Software Engineering - I

An Introduction to Software Construction Techniques for Industrial Strength Software

Chapter 4 – Process Models

Domain Models

During requirements analysis phase, different models are developed to express requirements of the system. Though it is difficult to draw a line between these models as they complement each other, they differ in the manner information is expressed in these models. Most of these models are pictorial and contain explanation to the diagrams. Some of these models are discussed in the following subsections.

Understanding the business domain

It must always be kept in mind that the first step in delivering a system is establishing what needs to be driven. That is, clear understanding of the problem domain is imperative in successful delivery of a software solution. A software developer has to develop an understanding of the business problem he is trying to solve. Unless he develops this understanding, it is really difficult, if not impossible, to develop the right solution. But at least if he collects both ends (sources, sinks) involved in different processes of the business system, the corresponding requirements will be complete and yield a better understanding of the problem domain. A software engineer works on domains that may not correspond to his field of specialization (computer science, software engineering). He may be involved in the development of an embedded application that automates the control pad of a microwave machine or a decision support application for a stock exchange broker. As the underlying systems for which these software applications are being developed are not software systems, the software engineer cannot be expected to know about these domains. So, how should he get all the required knowledge about these systems? As without acquiring this knowledge, he may not be able to write down complete and unambiguous requirements which are acceptable to users as well.
An important difference between software and another engineering discipline is that the software engineer has to work on problems that do not directly relate to software engineering. Whereas, an electrical engineer will work on electrical domain problems, a civil engineer will work on civil engineering problems and so on. So, software engineer has to learn user vocabulary and terms which they use in their routine operations. To overcome this problem, a number of domain gathering techniques are used. These techniques help in extracting requirements from systems which are not known to a software engineer. Using these techniques the requirements gathering and validation process becomes convenient and manageable for a software engineer.

The following subsections discuss some of these techniques.

Logical System Models

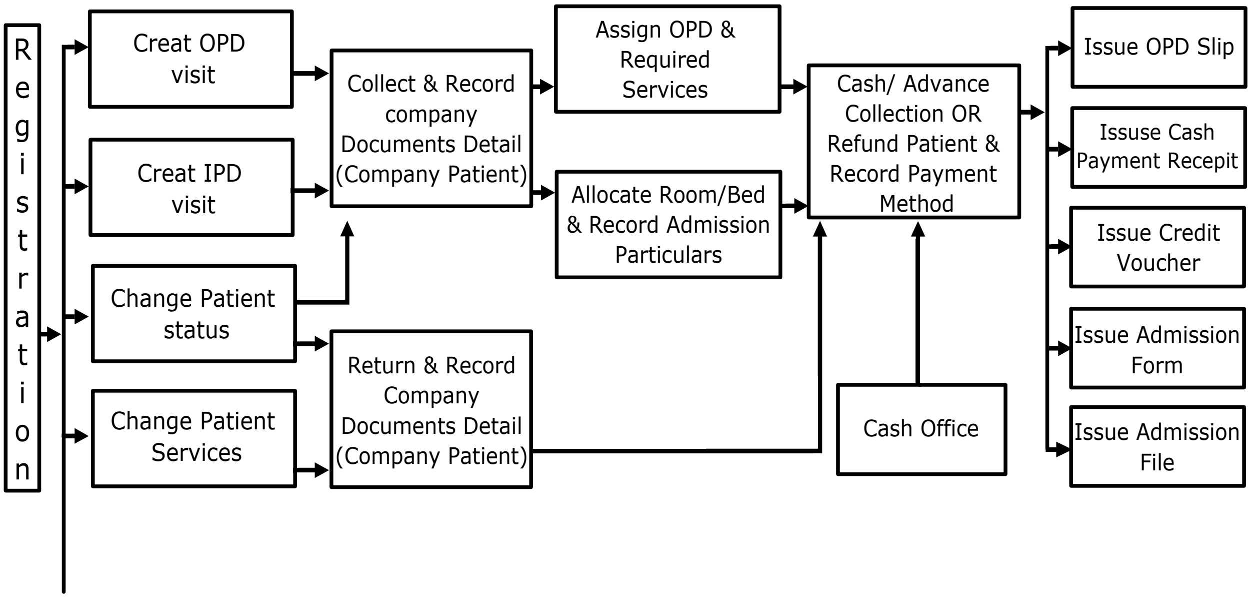
System models are techniques used to understand user needs and software engineer use these techniques in order to understand business domain. Software engineers develop diagrams to model different business processes. System models include the following

- User business processes
- User activities for conducting the business processes

- Processes that need to be automated
- Processes which are not to be automated

Business process model

The first model that we will look at is called the process model. This model provides a high-level pictorial view of the business process. This model can be used as a starting point in giving the basic orientation to the reader of the document. Following is an example of a hospital registration system which deals with two types of patients.


Registration
Creat OPD visit
Creat IPD visit
Change Patient status
Change Patient Services
Collect & Record company Documents Detail (Company Patient)
Return & Record Company Documents Detail (Company Patient)
Assign OPD & Required Services
Allocate Room/Bed & Record Admission Particulars
Cash/ Advance Collection OR Refund Patient & Record Payment Method
Cash Office
Issue OPD Slip
Issuse Cash Payment Recepit
Issue Credit Voucher
Issue Admission Form
Issue Admission File

As opposed to flow charts, there are parallel activities in this diagram which are further elaborated by specifying their major activities. The process described in this diagram is as follows

- A patient may come to visit In Patient Department (IPD) or output patient department (OPD)
- System determines if he is a company patient or a private patient.
- For a company patient, system verifies him.
- For an OPD patient, system will issue a chit to the patient and inform him about his number and the consultant to whom he has to consult and he will have to wait for his turn.
- After verifying an IPD patient, system will create a visit and allocate him a room or a bed etc. If system cannot allocate this, then it will inform the patient. Otherwise the patient is checked in and his information is maintained in the system.
- System displays information about the expenses of the required service to the patient so that he is informed of his expected expenditure.
- Some advance payment is also received against the required service and this amount is adjusted in the final settlement.

- All this information is supplied to cash office that eventually deals with payments, etc.
- Upon receiving the cash, for OPD patient, a chit will be issued. For IPD patient, an admission form will be filled and this information will be maintained in the system. A receipt will be issued to the patient.
- For credit transaction, corresponding voucher will be prepared.
- So the model depicts process before the start of the treatment.
- A patient may ask to change his service on event of an unsatisfied response from the hospital staff or any other reason. System may cancel his record and pay his amount back.
- Similarly, a doctor may ask a patient to change his status from OPD to IPD.

