BURGE HUGHES WALSH

The Systems Engineering Tool Box

Dr Stuart Burge

"Give us the tools and we will finish the job"

Winston Churchill

Functional Modelling (FM)

What is it and what does it do?

Functional Modelling is a tool that allows a team or an individual to produce a behavioural/operational model of an existing or planned system. The resulting model shows the system functionality and the logical interconnections between that functionality. In essence, it describes how the system functionality has to cooperate to deliver the Operational Requirements[1] of the system. By constructing the model, it is possible to:

- Deduce the necessary system functionality.
- Test out the basic operational concept.
- Determine potential (logical) system interfaces.

Functional Modelling uses three sub-tools to construct a model of the system of interest:

- The **Function Flow Diagram**[2] (FFD) is a network representation of the system. It portrays the system in terms of its component functions and the logical interdependencies or "flows" between the functions. One of the beauties of the Functional Flow Diagram is its simple diagramming conventions. This minimum set enables the team to concentrate on discovering and understanding the system rather than the modelling technique. The conventions are shown in Figure 1.

[1] The Operational Requirement is defined as "the major purpose of a system (i.e. what it fundamentally does; its capability) together with the key overarching constraints (that define the context of the system)".
[2] Software Engineers may have come across a similar modelling method called Data Flow Diagrams. These have similar diagramming conventions and use a forth symbol to represent data stores. In general system modelling (as opposed to software intensive system modelling) the use of a data store symbol is less useful and has been omitted. In essence Functional Flow Diagrams are a generalisation of Data Flow Diagram.

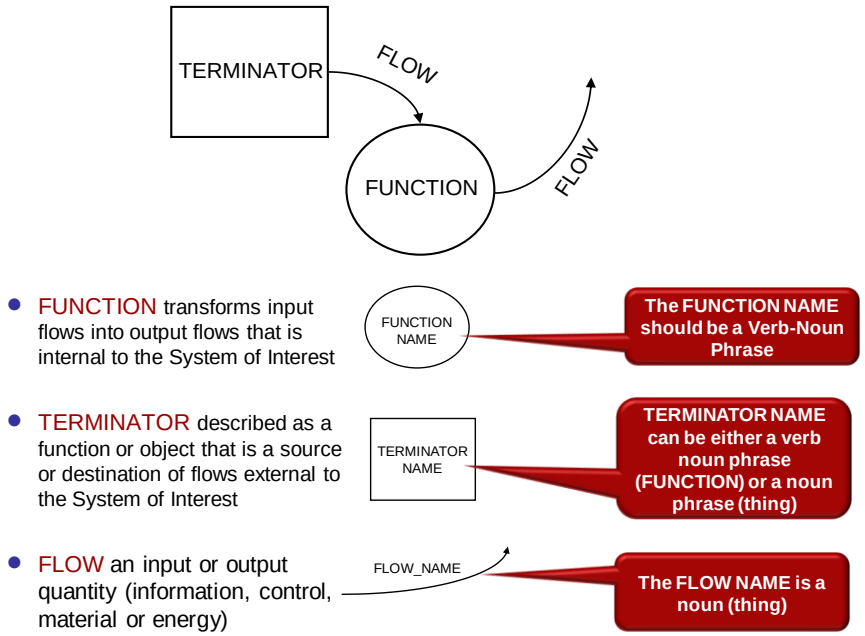


Figure 1: Function Flow Diagram Modelling Conventions

- The **Flow Dictionary (FD)** complements the Functional Flow Diagrams by documenting the flows found on any of the diagrams. It is a set of definitions which declare the component elements of each flow, and the relationships that apply among them. The flows represent the logical interfaces of the system some of which some will become real interfaces.
- The **Function Specification (FS)** specifies the component functions by defining the transformation that converts inputs to outputs.

These sub-tools allow for the construction of a model which shows all the relevant details, but are simple to follow and understand as indicated in Figure 2.

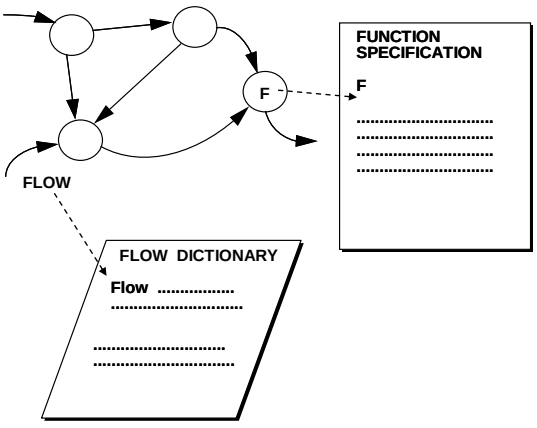


Figure 2: The three sub-tools build an unambiguous model

Why do it?

Customer/stakeholder requirements are never complete, consistent, unambiguous or even correct. Functional Modelling provides a way of logically exploring such requirements in order to address their deficiencies and help to build a complete and consistent set of measurable system design requirements. This exploration leads to the generation of new requirements and the clarification of existing requirements. The result is a model of the system requirements that emphasises the functional operation of the system.

The benefits of Functional Modelling are:

- Diagrammatically building a logical model of how the system functionality is related. It helps the modelling team logically work through the functionality of the proposed or existing system and thereby discovering and generating functionality (requirements) missing from the original customer requirements.
- It identifies the potential interfaces of a system and encourages their clear definition.
- It encourages the team to consider not only the system of interest but also the wider system. All systems operate in an environment; failure to pay attention to that environment will lead to failure.
- It can help to gain more understanding about the system of interest and what all the customers' expect.
- Models that can help in showing the customer that you understand what they are after.
- When used within a team context, it allows the whole team to share information and agree at a common understanding.

Where and when to use it?

Functional Modelling is used to help understand and engineer a set of requirements. It is particularly useful when we have:

- Limited information from the customer such as an operational requirement or idea of a potential operational requirement. In this situation, we use Functional Modelling to derive, organise and document requirements.
- The need to check for completeness and consistency of requirement. This is frequently necessary for safety critical systems.

Functional Modelling can be used in three situations:

1. Analysis and modelling of an existing system.
2. Analysis of an existing system together with the development of modifications to that system.
3. Analysis and modelling of an entirely new system.

Who does it?

Functional Modelling can be used by the individual or a team. Whether it is team or individually based depends on the problem being tackled and the phase of system development. Team-based use is recommended during the early stages of system development. This is because most of the requirements engineering takes place at this stage and developing a common understanding amongst the team is highly desirable. Later when higher levels of the functional model are reasonably mature, detailed aspects are best addressed by individuals. Unfortunately, what happens in practice is often the other way round! Typically this is a consequence of poorly resourced teams during the early stages of system development and especially during the biding type activities. Later, when the deficiencies of the functional model are discovered, management attempt to resolve these by “throwing bodies” at the modelling activity. This never works satisfactorily.

