Functional Modeling with Data Flow Diagrams

Amasi Elbakush

5771668

Teaching Assistant : Daniel Alami

Utrecht University
Introduction

Data Flow Diagrams (DFDs) are a visual representation of the flow of data in a system. It is widely used in the analysis and design of systems in spectrum of disciplines. It was developed by many established researchers such as Edward Yourdon and Tom DeMacro (Yourdon, 2006). One of the DFD’s core step is decomposition of processes. After looking at the ‘big picture’ of the system, which is the context diagram, the decomposition process for the main activities in it starts in a top down approach. It starts with a process along with its relative data flows and further break it down into a simpler processes and so on until the entire diagram is simple enough to analyse and understand. According to Li and Chen (2009) it can be achieved in 7 steps that require many iterations depending on the size and complexity of the system at hand, and the gradual refinement of each activity until the DFD is well-representative of the system. Yourdon (2006) best defines Data flow diagram (DFD for short) as “a modeling tool that allows us to picture a system as a network of functional processes, connected to one another by ‘pipelines’ and ‘holding tanks’ of data”. DFDs illustrate functions and how data is exchanged in a system (Li & Chen, 2009). DFDs are commonly used to model systems, specifically operational systems, where functions are very complex and of high importance (Yourdon, 2006).

Some characteristics of DFDs include annotation, ability to provide a system’s activities and processes descriptions, analysis of a system in its design stages, and hierarchical decomposition of functions and processes (Li & Chen, 2009). DFD provides a set of symbols to illustrate the flow of data as well as a decomposition mechanism to explain the system at different levels of details. For the data flow analysis to be complete, a data dictionary explaining the different labels and necessary details as well as a process specification need to be included according to Li and Chen (2009).

As mentioned earlier, one of DFDs’ main characteristics is hierarchical decomposition; that activities/processes can be expanded into deeper activities/processes. More specifically, some activities/processes in the “parent” diagram can be decomposed into elaborate ones in the “child” diagrams (Li & Chen, 2009). A context diagram, level-0 diagram, and related level-n diagrams comprise the components of DFDs modelling set. The context diagram represents the scope of the system that demonstrate an overview of the different aspects of the system. Whereas the level-0 diagram represents the main processes, detailed data stores and flows in the system. The successive level-n diagrams represent an expansion of a single activity/process in the previous diagram or the parent diagram (Li & Chen, 2009). The expansion or decomposition step is iterated to a point where the entire system is modelled thoroughly (Durugbo et al., 2010). A tree-like pattern emerges with decomposition each time (Li & Chen, 2009).
**Procedure**

The procedure for making a DFD involves many iterations that gradually refine each process. The general steps to make DFDs are adopted from Li and Chen (2009) and they are as follows:

Step 1: Create a preliminary Context Diagram.
Step 2: Identify Use Cases, i.e., the ways in which users most commonly use the system.
Step 3: Create DFD fragments for each use case.
Step 4: Create a level-0 diagram from fragments.
Step 5: Decompose to level 1, 2, . . .
Step 6: Go to step (1) and revise as necessary.
Step 7: Validate the DFDs with users.

**Notation**

The four essential building blocks of DFDs are processes/activities, external entities, data flow and data store. First, an activity/process takes data flow as input and produces output. An activity/process can also be decomposed to sub-activities/processes for more information. The label for the activity needs to be a verb. Second, external entities represent start and end of external flow of data. They “provide connection to the system’s context” (Li & Chen, 2009). The label for external entities needs to be a noun.

The third part is data flow which represents the flow of information in the system. It transfers data between different parts of the system. It has direction, joints or splits, and its label needs to be a noun.

Lastly, data store represents the permanent store of information as well as database placeholder. The label of a data store needs to be a noun (Li & Chen, 2009).

There are two main notations for DFDs, the notation of Gane and Sarson and the extended notation of Ward and Mellor (Li & Chen, 2009). Figure 1 and 2 below demonstrates these two notations.