In a business process diagram, following points are important and should be noted

- It does not describe the automated system
- It only reflects the existing process of the user to help software engineer/analyst in understanding business domain.
- It may contain information on processes that need not be automated.

State Transition Diagrams

State transition diagrams (STDs) are another technique to document domain knowledge. In many cases, information flows from one place to the other and at each place certain action is taken on that piece of information before it moves to the next place. A file in an office is a typical office is example of such system. In this case, different people make comments and add information to that file and it moves from one table to the other> this movement is controlled by a pre-defined set of rules which define under what condition the file moves from place A to place B and so on. We can easily document these set of rules with the help of state transition diagrams.

Following is an example of a use of STD to document the life cycle of a trouble ticket (this example has been taken from ITU-X.790 document).

A Trouble report and its life cycle – and introduction

From time to time all systems, including communications networks, develop problems or malfunctions referred to in this Recommendation as “troubles”. A “trouble” in a communications network is a problem that has an adverse effect on the quality of service perceived by network users. When a trouble is detected, possibly as a result of an alarm report, a trouble report may be entered by a user or the system may raise a report automatically. Management of that trouble report is necessary to ensure that it receives attention and that the trouble is cleared to restore the service to its previous level of capability.

At the time of a trouble, a network may have been inter-working with another network to provide a service, and the problem or malfunction may be due to the other network. Therefore it may be necessary to exchange trouble management information between management systems across interfaces which may be client to service provider or service provider to service provider interfaces and may represent inter-jurisdictional as well as intra-jurisdictional boundaries. In addition to exchanging information on trouble that has already been detected, advance information on service inaccessibility may also need to be exchanged. Thus, a service provider may need to inform a customer of future service inaccessibility (because of planned maintenance, for example).

Trouble report states and status

Referring to the State transition diagram in Figure 2, a trouble report may go through any of six states during its life cycle. In addition, a Trouble Status attribute is defined which qualifies the state (finer granularity) e.g. cleared awaiting customer verification. The time at which the status attribute change is also captured in the trouble report

.

Following is a description of states of a trouble report.

Queued

A trouble report is in a queued state when it has been instantiated but the trouble resolution process has not yet been initiated. A trouble report which is in the queued state may be cancelled by the manager. The agent on receiving such a request will attempt to close the trouble report.

Open/active
The trouble report becomes "open/active" when appropriate actions to resolve the trouble are initiated.
An "open/active" trouble report may be "referred" to another Hand-off Person, or "transferred" to another Responsible Person for further processing. The state however remains unchanged as "open/active". A trouble report in the open/active state may be cancelled by the manager. The agent on receiving such a request will attempt to close the trouble report.

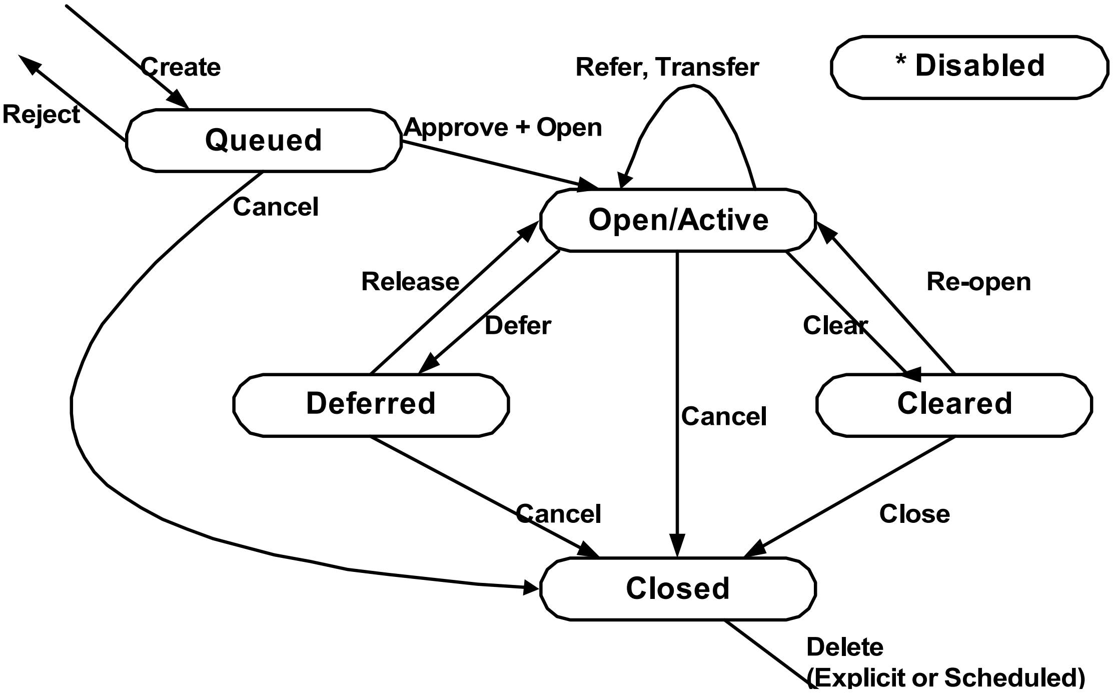
Deferred
This state indicates that corrective action to resolve the trouble has been postponed. This can occur when the faulty resource is inaccessible for a period and repair activity cannot proceed. A deferred Telecommunications Trouble Report may become "open/active" again, or move directly to the "closed" state if it is cancelled for some reason. A trouble report in the deferred state may be cancelled by the manager. The agent on receiving such a request will attempt to close the trouble report.

Cleared
A trouble report is moved by the agent to the "cleared" state when it determines that the trouble has been resolved. If the manager needs to verify that the trouble has been resolved, verification may optionally be awaited by the agent prior to closure of the trouble report.

Closed
This state indicates that the trouble resolution process is complete. Upon closure, the trouble report attributes are captured in a historical event generated at trouble report closure which may then be stored in a log of trouble history records, for future reference. The trouble report may then be eliminated at the agent's convenience. However, the agent may be required to maintain such records for a period of time as per business agreements.