In the early stages the team size need not be large. Indeed, a very small team of 3 - 5 works best: provided of course there is sufficient knowledge.

There is great benefit in terms of quality of output and time efficiency if the modelling sessions are facilitated by a modelling craftsman. Such people, who are familiar with to tool and its use, can often help to choose model structures that require little subsequent iteration.

How to do it?

The concept

Before giving a detailed description of Functional Modelling it is important to reflect upon the modelling concepts which predicate its use. These are:

- **Diagramming**: natural languages are ambiguous and the use of textual methods to convey complex information will lead to errors. The prime reason for this is the relative inability of text based descriptions to convey structure. Diagrams on the other hand are very good at being able to convey structure.

- **Information hiding**: most systems are too complex to represent on one diagram, but humans can understand any degree of complexity if it is presented in small ‘chunks’ together with a ‘map’ of how the chunks are structured together. Combined with diagramming, information hiding allows complex systems to be represented in a logical hierarchical fashion. The highest levels leave out the fine detail and concentrate on the essential information at that level. The information that is hidden at the higher level can be revealed in lower level diagrams. This approach has many names including top-down, modular, and abstraction. The latter name however also has another very important meaning.

- **Abstraction**: although sometimes used imply information hiding, abstraction also refers to the diagramming conventions whereby a single convention is used to represent different real world items. For example, the single diagramming convention of an arrow to represent an input or output flow is used represent material, energy, information and control type flows.

The Process

The process for functional modelling is shown in figure 3

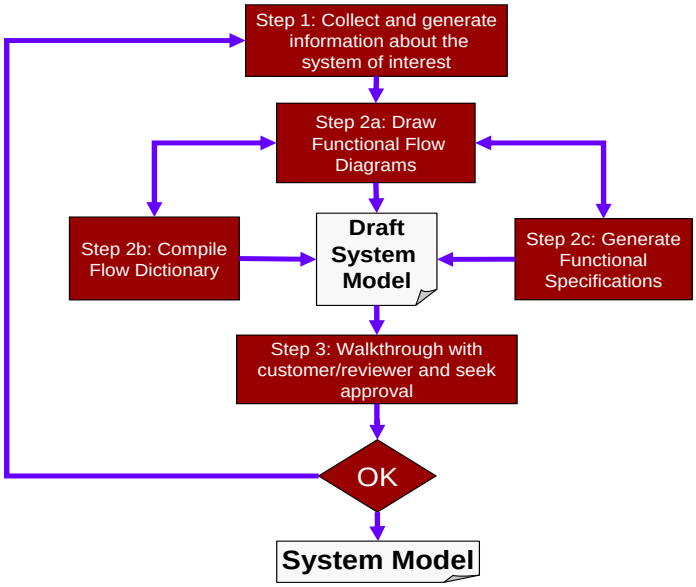


Figure 3: Process for Functional Modelling.

The first step is to collect information about the system of interest. This is achieved via a variety of means that include:

- Viewpoint Analysis.
- Customer Requirements documents.
- Textual Analysis.
- Use Cases.
- Involving customers and users.
- Involving experienced staff.

The purpose of this step is to gain the information necessary to produce a preliminary set of Functional Flow Diagrams together with the basic Flow Dictionary and Functional Specifications. This is step 2. The draft Functional Flow Diagrams can then be shown to the customer/user or some other reviewer, walking them through each diagram, explaining the meanings of the flows together with a brief description of each function. The purpose of the review or walkthrough is to seeking approval of the model. The process of constructing a model and seeking approval is an iterative one. However, the eventual outcome is a validated model of the system requirements.

The Role and Purpose of Functional Flow Diagrams (FFD)

The role of the FFD is to present a description of the system in terms of its component functions and how those functions have to interact (the flows) to deliver the intended system outcome (the operational requirement). By attempting to describe how the system functionality has to logically interact helps, the diagramming team, uncover previously unidentified functions (requirements). This aspect is one of the prime reasons for undertaking functional modeling. The tool has a second purpose which is to identify the potential interfaces of the system. Each flow on a FFD is a potential or logical interface between functions. That is, logic says there should be a connection or dependency (a flow) between two functions. The key point here is that some of these logical interfaces will become real interfaces when the system is realized. This is critical knowledge because experience shows that most system inefficiency and ineffectiveness occurs at the interfaces. FFDs therefore highlight all the potentials interfaces and therefore the areas where problems are likely to occur.

This philosophy is further reinforced by downplaying control and decision making within the system. There is no “decision” symbol on a Functional Flow Diagram. Indeed, functional Flow Diagrams deliberately de-emphasize the flow of control within the system in order to emphasize the logical interfaces. Indeed, decisions and control aspects are hidden within functions and defined in the Functional Specifications.

The flows on a FFD can represent physical qualities, energy flows, control flows or information flows. It is the ability of the FFD to show these various types of flow simply that makes it such a powerful tool.

In addition to the documentation and modelling benefits, the FFD provides a very useful partitioning of the system into smaller “sub-systems” or, at the lowest level, into a set of interconnected primitive functions. It shows all the logical interfaces and the appropriate Flow Dictionary entry can be used to specify the detail of any one of these.

While the FFD can be used to describe any system by showing a set of connected primitive processes, it is clear that for anything larger than a very small system, a single FFD will be cluttered with detail and much too extensive to be comprehended easily. Large systems, therefore, require a 'top-down' treatment is used to produce a hierarchical set of FFDs. This will allow the use of a single FFD to represent the whole system at a level of detail which can be realistically represented on one diagram. The 'Functions' of this top-level FFD can then be taken in turn and described on lower level FFDs. This activity of successive lower levels of description is continued down to the 'primitive' level or lowest level where functions cannot usefully be described using the diagramming conventions. This point is typically where, to describe a lower level of detail requires, design decisions have to be made. At this primitive level the functions are described rigorously with a Function Specification. The result is therefore a hierarchical set of FFDs sometimes called a 'leveled' FFD set.

Functional Modeling is not without limitations. Like all models, the functional model is a representation of a real system (existing or intended) and cannot capture every detail. The prime limitation is the deliberate avoidance of an explicit time base. The sequence of functionality is implicit rather than explicit making the tool less useful when deal with systems that have time based deadlines[3]. This limitation is more than compensated for by the benefits the simple tool offers. It does, however, mean that on occasions several different models may need to be built for different modes of operation, viewpoints or scenarios. For example it is possible to construct a set of Functional Flow Diagram that defines the deployment or implementation of a system. It also possible to construct an operational model that shows who the system intends to deliver its operational requirement. The two models are definitely linked but it may not be easily possible to construct a single set of diagrams that shows both modes of operation.