![Figure 1: DFD notation by Gane and Sarson](Source: Li & Chen, 2009)
The main use of DFDs is modelling functions of systems and the analysis of a system. Since a system can be very complex to understand, DFDs provide a tool to study and analyse each part of the system in more details (Ibrahim & Yen, 2011). DFDs help identify the functions a system uses, the interaction between these functions, as well as the flow of inputs and outputs. (Yourdon, 2006)

DFDs were first introduced in the 1970s by Yourdon and Constantine as a tool for analysing structured designs. In the late 1970s, DFDs were developed by Tom DeMarco (Durugbo et al., 2010).

**Example**[ to be re-created for final version]

To get a better idea of the application of DFDs in function modeling, an example of a food ordering System adopted from Li and Chen (2009) is discussed. The functionality of this system include: 1) A customer ordering food 2) Food order received 3) Food order sent to the kitchen 4) A receipt is sent to the customer 5) and a report is sent to the manager.

The first step to create a complete DFD is to create a context diagram of the system. In this step, the system boundaries are outlined as well as main data flows and interaction between external entities and the system. In this diagram there is a single main process/activity: Order Food. Customer, kitchen and Restaurant Manager represent the external entities.
The following steps are the identification of possible emergent use cases or processes from the previous diagram and combining their fragments into the Level-0 diagram. Those emergent use cases are updating inventory file, Goods sold file, and produce management report. The Level-0 diagram of the system is illustrated below as well as the flows and stores of data.

![Level-0 Diagram of the Food Ordering System](source: Li & Chen, 2009)

**Procedure Description**

To give an idea of the DFD technique, a generic meta-model is created through the application of method engineering approach. Method engineering is an engineering discipline created to guide the adaptation of methods in systems development. According to its original developer Sjaak Brinkkemper (1996) method engineering is “the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems” (p. 276). One of the highlights of this discipline is the development of what is known as Process-Deliverable Diagram (Weerd & Brinkkemper, 2008) for modeling methods and techniques. The main parts that represent a PDD are the meta-process model and the meta-data model and are based on UML diagrams. The meta-process model resembles the activities of the technique, whereas the meta-data model resemble the deliverable part of the PDD. The activity part is modeled on the left hand side of the PDD and the deliverables are modeled on the right hand side of the PDD. A PDD for creating a Data Flow Diagrams as proposed by Li and Chen (2009) is depicted in Figure 1 below. Both meta model sides have tables that further explain their contents. These are presented in the next section.
Figure 1. *Process Deliverable Diagram for creating Data flow diagram*

The PDD includes the 7 steps proposed by Li and Chen (2009) to create a DFD. These steps are modeled as activities that get executed concurrently and sequentially (the 6th step is modeled as the condition at the end of the process).
Activity Table

Activities contained in the PDD are illustrated in the table below. This table summarizes the activities performed to create a DFD. Each activity is supplemented with a corresponding Description of its purpose. Note that these activities correspond to the left side of the above PDD.

Table 1. Activity Table of the DFD technique

<table>
<thead>
<tr>
<th>Activity</th>
<th>sub activities</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create context diagram</td>
<td>● Identify system&lt;br&gt;● Identify core functions&lt;br&gt;● Draw CONTEXT DIAGRAM</td>
<td>This creates a basic system overview with the core system functions needed.</td>
</tr>
<tr>
<td>Identify use cases in the context diagram</td>
<td>● Define users&lt;br&gt;● Define use case&lt;br&gt;● Identify relations</td>
<td>Identifying the possible the ways that users might use the system. For example, order food, update inventory, check balance etc.</td>
</tr>
<tr>
<td>Create a DFD fragments (for each use case)</td>
<td></td>
<td>For each use case identified, create a detailed DFD that decomposes each USE CASE into smaller detailed ones.</td>
</tr>
<tr>
<td>Create Level 0 diagram</td>
<td></td>
<td>Collect the DFD FRAGMENT created for each USE CASE and combine them in one DFD called LEVEL-0 DIAGRAM</td>
</tr>
<tr>
<td>Decompose Level 0 into relative child diagrams</td>
<td></td>
<td>From LEVEL-0 DIAGRAM take any process that can be further broken down into simpler processes and create the next level-n diagram OR CHILD DIAGRAM (level-1,2,...)</td>
</tr>
<tr>
<td>Validate the DFDs with users</td>
<td></td>
<td>Check with the system users if this final DFD is an accurate representation of the system.</td>
</tr>
<tr>
<td>[Condition]: If not approved, go to Step 1 and revise as necessary</td>
<td></td>
<td>Reviewing the first diagram and see if any existent or emergent functionalities could be modified or added.</td>
</tr>
</tbody>
</table>

Concept Table

The deliverables from each activity performed to create a DFD are presented as concepts in the following table. Each concept is described further in the adjacent column. These deliverables represent the right hand side of the PDD. They are the result of each activity performed in order to create the DFD.