Disabled
A "disabled" value is exhibited when a trouble report's information cannot be updated due to local conditions. In the "disabled" condition only read operations can be performed.

The following figure shows the STD for a trouble ticket. This diagram depicts the movement of a trouble ticket from one state to the other, thus making it easy to understand.


Create
Reject
Queued
Approve + Open
Cancel
Refer, Transfer
* Disabled
Open/Active
Release
Defer
Re-open
Clear
Deferred
Cancel
Cleared
Cancel
Close
Closed
Delete
(Explicit or Scheduled)

Arranging information in tabular form

Sometimes it is better and more convenient to arrange information in a tabular form. This makes it easier for the reader to understand and comprehend the information and hence designing, coding, and testing become less challenging. As an example, let us look at the following definitions used for identifying different data functions in the function point analysis taken from International Function Point User's Group (IFPUG) Counting Practices Manual (CPM 4.1).

External Inputs

An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.

External Outputs

An external output (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other than, or in addition to, the retrieval of data or control information. The processing logic must contain at least one mathematical formula or calculation, or create derived data. An external output may also maintain one or more ILFs and/or alter the behavior of the system.

External Inquiry

An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.

It is difficult to understand these definitions and one has to read them a number of times to understand what is the difference between EI, EO, and EQ and in which case a function would be classified as EI, EO, or EQ.

Now the same information is presented in the tabular form as follows:

Function	Transactional Function Types		
	EI	EO	EQ
Alter the behaviour of the system	PI	M	N/A
Maintain one or more ILFs	PI	M	N/A
Present information to the user	M	PI	PI

PI – Primary intent; M – may be; N/A – not allowed.

This table simply says that a function can alter the behaviour of the system, it can maintain one or more ILFs, and/or it can present information to the user. The next step is to determine whether it is EI, EO, or EQ. For that we have to determine what is the primary intent (PI) of the function and in addition to this primary intent, what else does it do. Identification of EQ is simple - in this case the only thing a function does is present information to the user, which is also it's primary intent. If it alters the behaviour of the system or maintains and ILF then it can either be an EI or and EO but not an EQ. On the other hand if the primary intent of the function is to present information to the user but at the same time it also performs any of the first two operations, it is an EO. Finally, if the primary intent of the function is either to alter the behaviour of the system of maintain one or more ILFs, then it is an EI.

Hence by putting and organizing the information in the form of a table, we have not only made it simple to understand the definition but also given an holistic picture which was not easily visible otherwise.

Let us look at another example. This time the information is taken from the Income Tax Ordinance of Pakistan 2001. Consider the following statement that describes the income tax rates applicable to people with different brackets:

If the taxable income is less than Rs. 60,000, there will be no income tax. If the income exceeds Rs. 60,000 but is less than Rs. 150,000 then income tax will be charged at the rate of 7.5% for income exceeding Rs. 60,000. If the income exceeds Rs. 150,000 but does not exceed Rs. 300,000 then the income tax will be computed at 12.5% of the amount exceeding Rs. 150,000 plus Rs. 6,750. If the income exceeds Rs. 300,000 but does not exceed Rs. 400,000 then the income tax will be computed at 20% of the amount exceeding Rs. 300,000 plus Rs. 25,500. If the income exceeds Rs. 400,000 by does not exceed Rs. 700,000 then the income tax will be computed at 25% of the amount exceeding Rs. 400,000 plus Rs. 45,500. If the income exceeds Rs. 700,000 then the income tax will be computed at 35% of the amount exceeding Rs. 700,000 plus Rs. 120,500.

The same information can be organized in the form of a table, making it more readable and easier to use.

Income	Tax
Less than Rs. 60,000	0%
Between Rs. 60,000 and Rs. 150,000	7.5% of (Income - 60,000)
Between Rs. 150,000 and Rs. 300,000	12.5% of (Income - 150,000) + 6,750
Between Rs. 300,000 and Rs. 400,000	20% of (Income - 300,000) + 25,500
Between Rs. 400,000 and Rs. 700,000	25% of (Income - 400,000) + 45,500
Greater than Rs. 700,000	35% of (Income - 700,000) +

	120,500

Once the information has been organized in the tabular form, in many cases it can be simply stored and mapped onto an array or a database table and the programming of this kind of a rule is simply reduced to a table or dictionary lookup. This reduces the complexity of the domain and hence reduces the over all effort for designing, coding, testing, and maintaining the system.

Data Flow Model

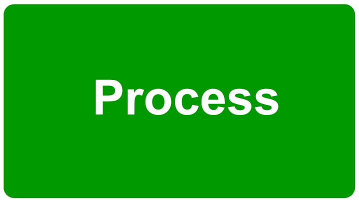
- Captures the flow of data in a system.
- It helps in developing an understanding of system's functionality.
- What are the different sources of data, what different transformations take place on data and what are final outputs generated by these transformations.
- It describes data origination, transformations and consumption in a system.
- Information is organized and disseminated at different levels of abstraction. Thus this technique becomes a conduit for top down system analysis and requirements modeling.

The Notation

There are several notations of the data flow diagrams. In the following, four different shapes are explained.

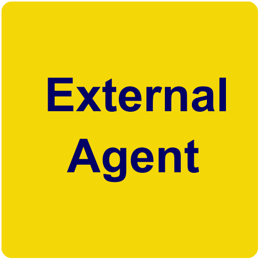
Process

- What are different processes or work to be done in the system.
- Transforms of data.



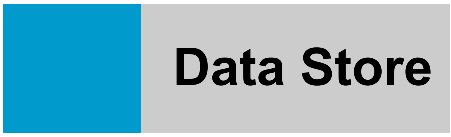
External Agent

External systems which are outside the boundary of this system. These are represented using the squares



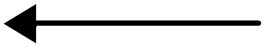
Data Store

- Where data is being stored for later retrieval.
- Provides input to the process
- Outputs of the processes may be going into these data stores.



Data Flow

- Where the data is flowing.
- Represents the movement of the data in a data flow diagram.