Rules for Constructing Functional Flow Diagrams (FFDs)

- The top of the diagram set is a diagram that has the system of interest as a circle (a single function). This diagram captures all the incoming and outgoing flows (interfaces) from the elements that are in the environment of the system of interest. These environmental elements are sources or destinations of flows – terminators – and therefore represented by squares. It is called the CONTEXT DIAGRAM and defines the boundary of the system. These points are shown in Figure 4.

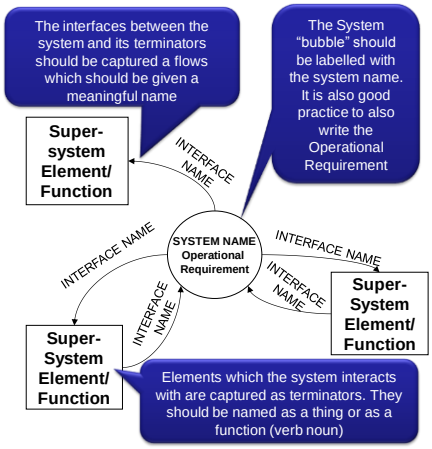
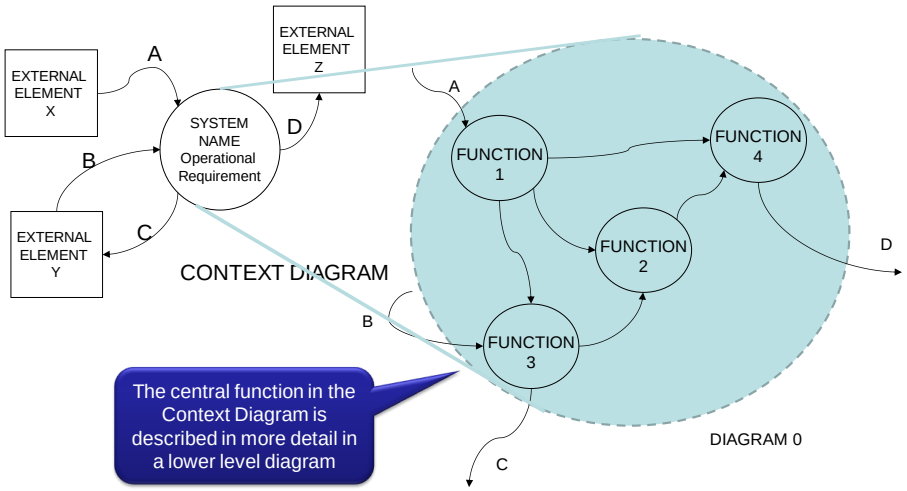


Figure 4: features of the CONTEXT DIAGRAM

The CONTEXT DIAGRAM should reflect the operation of the system and it is worth writing the operational requirement inside the circle representing the system as well as naming the system.

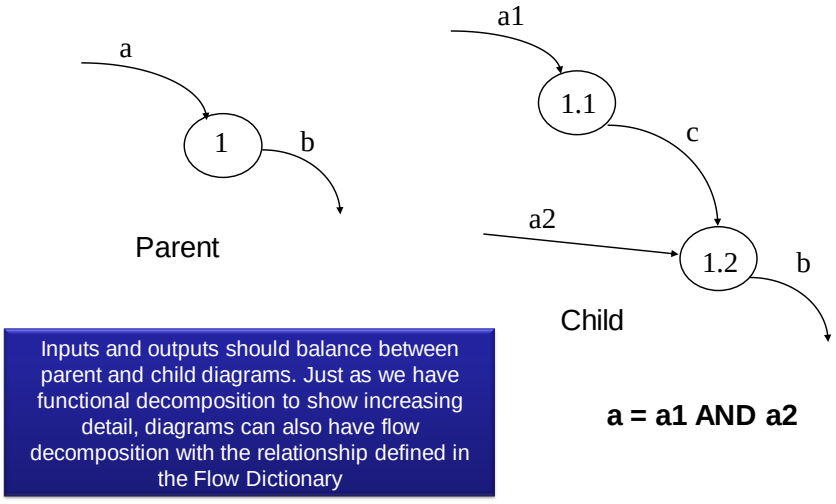
[3] There are extensions to the tool that can be incorporated to help explicitly the time dependent sequence of functionality.

- The Context Diagram is the 'parent' of a diagram representing the first-level of detail. That diagram is in turn parent to a number of 'child' diagrams, which constitute the second-level. There will be as many level two DFDs as there are functions (bubbles) on the parent. Similarly, level-two diagrams may be parent to level-three diagrams, and so on.
- Flows on any diagrams must be declared as “things” using a noun-based description. Flows should never describe an action or activity (i.e. function).
- The first-level diagram is termed DIAGRAM 0. The function bubbles in this diagram are numbered sequentially from 1 as well as given a name that describes the function they perform. In deciding the name it is important to capture it as a verb-noun combination – and action on an object.



- If a function on DIAGRAM 0 contains more detail, a new FFD is constructed and labeled DIAGRAM N, where N is the function number on DIAGRAM 0. The lower-level functions on this new FFD are labeled N.1, N.2, N.3, etc. If any of these lower level functions require further description, again a new FFD is constructed and the simple numbering sequence continues. Care must be exercised when modeling systems of systems, where effectively there are a number of context diagrams and therefore potentially a number of DIAGRAM 0s. In such cases, a “super” CONTEXT DIAGRAM could be constructed and individual system CONTEXT DIAGRAMS can be numbered accordingly.
- A 'balancing' rule applies, such that the flows into and out of a function on a parent diagram are equivalent to the net inputs and outputs to and from a child diagram.

- It is possible to show balanced flows with more detailed representation on the child diagram than on the parent. In that case the DFDs show parallel decomposition, both of data and function. The Flow Dictionary must be used in this instance to confirm that the decomposition is *accurate.* For example:



In this example, Function 1 has been decomposed into two Functions: 1.1 and 1.2. At the same time dataflow 'a' has been decomposed into 'a1' and 'a2'. The Flow Dictionary will define:

a = 'a1' AND 'a2'.

- In order to determine the source or destination of flows which enter or leave any diagram, it is necessary to reference its parent diagram.
- If there is no further decompositioning of a function, then there must be a Functional Specification for it.
- The bottom of the set consists of a number of FFD that show the lowest level of function, called the primitive functions.

