. (2011) *Maxis*, `http://www.maxis.com.my/personal/about_us/index.asp`. [Online; accessed 9-February-2012].

Brooks, R. A. (1985) *A robust layered control system for a mobile robot*, `http://dspace.mit.edu/bitstream/handle/1721.1/6432/AIM-864.pdf`. [Online; accessed 29-March-2012].

Campbell, S. J. (2011) *Mobile app market report*, `http://fixed-mobile-convergence.tmcnet.com/topics/mobile-communications/articles/140033-mobile-app-downloads-market-expected-reach-58-billion.htm`. [Online; accessed 27-February-2012].

CART (n.d.) `http://www.themeasurementgroup.com/definitions/cart.htm`. [Online; accessed 12-February-2012].

Chou, P. (1991) *Optimal partitioning for classification and regression trees*, `http://www.cs.nyu.edu/~roweis/csc2515-2006/readings/chou_pami.pdf`. [Online; accessed 12-February-2012].

Classication and Regression Tree Methods (n.d.) `http://www.stat.wisc.edu/~loh/treeprogs/guide/eqr.pdf`. [Online; accessed 10-March-2012].

CMU Testing (n.d.) `http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/`. [Online; accessed 4-March-2012].

Crowdsourcing Debate (2009) `http://communitygirl.wordpress.com/2009/10/01/five-problems-with-crowdsourcing/`. [Online; accessed 10-January-2012].

The DARPA Challenge (n.d.) `http://www.darpagrandchallenge.com/`. [Online; accessed 10-March-2012].

Deductive And Inductive Reasoning (n.d.) `http://www.nakedscience.org/mrg/Deductive%20and%20Inductive%20Reasoning.htm`. [Online; accessed 10-March-2012].

Deep Space 1 (n.d.) `http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1998-061A`. [Online; accessed 5-November-2011].

FIPA (2009) *The foundation for intelligent physical agents (fipa)*, `http://www.fipa.org/`. [Online; accessed 29-March-2012].

FoneMonkey (n.d.) `http://www.gorillalogic.com/fonemonkey`. [Online; accessed 4-March-2012].

Gaudiosi, J. (2012) *Hivemind*, `http://www.huffingtonpost.com/2012/01/02/hivemind-the-sims-will-wright_n_1179594.html`. [Online; accessed 9-January-2012].

Geotasks Task Manager (n.d.) `http://www.geotasks.com/`. [Online; accessed 15-November-2011].

Grovery List Manager (n.d.) `http://greatgrocerylist.com/`. [Online; accessed 15-November-2011].

Hayes, B. (2008) *Cloud computing*, `http://dosen.narotama.ac.id/wp-content/uploads/2012/01/news-cloud-computing.pdf`. [Online; accessed 25-November-2011].

The History of Artificial Intelligence (n.d.) `http://library.thinkquest.org/2705/history.html`. [Online; accessed 10-November-2011].

Hope, C. (2012) *The invention of unix/linux*, `http://www.computerhope.com/history/unix.htm`. [Online; accessed 10-February-2012].

Howe, J. (2008) *Crowd sourcing*, `http://www.openlettersmonthly.com/oct08-jeff-howe-crowdsourcing/`. [Online; accessed 9-January-2012].

InFlow (2006-2011) `http://www.inflowinventory.com/`.

Introduction To Test Driven Development (TDD) (n.d.) `http://www.agiledata.org/essays/tdd.html`. [Online; accessed 25-March-2012].

Inventory Droid (n.d.) `http://romisys.tech.officelive.com/InventoryDroid.aspx`. [Online; accessed 25-November-2011].

Italia, T. (2003) *Jade*, `http://jade.tilab.com/`. [Online; accessed 29-March-2012].

Ivings, J. (2011) *Phonegap*, `http://www.ivings.org.uk/category/phonegap/`. [Online; accessed 27-February-2012].