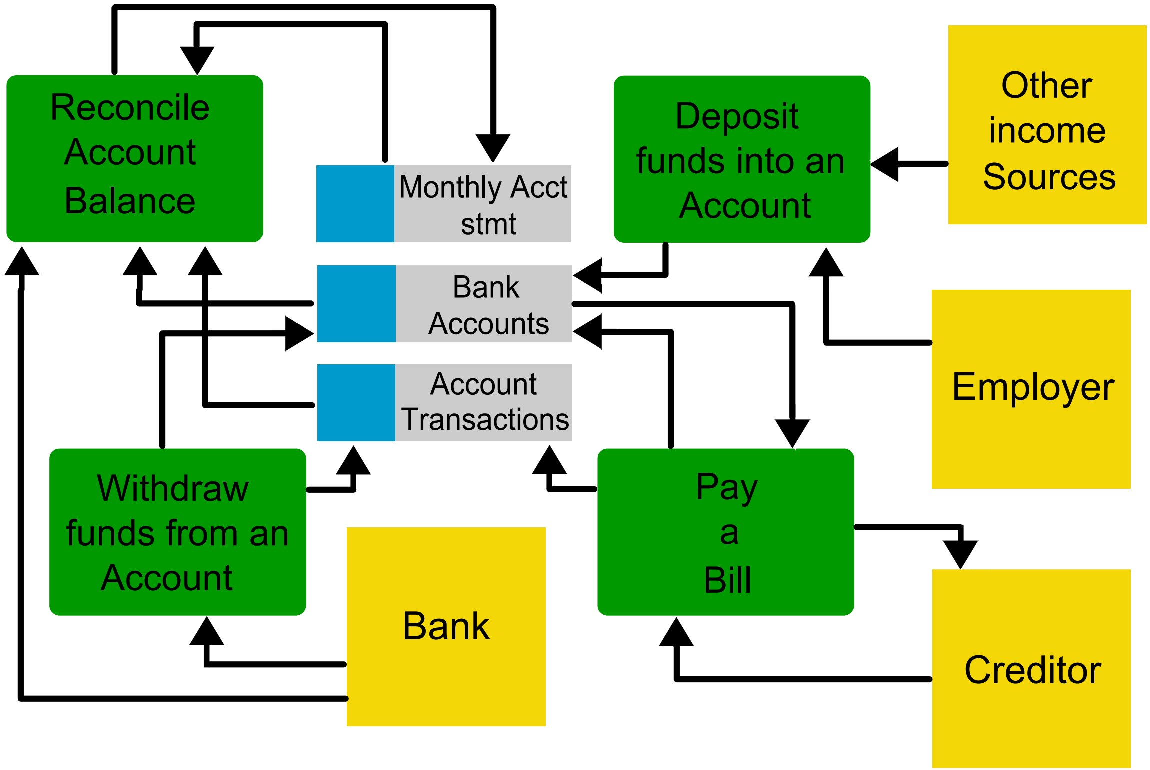
DFD versus Flow Charts

Flow charts are usually used to describe flow of control in a system. It describes control flow in an algorithm. Flow charts are quite detailed. Whereas DFD does not captures control flow information, it just shows the flow of the data in a system. Flow charts show the sequential activities of an algorithm. So, decisions are made, loops or iterations are described. On the other hand, DFD does not show the sequential activities. It just displays the business flow (without sequence among activities). As if you visit an organization, business activities are being performed in parallel. Therefore, DFD does not contain control or sequential activities just data transition is captured.

DFD	**Flow Chart**
• Processes on a data flow can operate in parallel. • Looping and branching are typically not shown. • Each process path may have a very different timing.	• Processes on flowcharts are sequential. • Show the sequence of steps as an algorithm and hence looping and branching are part of flowcharts.

Data Flow Model – Bank Account Management System

In the following, we are presenting a data flow model that describes an accounts management system for a bank. This data flow diagram consists of the following entities



Processes

1. Reconcile account balance
2. Deposit funds into an account
3. Pay a bill
4. Withdraw funds from an account

External agents

1. Bank
2. Creditor
3. Employer
4. Other income sources

Data stores

1. Monthly Account statement
2. Bank accounts
3. Account transactions

Description:

First we shall discuss ‘withdraw funds from an account’ process. In this process, information about the accounts and account transactions is retrieved (from the data stores) and bank releases the funds. After this, it sends this information to ‘reconcile account balance’ process which prepares a monthly account statement. In this statement, information regarding bank accounts and account transactions are described. Next is the ‘pay a bill’ process through which a creditor pays his dues and the corresponding accounts are updated against the cash transaction. A receipt is issued back to the creditor. The fourth process is ‘deposits funds in an account’ in which an employer deposits salaries of his employees and the salary information is deposited in the corresponding bank accounts of the employees. Similarly, income received through other income sources is also received and deposited in the corresponding bank accounts.

Data Flow Modeling

When data flow modeling is used to model a system’s functionality, following points need to be remembered

- Data flow model captures the transformation of data between processes/functions of a system. It does not represent the control flow information that is occurring in a system to invoke certain functionality.
- A number of parallel activities are shown in this diagram where no specific sequence among these activities is depicted
- All the previous models that we studied like business process models, state transition diagrams, are used to capture business domain irrespective of their automation.
- However, in data flow models, we represent only those processes which we need to automate as they involve certain computation, processing or transformation of data that can be best implemented using an automated system.
- For example, we may consider a mail desk in an office that receives mail and just forwards it to their respective addressees. In this example, as the mail desk does not process the mail, just forwards it, therefore it does not include any process that need to be automated. Hence, we shall not use data flow diagrams to model this process.
- In nutshell, processes that just move or transfer data (do not perform any processing on that data), should not be described using data flow models.
- Taking the same example, if we modify the scenario such that a mail desk clerk receives the mail, notes it down into a register and then delivers it to their respective addressees then a processing has got involved in this scenario. At least one process is there that can be automated. That is, the recording of mail information into the register. Now we can use a data flow model in which we shall use a data transformation that captures the detail of recording mail information into a register (or a data store). Thus with this addition, it makes sense to use data flow model to capture the details of this process.