Approaches for Constructing Functional Flow Diagrams (FFDs)

Constructing a set of Functional Flow Diagrams is often not easy. This not due to the modelling tool but is a consequence of the inherent complexity of the problem under consideration. Humans, with familiarity, tend to take things for granted and assume they are easy – actually they rarely are. Even what are considered to be the simplest of systems, often under analysis turn out to be surprisingly complex. Constructing the diagrams will be an iterative process; the construction of lower-level diagrams often uncovers aspects that will require the modification of higher level parent diagrams. When modelling a system, existing or intended, it is necessary to be prepared to modify diagrams (several times).

Writing a prescriptive process for constructing a set of Functional Flow Diagrams is not possible. However, it possible to provide a framework:

- Do consider using white-boards for the early drafting work. The initial diagram will require several iterations and a white provides a convenient medium. Furthermore, it is useful if team members can "sketch" out their ideas to show other team members. If white boards are not available, flip charts are an alternative, but are less easy to modify. Software tools are available to capture the outcome, but, in general they are less useful for constructing diagrams using a team.
- Consider the operational view of the system first. It is possible to create many different models of any one system (usually based on phases of the life-cycle of the system). This may well be necessary at some point, but when initiating a modelling exercise it is best practice to start with the operational view, i.e. the system has been designed and installed and consideration is aimed at its day-today operation.
- Start with the CONTEXT DIAGRAM. A team can often quite quickly put together a CONTEXT DIAGRAM but can either
  - Struggle to make progress.
  - Endlessly debate aspects of the diagram.

  In both situations it is necessary to start constructing some of the lower level diagrams in order to resolve or re-invigorate the debate.
- The initial drafting of a CONTEXT DIAGRAM should consider every possible or potential flow. This often results in a very "busy" diagram and there is a tendency to either ignore flows because they are considered not important. It is preferable to capture all these flows and rationalise and simplify the diagram later. Indeed, having captured all the flows the diagram can be simplified by collecting similar flow together and creating a collective name which can be detailed in the Flow Dictionary. The grouping of flows is often easier once a DIAGRAM 0 has been constructed.
- Once a draft CONTEXT DIAGRAM is available consideration should be given to constructing the associated DIAGRAM 0. This diagram is often critical and is the one that will receive the most modification and be the most difficult to draft. The difficulties arise from the need to group lower level functionality in an attempt to comply with the 7 or fewer functions on the diagram. There are two basic strategies that can be adopted:

1. Work forwards and backwards through the "problem" to logically deduce the functionality necessary.
   a. Working forwards by consideration of a scenario and start with a key system input (look for the one that initiates the operational requirement) and ask what happens to it as a "sequence" of functions that leads to an output.

b. Working backwards by starting with a system output (look for the one that relates to the operational requirement) and determine what function would deliver this output, then ask what inputs are necessary, then what functions generate those inputs etc

When working forwards and backwards following the logical sequence of functionality of results in an initial DIAGRAM 0 that has more than 7 “functions”. Care should be exercised not to reduce the readability of the diagram by forcing only 7 “functions”. A readable and understandable diagram with say 9 “functions” is far superior to one with exactly 7 “functions” that is difficult to follow. Moreover, DIAGRAM 0 frequently has to be modified as a consequence of constructing the lower level diagrams.

2. If a Viewpoint Analysis has been undertaken, the Viewpoint Structure Chart is often a good starting point for constructing DIAGRAM 0 – BUT BEWARE while all the functions identified on a Viewpoint Structure Chart should appear on the Functional Flow Diagrams., the grouping on the Viewpoint Structure Chart may not be the best aggregations for developing a simple set of FFDs. This idea of using the Viewpoint Analysis is shown in Figure 5.

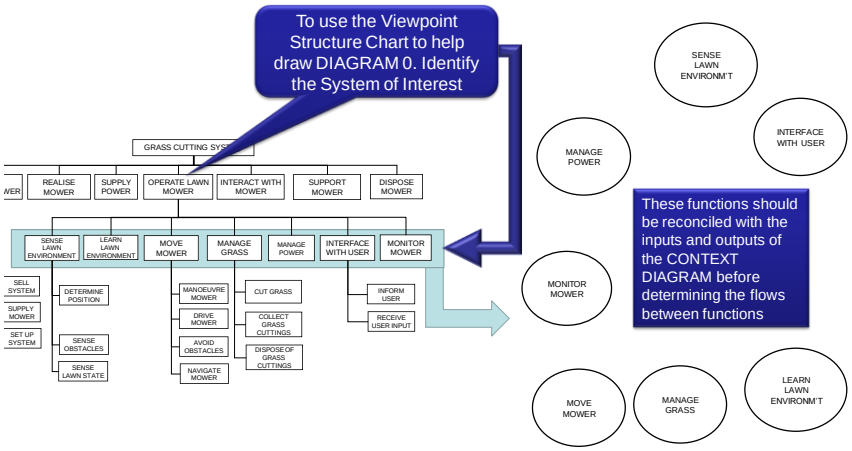


Figure 5: Using The Viewpoint Structure Chart to identify a suitable set of Diagram 0 functions

- As with the CONTEXT DIAGRAM, a team may find itself:
  - Struggling to make progress.
  - Endlessly debate aspects of the diagram.

  In both situations it is necessary to start constructing some of the lower level diagrams in order to resolve or re-invigorate the debate.
- Once draft DIAGRAM 0 exists, the CONTEXT DIAGRAM can be reviewed for consistency and modified accordingly. At this point it may also be possible to rationalize the flows on the CONTEXT DIAGRAM.

- Work should now concentrate on constructing the lower level diagrams. In general, these are easy to construct and it may be expedient for these to be developed by individuals rather than the team. Common sense is necessary here, if there are concerns with the DIAGRAM 0 then in may be best to continue working as a team for some of the diagrams.

The Role and Purpose of the Flow Dictionary

The Flow Dictionary is used to specify precisely what is meant by every flow on every Functional Flow Diagram. In particular it specifies the elements which are contained in each flow. In order to manage complexity, at the highest levels, the flows are best defined as groups of subordinate items. This deconstruction of flows into more detailed flow elements is continued until it cannot be usefully pursued further.

The flows can represent different input/output quantities of:

- Materials
- Energy
- Information/data
- Control.

Determining a suitable name for a flow is important in order to make the FFDs easily comprehendible. Flows labelled “data” or “control signal” are poor choices. Better choices specify “what data” (e.g. sensed temperature data) and “what control signal” (e.g. stock replenishment control signal). The dictionary plays an important role in making sure the flows are clearly understood.