Table 2. Concept table of the technique DFD
<table>
<thead>
<tr>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFD</td>
<td>A diagram that represent data flows, stores, sinks as well as processes that are performed on the data. The flow of data between the node is represented by links connecting these nodes. (ISO/IEC/IEEE, 2010).</td>
</tr>
<tr>
<td>SYSTEM</td>
<td>System scope identified for creating the context diagram and the DFD.</td>
</tr>
<tr>
<td>CORE FUNCTION</td>
<td>Specifies the core functions of the system identified with the users using the system.</td>
</tr>
<tr>
<td>CONTEXT DIAGRAM</td>
<td>The CONTEXT DIAGRAM is the DFD representing the scope of a system. It shows the boundaries, external entities which interact with the system and the main flows of information between the system and the external entities (Li &amp; Chen, 2009).</td>
</tr>
<tr>
<td>USE CASE</td>
<td>A stepwise description of events or scenarios that may happen simultaneously (ISO/IEC/IEEE, 2010). According to Li and Chen (2009), a use case represents the ways in which users most commonly use the system.</td>
</tr>
<tr>
<td>USER</td>
<td>Users identified that are involved in the use cases.</td>
</tr>
<tr>
<td>RELATION</td>
<td>The relation between the different functions in a use case. For example, the 'extend' relationship.</td>
</tr>
<tr>
<td>DFD FRAGMENT</td>
<td>A DFD of the USE CASE of each activity in the CONTEXT DIAGRAM that have more details.</td>
</tr>
<tr>
<td>LEVEL-0 DIAGRAM</td>
<td>The LEVEL-0 DIAGRAM is a DFD that represents the major processes, data flows and stores in a system at a high level of detail (Li &amp; Chen, 2009). It is also regarded as the parent diagram.</td>
</tr>
<tr>
<td>CHILD DIAGRAM</td>
<td>A stepwise decomposition of LEVEL-0 yields LEVEL-1 DIAGRAM, LEVEL-2 DIAGRAM etc (Li &amp; Chen, 2009).</td>
</tr>
<tr>
<td>VALIDATION REPORT</td>
<td>A report from users of the system confirming the completeness and correctness of the final DFD.</td>
</tr>
</tbody>
</table>

**Related literature**[to be re-written for final version]

DFDs were first described in the 1970s in classical books such as “Structured Design” by Constantine, Steven and Myer (1974) and “Structured Design” by Yourdon and Constantine (1975). According to Yourdon (2006), DFDs were first used as a notation for analysing systems problems in the software engineering specialty. As a result, DFD notation had been “borrowed”
in many graph theory papers. It remained a useful and suitable method for modeling functions and analysing systems (Yourdon, 2006).

Although DFDs offer a suitable functional modeling method, Yourdon (2006) argues that they do not give specific details regarding the function components of the system that they model and that they need to be accompanied by dictionaries and specifications.

Kojarski and Lorenz (2006) emphasise that DFDs do not imply process ordering. The labels resemble the role of identifiers only. Thus, when the communication between different parts of the system occur is not implied in DFDs.

Avison and Fitzgerald (2003) mention the fact that data flow diagramming is a structured analysis technique provides the ability to analyse complex system from top to bottom. DFDs represent a useful systems analysis tool as they illustrate a clear overview of a system’s boundaries, data flow, and connections. Le Vie and Donald (2000) state that DFDs offer system designers help in visualizing their current system while in the preliminary stages. Such benefit will make it convenient for designers to make changes and meet requirements for the system. However, the authors state that DFDs are most useful with smaller system as they tend to be complicated to interpret with large systems (Le Vie & Donald, 2000). Le Vie and Donald (2000) also recommend using DFDs in object oriented architectures and that they can be beneficial when used with object oriented languages.
References