Typical Processes

Now we shall discuss processes which are typically modeled using data flow diagrams. These processes transform data in one or the other way but these are found in almost all the automated systems. Following are the examples

- Processes that take inputs and perform certain computations. For example, CalculateCommission is a process that takes a few inputs like transaction amount, transaction type, etc and calculates the commission on the deal.
- Processes which are involved in some sort of decision-making. For example, in a point of sales application a process may be invoked that determines the availability of a product by evaluating existing stocks in the inventory.
- Processes that alter information or apply a filter on data in a database.
  For example , an organization is maintaining an issue log of the issues or complaints that their clients report. Now if they want to see issues which are outstanding for more then a weeks time then a filter would have to be applied to sort out all the issues with Pending status and whose initiation date is a week old.
- Processes that sort data and present the results to users. For example, we pass an array of arbitrary numbers to a QuickSort program and it returns an array that contains the sorted numbers.
- Processes that trigger some other function/process
  For example, monthly billing that a utility company like WAPDA, PTCL generates. This is a trigger that invokes the billing application every month and it prepares and prints all the consumer bills.
- Actions performed on the stored data. These are called CRUD operations and described in the next subsection

CRUD Operations

These are four operations as describes below

- **Create**: creates data and stores it.
- **Read**: retrieves the stored data for viewing.
- **Update**: makes changes in an stored data.
- **Delete**: deletes an already stored data permanently.

Adding Levels of Abstraction to Data Flow Modeling

As we have already described that in data flow modeling only those processes can be expressed that perform certain processing or transformation of information. Now the question arises how far these processes need to be expressed? As a single process like CalculateCommission as described in the above section, can be described in sufficient detail such that all of its minute activities can be captured in the data flow diagram. However, if we start adding each bit of system functionality in a single data flow diagram, it would become an enormously large diagram to be drawn on a single piece of paper. Moreover, requirement analysis is an ongoing activity in which knowledge expands as you dig out details of processes. Therefore, it may not be possible for an analyst to know each bit of all the processes of the system from the very beginning. Keeping the complexity of systems in view, data flow modeling technique has suggested disseminating information of a system in more then just one levels of abstraction. What are these levels, please see below for a discussion

Context Level Data Flow Diagram

In a top-down system analysis, an analyst is required to develop high level view of the system at first. In data flow modeling, this high-level view is the Context level data flow diagram. In this diagram, system's context is clarified such that all the external agents or entities with which the system interacts are captured. It captures the details of what information flows between the system and these external entities, and what outputs are generated against inputs from these external agents and so on. So, the analyst probes out all the external agents that may involve persons, organizations or other systems who directly interacts with this system and their specific involvement in the system. At this level, systems internal details are not exposed, as we want to see system behavior as a black box.

Detailed Data Flow diagrams

Once context of a system has been captured using context level diagram, the analyst would expand his activities and start digging out system's internal details. Therefore, the same context level diagram is further expanded to include all major processes of the system that make up system functionality. So, instead of portraying system as a black box entity, the analyst would add processes that deal with the external agents and produces certain outputs. This is level one of a data flow model.

In level two of data flow model, instead of refining the previous levels further, we take one process from the level one diagram and expands it in a level two diagram. Hence, a level one diagram that depict the whole system, may be expanded to more then one level two diagrams each of which describes exactly one process in detail which were listed in level one diagram as simply an oval (process or transform).

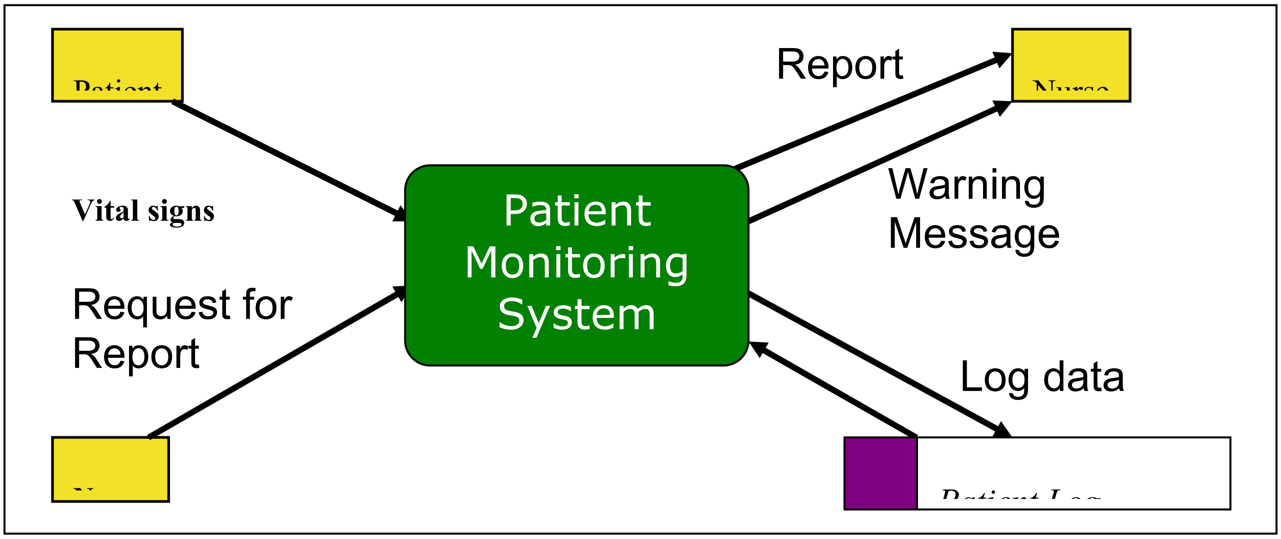
This process may continue to any level of details as the analyst can conveniently captures. Where diagram at a specific level is a refinement of one of the processes listed in a previous level. By adding levels of abstraction to a data flow diagram, it becomes natural for a software engineer or a requirement analyst to readily express his knowledge about the system in an appropriate level of data flow model that corresponds only to a specific set of functionality.

It should be noted here that the number of external agents and their inputs to the system and the outputs that the system would return to them, should remain the same throughout different levels of a data flow model. It should be considered a mistake if context level diagram contains three external agents, which are providing two inputs each, and getting one output in return but at level one, we add one more external agent or input or the outputs. This would make level one model inconsistent with the context level diagram. This is true for any level of data flow model. For instance, at level two the number of external agents, inputs and outputs shown in (all of level two) diagrams should match exactly with the external agents, inputs and outputs shown in level one diagram. Therefore, disseminating information at an appropriate level of abstraction with the additional check of inter-level consistency makes data flow modeling a very powerful domain-modeling tool. After this discussion, we shall give the reader an example in which a system is modeled using data flow modeling technique where three levels of abstraction have been developed.