It is also important to remember also that every Flow represents a logical interface (that could be a real interface in the real world) hence the Flow Dictionary is an initial specification the potential system interfaces. In order to reduce ambiguity the Flow Dictionary uses a set of standard notational conventions.

Dictionary Conventions and Notation

The Flow Dictionary makes use of a set of relational operators that allow for the formal definition of any possible dictionary entry. This restricted set consists of simple operators which can be used in combination to construct more complex operators. The set comprises:

IS EQUIVALENT TO
AND
EITHER-OR
ITERATIONS OF
OPTIONAL

An example of a Flow Dictionary definition that uses all five is:

Telephone_directory IS EQUIVALENT TO:
ITERATIONS OF:
EITHER: Business_Name OR: Personal_Surname
AND: OPTIONAL: Initials AND: Address AND: Telephone_number

While the use relational operators brings discipline and consistency to the Flow Dictionary, it is clear from the above example, their use is tedious. Accordingly, a more concise notation is used in practice that comprises:

Operator	Shorthand
IS EQUIVALENT TO	=
AND	+
EITHER-OR	[option1/option2]
INTERATIONS OF	{items}
OPTIONAL	(item)

In addition to this basic notation set, there are a few other conventions for dictionary entries. The iteration brackets can be annotated with upper and/or lower limits

e.g 1{XXXX}8 = from one to eight iterations

8{XXXX]8 = exactly eight iterations

Thus for the telephone directory example

Telephone_Directory = {[Business_Name/Personal_Surname
+(Initials)]+Address+Telephone Number}

The entries in the Flow Dictionary start the process of defining potential or logical interfaces. It is where the logical interface requirements are captured and recorded making it an early form of an Interface Control Document (ICD).

The Flow Dictionary is also where many of the non-functional requirements are captured.

Role and Purpose of the Functional Specification

The purpose of Functional Specifications is to describe in more detail what has to happen inside a particular function on a Functional Flow Diagram. In essence, the Functional Specification of a function will define the rules governing the transformation of flows entering the associated function into flows leaving it. It defines the policy governing transformation (what has to be done) but not the method of implementing it (how it is to be done). The Functional Specifications are also where many of the non-functional requirements, i.e. constraints, are captured. Thus, the non-functional requirements are related in a structured way to the functional requirements.

Only the primitive Functions require a Functional Specification, since higher level functions are simply the combination of lower level ones. However, there may be instances where constraints apply at a certain level. In such cases, constructing a Functional Specification would be valuable.

Functional Specification Conventions and Notation

In order to minimize ambiguity normal English is used to write the Functional Specifications. Instead, some form of specification language is used. These are typically based on high-level procedural software languages which make use of a limited (English) vocabulary and limited syntax. Any suitable high-level language, such as Ada, will suffice. Using software-based languages does offer the potential of compilation and execution as a possible verification and validation route.

In the absence of such software-based languages, Structured English can be used. It, like software-based languages, has a limited vocabulary and syntax. The vocabulary consists of:

- Imperative English language verbs.
- Terms defined in the Data Dictionary.
- Reserved words for logic formulation.

The syntax of a statement is limited to:

- Procedural sentence.
- Closed-end decision construct.
- Closed-end repetition construct.

or combinations of these. Thus to build any required Functional Specification the following constructs are used:

Sequence construct

The sequence construct is a list of simple procedural sentences which are to be applied in the order:

<Procedural sentence 1>
< Procedural sentence 2>
<Procedural sentence 3>
etc

Decision construct

This consists of two possible constructs:

1) **IF** <condition>
<then policy>
OTHERWISE
<otherwise Policy>

2) <Selection policy>
CASE 1: case 1 condition>
<case 1 policy>
CASE 2: <case 2 condition>
<case 2 policy>
etc.

Repetition construct

This consists of a policy which is repeated several times within some specified limit. There are two constructs:

1) **FOR EACH** <item>, **WHILE** <condition>
DO THE FOLLOWING
<policy statements>

2) **REPEAT THE FOLLOWING:**
<policy statements>
UNTIL <condition>

Clearly, to describe any particular function will require combinations of the above. It is possible to describe any function using these constructs resulting in clear, concise and unambiguous definition.

Box 1 and 2 show example of Functional Specifications written in Structured English:

```
FUNCTION 2.3 TEND PLANTS
    REPEAT THE FOLLOWING
        INSPECT PLANTED_VEGETABLES
        DETERMINE appropriate action
            CASE 1:     PLANTED_VEGETABLES wilting
                        water affected PLANTED_VEGETABLES
            CASE 2:     PLANTED_VEGETABLES diseased
                        determine disease type
                        apply appropriate remedy
            CASE 3:     animal attack
                        determine animal type
                        take appropriate action
            CASE 4:     PLANTED_VEGETABLES satisfactory
    UNTIL PLANTED_VEGETABLES ripe
```

Box 1: Example Functional Specification for the Function "TEND PLANTS"

```
FUNCTION 2.1 ACTIVATE MOWER
IF ACTIVITATION on
        DO THE FOLLOWING
            EXECUTE SELF_TEST
            IF SELF_TEST Failed
                GENERATE USER_MESSAGE
                DELIVER USER_MESSAGE
            ELSE IF SELF_TEST ok
                GENERATE USER_MESSAGE
                DELIVER USER_MESSAGE
                ACTIVATE NAVIGATION
            END
        UNTIL POWER OFF
```

Box 2: Example Functional Specification for the function "ACTIVATE MOWER"

The writing of Functional Specifications can provide guidance as to whether the functional decomposition has gone far enough or conversely too far. The basic rule of thumb is that if a Functional Specification exceeds one side of A4 paper, consider a further decomposition in the associated Functional Flow Diagram. On the other hand, if a Functional Specification is less than ¼ page of A4 consider grouping with another function. Again, common sense is important here; the overall purpose of Functional Modeling is clarity of understanding.

What Goes Wrong: The limitations of Functional Modelling

Functional Modelling is a very simple but powerful tool for exploring the requirements of a proposed system or analysing an existing one. Like ALL modelling methods, it has limitations. The following outline these limitations and where possible propose approaches to minimise their effect.

- Functional Flow Diagrams are abstract models that focus on the system's functionality. The resulting model is not a physically related model and may appear to not to reflect current thinking or practice. Teams, particularly inexperienced teams, try to construct a set of diagrams that reflect the likely physical manifestation of the system. This can lead to inconsistencies and difficulty in capturing all the requirements. It can also lead to a biased model that dictates a future architecture. A classic example is the functionality that is concerned with the diagnosis of faults or prognosis of incipient failures. Both functions are the responsibility of the system support system (i.e. the system that will support the system of interest), however, when implemented the functionality may reside in the system of interest. When constructing a model of either system introduces the debate as to where the functionality should be captured.