J. Austen-Smith, J. B. (1996) *Condorcets jury theorem*, `http://www.jstor.org/pss/2082796`. [Online; accessed 10-January-2012].

JUnit Unit Testing (2012) `https://supportweb.cs.bham.ac.uk/docs/tutorials/docsystem/build/tutorials/junit/junit.html`. [Online; accessed 4-March-2012].

Linear Regression (n.d.) `http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm`. [Online; accessed 10-March-2012].

The Logic Theorist and Its Children: AI in Action (n.d.) `http://www.cs.swarthmore.edu/~eroberts/cs91/projects/ethics-of-ai/sec1_2.html`. [Online; accessed 26-November-2011].

Magerman, D. (n.d.) *Statistical decision - tree models for parsing*, `http://delivery.acm.org/10.1145/990000/981695/p276-magerman.pdf?`

ip=94.169.108.111&acc=OPEN&CFID=73888666&CFTOKEN=43360004&_
_acm__=1332687500_e30050e9dab48c534bbffd8238dec3b0. [Online; accessed 5-January-2012].

McGarry, M. (2009) *Norbert wiener and cybernetics*, http://telasiado.suite101.com/norbert-wiener-and-cybernetics-a177496.

Mister Lister Grocery List Manager (n.d.) http://itunes.apple.com/gb/app/mister-lister-grocery-list/id301010289?mt=8. [Online; accessed 15-November-2011].

Moore's Law (2012) http://www.intel.com/about/companyinfo/museum/exhibits/moore.htm. [Online; accessed 7-November-2011].

Murch, R. (2004), Autonomic Computing.

Omidyar, P. (1995) *Ebay*, http://www.ebay.co.uk/. [Online; accessed 29-March-2012].

Oracle - Databases (n.d.) http://www.oracle.com/us/products/database/index.html. [Online; accessed 14-December-2011].

Osherove, R. (2009), The Art Of Unit Testing, Manning Publications Co. [Online; accessed 4-March-2012].

Perceptron Learning Rule (n.d.) http://hagan.okstate.edu/4_Perceptron.pdf. [Online; accessed 11-March-2012].

PhoneGap (2012) *Phonegap presentation*, `http://phonegap.com/`. [Online; accessed 27-February-2012].

P.N.Johnson-Laird and P.C.Wason (1977), Readings in Cognitive Science, Cambridge: Cambridge University Press, pp. 355-376.

Quick, D. (2009) *U.s. biological machines*, `http://www.gizmag.com/bacteria-powered-machines/13642/`. [Online; accessed 8-February-2012].

Robotium (n.d.) `http://code.google.com/p/robotium/`. [Online; accessed 4-March-2012].

Rose, M. (2011) *Hivemind responses*, `http://www.gamasutra.com/view/news/38601/Will_Wright_Reveals_Personal_Gaming_Project_Hivemind.php#comments`. [Online; accessed 10-January-2012].

Selenium (n.d.) `http://seleniumhq.org/`. [Online; accessed 4-March-2012].

The Sims (n.d.) `http://thesims2.ea.com/index.php`. [Online; accessed 9-February-2012].

Vanderbilt, T. (2012) *Autonomous vehicles*, `http://www.wired.com/autopia/2012/02/autonomous-vehicles-q-and-a/`. [Online; accessed 8-February-2012].

What Makes a Good Inventory Control System? (2011) `http://thextonarmstrong.com.au/ME2/dirmod.asp?sid=&nm=&type=`

news&mod=News&mid=9A02E3B96F2A415ABC72CB5F516B4C10&tier=3&nid=B3AB79CADA28479DA3EE38AAE7259708.

Whitla, P. (2009) *Crowdsourcing and its application in marketing activities*, http://www.cmr-journal.org/article/viewFile/1145/2641:/. [Online; accessed 10-January-2012].

Wooldridge, M. (2009), An Introduction To Multi-Agent Systems, Second Edition, Bell And Bain. [Online; accessed 29-March-2012].