Patient Monitoring System – A Data Flow Modeling Example

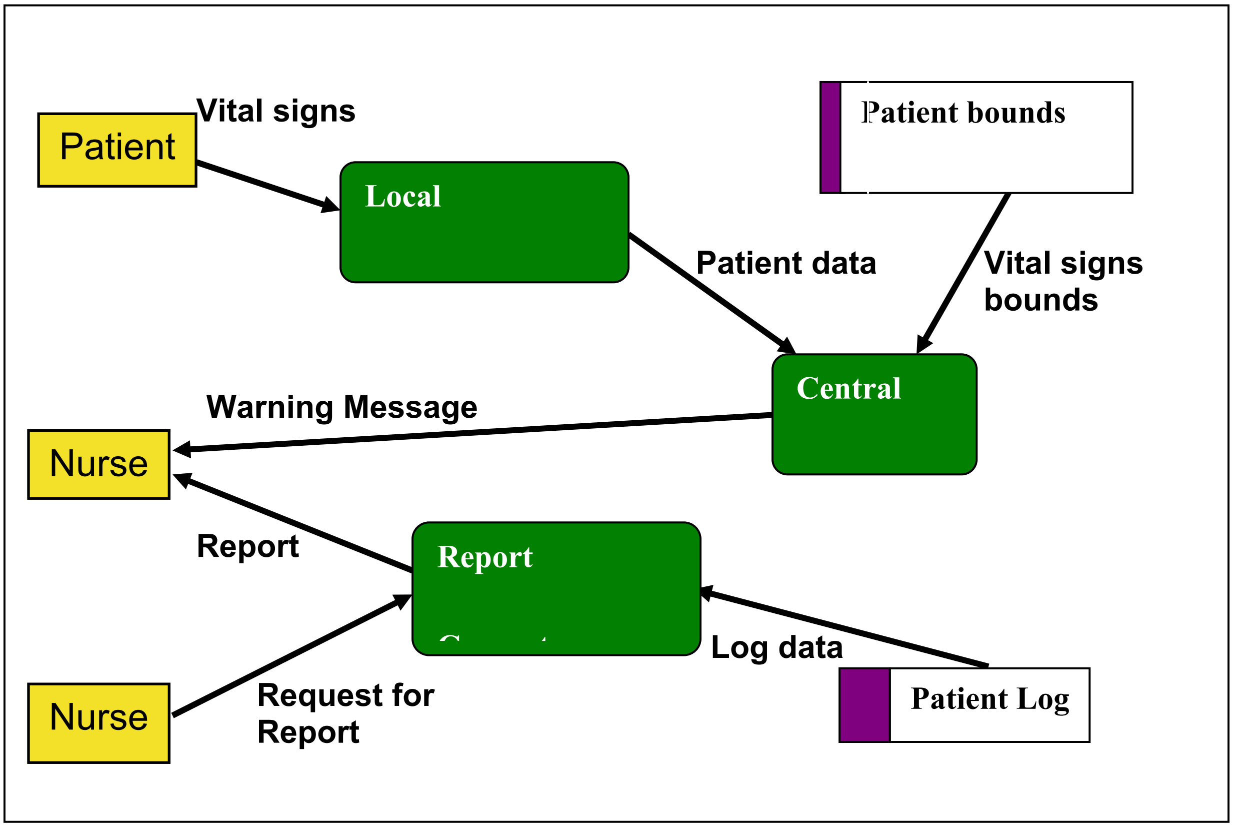
Context Diagram

Following is the 0-level or the context level data flow diagram of the Patient Monitoring System. In this data flow diagram, three external entities (users) are interacting with the centralized system. Point to note here is that in this context level diagram, only one process or transform takes place that is the Patient Monitoring System itself. A patient's vital signs are transmitted to this system which may invoke a warning message to the nurse if these signs fall into the critical range. Nurse may request for a report, which the patient monitoring system retrieves from the patient log, and return it to the nurse again. In this manner the 0-level data flow diagram describes the context of this system.



In order to see detail processes involved in Patient Monitoring System, a level 1 data flow diagram will have to be made. In the following, we are providing level 1 data flow diagram which is a refinement of the level-0 data flow diagram.

Patient Monitoring System – Level 1 Data Flow Diagram

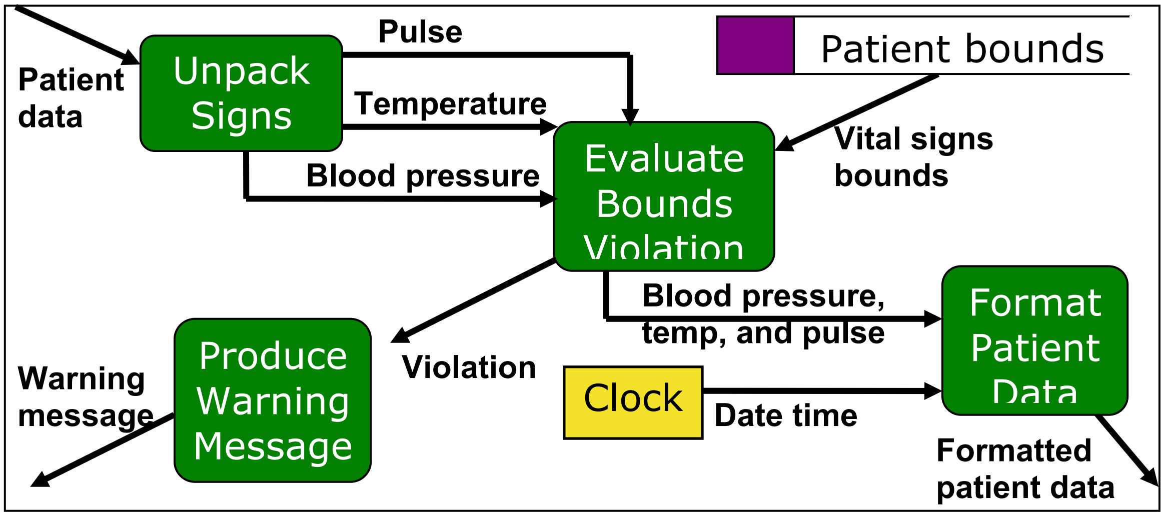


Level 1 data flow diagram is the refinement of the context (0-level) data flow diagram. All the external entities are the same (Nurse, and Patient), however, the process of ‘Patient Monitoring System’ is further elaborated by the three processes Local Monitoring, Report Generator, and Central Monitoring. The Local Monitoring process transforms vital signs that it receives from Patient entity into Patient data and passes this information to Central Monitoring process. Central Monitoring process retrieves vital signs bounds and compares Patient data and it may generate Warning message if the Patient data goes out of normal Vital signs bounds. A nurse may request for a report, in response the Report Generator process retrieves Log data from Patient Log, generates the report and displays it back to the nurse.