- To cover the complete operation of a system may require several functional Flow Diagram sets. The modelling method does not lend itself to simultaneously capture multiple modes of operation. Many systems have several different modes of operation (often due to dealing with different scenarios). For example a system may have to be deployed before it is operated, yet constructing a single set of FFDs, would be difficult to achieve. In such cases two sets, one for deploying the system and one for operating the system may best capture the necessary detail. In this case a third set may also be possible that shows the relationships between the two life-cycle modes.
- Modelling architectural functionality is difficult using Functional Flow Diagrams. Product based systems often require some form of casing or chassis. These items do indeed possess functionality, they “protect”, “support”, “align”, etc, but this type of functionality is difficult to capture on a model that is typically operationally based (i.e. a model that is concerned with the functionality necessary for the system to achieve it operational requirement). It is possible to include such architectural functionality, but the model is often too abstract and often causes more issues that it solves. In such cases the best course of action is to recognise the limitations of the model and proceed with the operational functionality only. The use of methods to identify and define the architectural functionality is of course essential.

Success Criteria

The following list represents a set of criteria that have been found to be useful when modelling a system. Ignore them at your peril!

- Team size between five and eight.
- Team constitution covers system life cycle and potential technology.
- Use an experience independent facilitator.
- Plan for a series of half-day events that:
  - Event 1: Draft out Context Diagram and DIAGRAM 0. This is best done on a large whyte board or equivalent. Be wary of constructing the diagrams directly in software! People should be encouraged to draw out their understanding – if they are intimidated by not being able to drive the software they will agree too readily with a team member view rather than explore their view.

    If a Viewpoint Analysis has been undertaken previously considered using this as a “springboard” for the DIAGRAM 0. If time consider some of the lower level diagrams.

Following Event 1 capture the draft diagrams electronically. Attempt to change the layout of functions and terminators to simplify the diagrams. Create, electronically, a draft Flow Dictionary. Attempt to be a precise as possible with the flow descriptions. Although it is unlikely that Functional Specifications can created after event 1, it is very worthwhile defining the functions that have been indentified and also what sub-functions they may contain.

- Event 2: Review the draft Context Diagram and DIAGRAM 0. Determine which lower level Functional Flow Diagrams need construction as a team and which can be performed by individuals. Begin to construct the lower level FFDs, modifying the higher level diagrams as the work proceeds. Attempt to construct all diagrams deemed necessary as a team and allocate the lower level diagrams to individuals for completion.

  Following Event 2 ensure all diagrams are updated and draft diagrams are captured electronically. Send copies of the diagrams to individual team members for comment and review together with deadlines for the submission of draft lowest level diagrams. Request that the Flow dictionary and Functional definitions be updated.

- Event 3: Assemble the team and review the diagram set by walkthrough with, ideally the customer, or an independent reviewer.

Illustrative Examples

One of the difficult aspects of Functional Modelling to capture in a text based description is the iterative nature of model development. To create a “good” set of Functional Flow Diagrams can take several attempts (iterations). Moreover, the final result is often relatively simple and almost obvious and hides the amount of effort expended in its creation. The following aims to show the iterative - almost exploratory - approach by describing the step by creation of a functional model.

In order to make the example meaningful to any reader the situation to be modelled is a familiar one - the domestic kitchen. It has been chosen for many reasons. It is a something that almost every person will have used. It has been chosen because many may not view it as a system – but it is. Also it may be considered to be too trivial – it is not. Indeed, the modelling ubiquitous systems that are often taken for granted will develop an individual’s modelling skills immeasurably.

Starting a model is often the hardest thing! As humans we have a desire to “get it right” and will even shy away from situations in which we are unlikely to get it right first time. Functional Modelling often inflicts this dilemma. The solution is to “have a go”. Figure 6 show the first draft of a CONTEXT DIAGRAM for the operation of a domestic kitchen. It has the following Characteristics:

- It is wrong; it was drawn knowing it would be wrong.
- It is not complete.

However, it has captured my thoughts on paper that I can review and show another human to build a second better version.

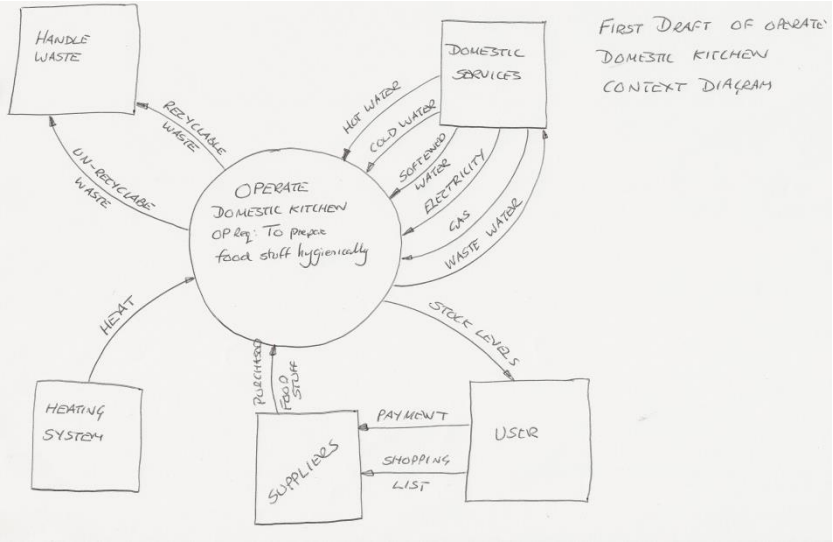


Figure 6: First Draft of a CONTEXT DIAGRAM of the Operation of a Domestic Kitchen

In drawing Figure 6, a number of decisions were taken in order to progress. The first was to decide exactly what aspect of the kitchen to model. Like all systems, a kitchen has a life-cycle, with a number of distinct phases; Design, Build, Operate and Maintain, is one possible breakdown as shown in Figure 7[4].

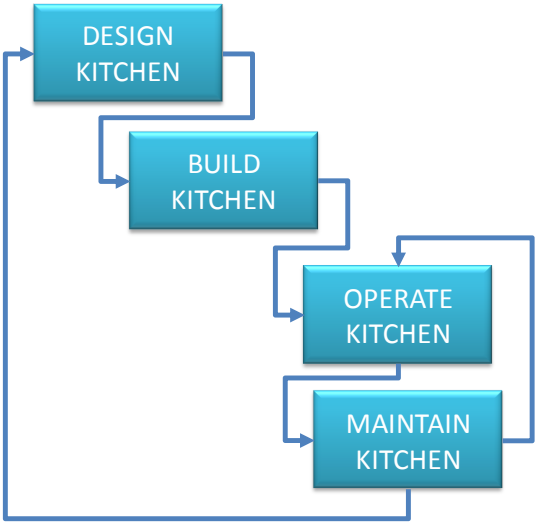


Figure 7: Life-Cycle of a Domestic Kitchen

[4] There are many other possible life-cycle views depending on the level of detail sought or desired. The one presented here represents the basic minimum set of phases.