It should be noted here that this level 1 diagram is a further refinement of level 0 diagram such that the underlying system is the same but processes which were hidden in level 0 are represented in this diagram.

A further refinement of this model is also possible if we expand any of these three processes to capture further details. For example, the Local Monitoring process may further be expanded to capture detailed activities involved in the monitoring process. Following is level 2 diagram of Central Monitoring process.

Central Monitoring System – Level 2 Data Flow Diagram

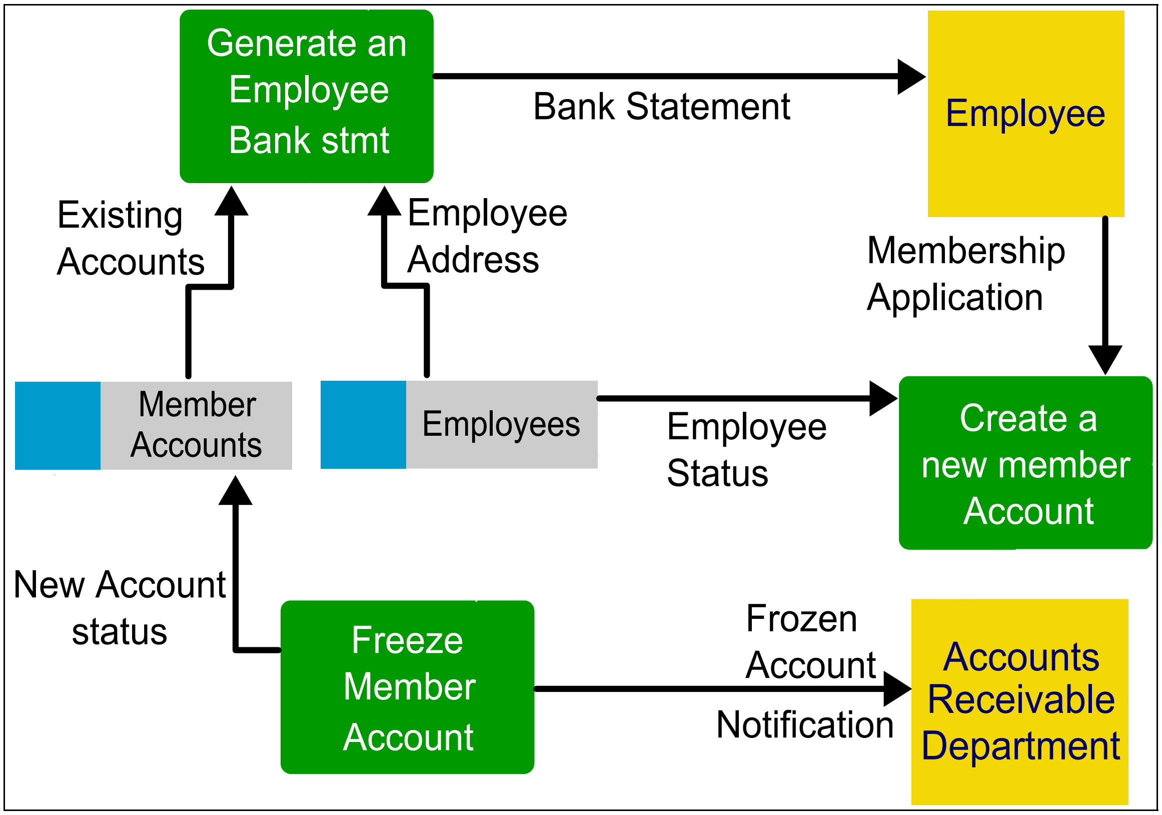


In the above level 2 data flow diagram, Patient's data is sent to the Unpack Signs process which unpacks it and send Pulse, Temperature, and Blood pressure to the Evaluate Bounds Violation process. This process retrieves Vital signs bounds information, compares it with unpacked patient data and sends a violation sign to the Produce Warning Message process upon an out of bound result of the comparison. The patient data is sent to Format Patient Data process that generates the formatted patient data to be maintained against patient profile. In this manner we elaborated the patient monitoring system up to three levels describing different details at each level. In a similar manner, other two processes in level 1 DFD could also be expanded in their respective level two diagrams in order to describe their functionality in more detail.

By going through this example, the reader would have learnt how data flow modeling technique helps in understanding domain of a system at different levels of abstractions. In the following sub-section, we shall describe common mistakes that the people do while preparing data flow diagrams.

Common Mistakes in Data Flow Diagrams

In the following data flow diagram, an accounting system has been described. Three processes are given Generate an Employee Bank Statement, Create a New Member Account, and Freeze Member Account. There are two external entities shown in this diagram Employee and Accounts Receivable Department. The three processes described in this diagram have associated problems. Can you guess these problems?
If you look at the arrows going inside each of these processes and coming out of them, you will observe some peculiarity.
In fact, here we can apply the source and sink analysis that we studied in the last lectures. What does the source and sink analysis suggest? It suggests that in order to check completeness of a requirement, evaluate the sources as well as the sinks of the requirements. Applying this knowledge in this case, we observe the following mistakes



- There is no input for the process Freeze Member Account
- In a similar manner, the process Create a New Member Account does not produce any output.
- Similarly, Generate Employee Bank Statement process is having two inputs and an output but the question really is, do these inputs correspond to the output?

The Freeze Member Account process that does not have any input is an example of a requirement whose source is not known. Similarly, the Create a New Member Account process that does not produce any output is an example of a requirement whose sink has not been specified. Lastly, the Generate an Employee Bank Account process though have two inputs and produces an output but in order to generate a bank statement, all that is needed is an account number and the time period for which the statement is required. If we analyze the inputs given to this process, we can observe that both of these inputs cannot help in generating the account statement. Therefore, these inputs are irrelevant to the output being generated by this process.