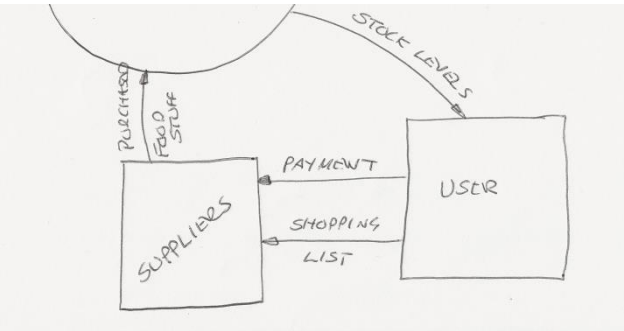
It follows from Figure 7 that is possible to model the complete life cycle of the kitchen. It is also possible to construct models for each of the four phases identified. This in is a general observation: if a complete understanding is required, several models will have to be developed. Often which aspect to model depends upon the viewpoint? If I was a Kitchen Installer, then the focus would be on the box “BUILD KITCHEN”. If I ran a repair business then the “MAINTAIN KITCHEN” would be appropriate. Interesting, however, the “OPERATE KITCHEN” is critical to all of the life-cycle phases. If I am a kitchen designer, builder, or maintainer, I have to know something about the *prime system*: the kitchen itself. Accordingly, Figure 6 the first draft CONTEXT DIAGRAM is concerned with the operation of the kitchen.

Life cycles are not the only reason for necessitating multiple models of a system. Many systems are moded. They have several modes of operation, for example: Deployment, Start up, Operation, Shut down, etc. When attempting to model such systems, a single “all-encompassing” model is often not possible. In such cases it is best to identify the modes of operation and construct a model for each mode of operation. A good example here is of an Autonomous Lawn Mower. The operational requirement of this system is to mow a lawn without ant human intervention excepting initial set up. As such when not mowing the lawn it will have to be stored in some way, it will have to transport itself to and from the store to the lawn and finally cut the grass on the lawn. Constructing a single functional model for these mo des of operation is not impossible but it will lead to a highly abstract model. In such instances it is easier to construct a model for each mode of operation thereby requiring several CONTEXT DIAGRAMs.

Returning to Figure 6, there are several features worthy of note at this point. In no particular order:

- The “bubble” on the CONTEXT DIAGRAM is labelled as verb-noun description – OPERATE DOMESTIC KITCHEN SYSTEM. The alternative label could be “Domestic Kitchen” which is fine but gives the impression it is a physical object rather than the more abstract function. Best practice dictates the pedantic use of verb-noun descriptions.
- To reinforce the functional nature of the modelling approach, the “bubble” is also annotated with the Operational Requirement of the system.

- It is an abstract representation not a physical model. For example the USER is an integral element of the system and will therefore have multiple instances in the model (for example, one of the functions of the kitchen is “prepare meals” this is highly likely to involve the user in some way). The USER also at times is external to the system. For example on Figure 6 there is the following:



Here the USER has been represented as a Terminator, i.e. external to the system. In fact this part of the model is capturing the situation of the USER checking the stock levels to prepare a “shopping list” in order to go shopping to replenish the stock. This, in itself is an interesting view because implicitly the decision has been made as to how something is to be done. A good functional model should not do this – but by capturing this viewpoint now will lead to a better model later.

Drafting a better CONTEXT DIAGRAM often requires knowledge of the next level of detail. In fact, a key modelling skill is knowing when to stop working on a higher level diagram and begin working on the lower level diagrams. Experience shows that if team members end up in circular debate about how to model an aspect it is perhaps time to consider the lower levels. Equally, is progress on a particular level slows too much, starting the lower level diagrams is in order. Figure 8 show the first draft of a DIAGRAM 0 for the Domestic Kitchen example.

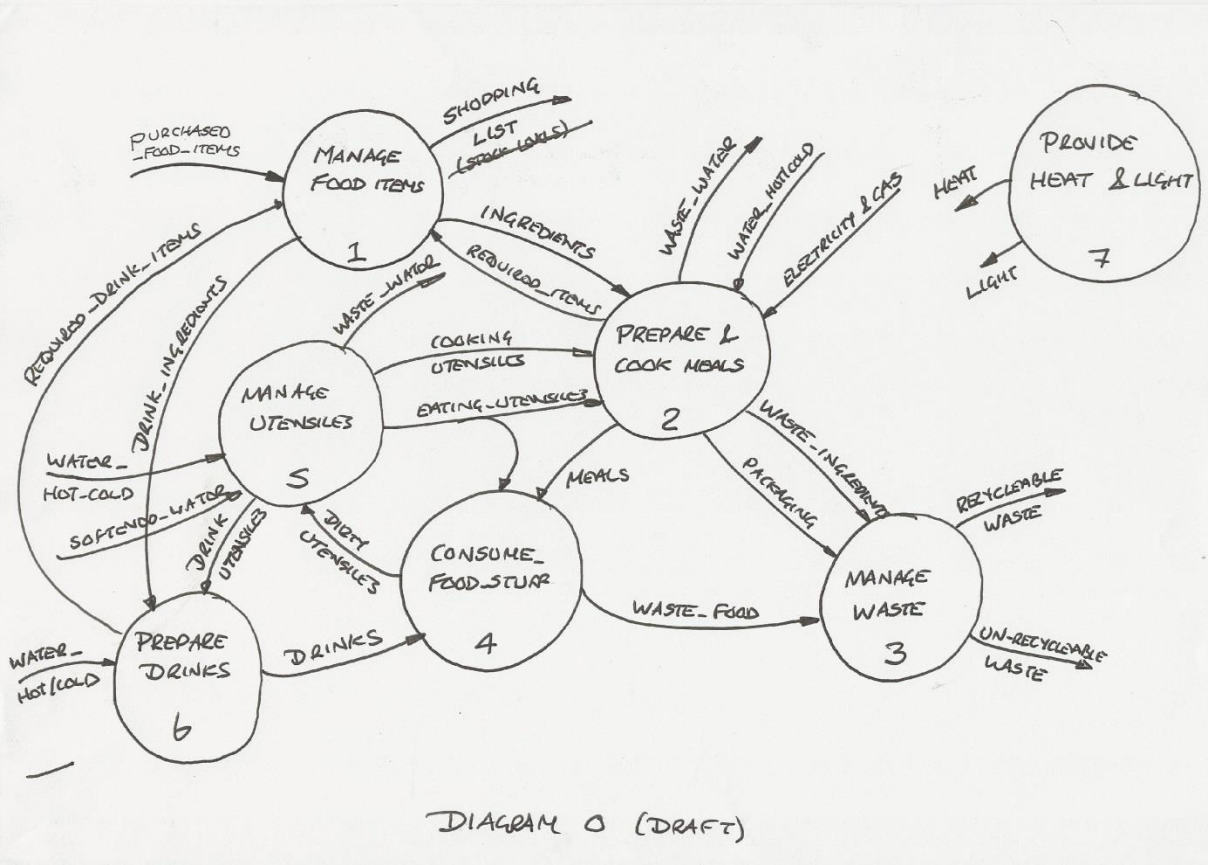


Figure 8: First Draft DIAGRAM 0 for the Domestic Kitchen

Drawing this diagram helped to clear up the issue raised on the draft CONTEXT DIAGRAM since it became clear that a better output of the MANAGE FOOD ITEMS function would be the SHOPPING LIST rather that the STOCK LEVELS. This was based initially on almost a intuitive feeling but confirmed by constructing a draft DIAGRAM 1 and simultaneously developing a Functional Specification for the MANAGE FOOD ITEMS. Technically, Functional Specification are only constructed for the lowest level of functionality – the so called primitive level – but on occasions it is highly useful to develop ones for higher level functionality. This situation is shown in Figure 9, which comprises a outline Functional Specification of the MANAGE FOOD ITEMS function and the corresponding lower level DIAGRAM 1.

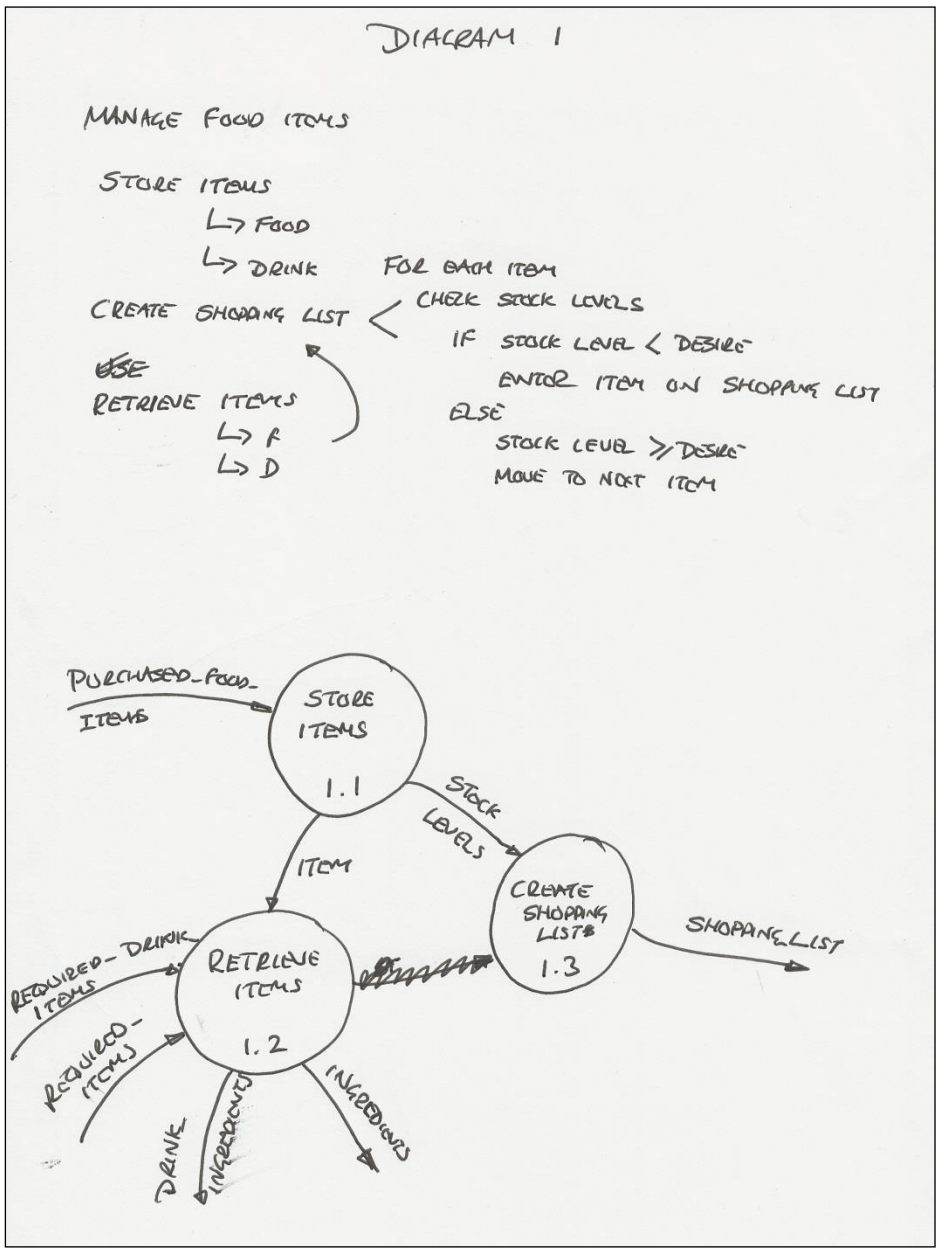


Figure 9: Outline Functional Specification for the MANAGE FOOD ITEMS function and associated DIAGRAM 1

Have explored the lower levels it is now possible to go back and modify the higher level Diagrams.

Another “fix” on the draft DIAGRAM 0 concerns function 7: PROVIDE HEAT & LIGHT. This particular function technically should output HEAT and LIGHT to all the other functions. However, all this would accomplish is to add complexity for no benefit. All models are wrong but some are useful[5] - the purpose of constructing functional models is to help clarify our thinking and adding pedantry detail does achieve this.

[5] This is quote from the great British statistician George Box

The Diagrams form an important part of the overall model but it is also important to compile the Flow Dictionary and generate the Functional Specifications. These should not be left but discussed and agreed as the diagramming proceeds. Quite simply because doing so will avoid later arguments over precisely what a function does and what is meant by a particular flow. Indeed, the whole purpose of functional modelling is for the team to arrive at a common understanding and collective view.