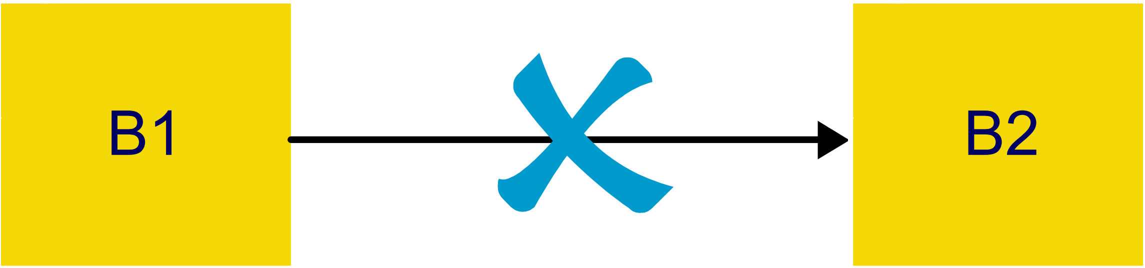
In the above mentioned example, it is evident that by applying the source and sink analysis we determined all the missing inputs and outputs to the processes of this diagram.

In the following subsection, we shall describe actions which are not only mistakes but illegal too for the data flow diagrams.

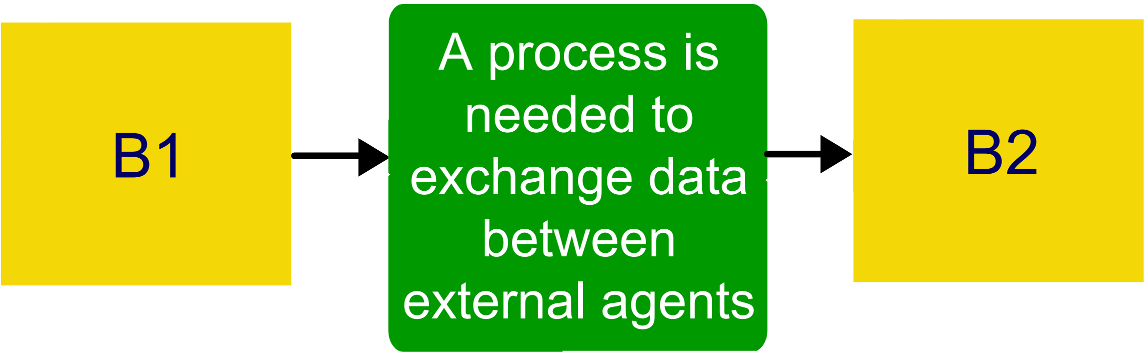
Illegal Data Flows

Directly Communicating External Agents

Following diagram depicts a scenario in which one external entity is directly communicating with another external entity. This form of communication is illegal to be shown in a data flow diagram.

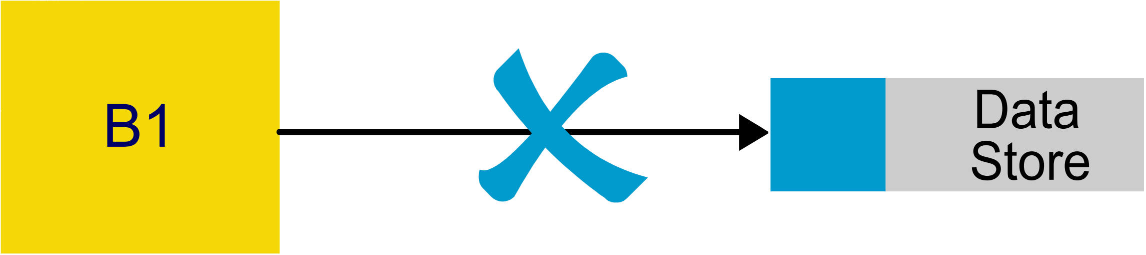


There must be an intermediate process which should transform data received from one external entity and then send the transformed data to the other external entity. As we have already described that data flow diagrams should be used to depict processes that transform or process data. Simple data movement from one entity to another should not be described using data flow diagrams.

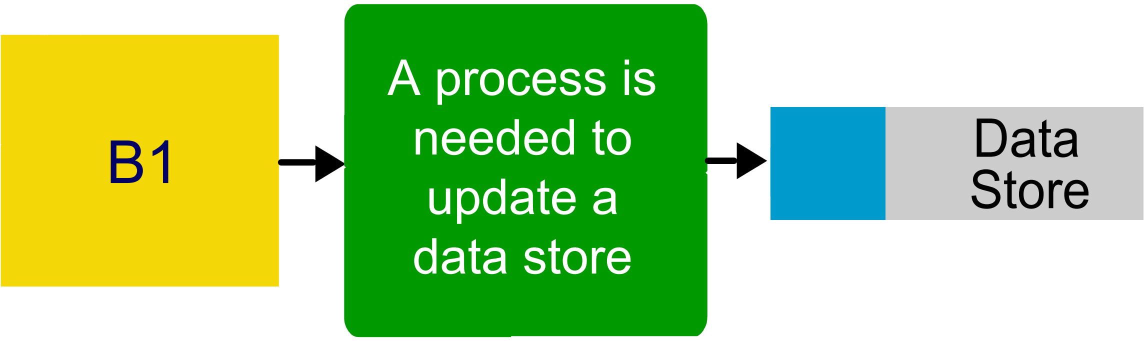

B1
A process is needed to exchange data between external agents
B2

External Agent updating information in a Data Store

As we explained in the above case, a transform/process is needed between communicating entities. This is true even for an External Entity that wants to store/update some information directly in a data store, a transformation would be required.

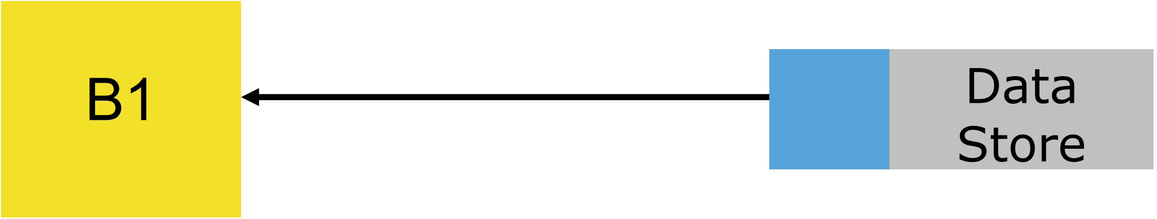


Therefore, a process should be inserted between the interacting entities (external agent, data store) that should store information received from the external agent after processing it.

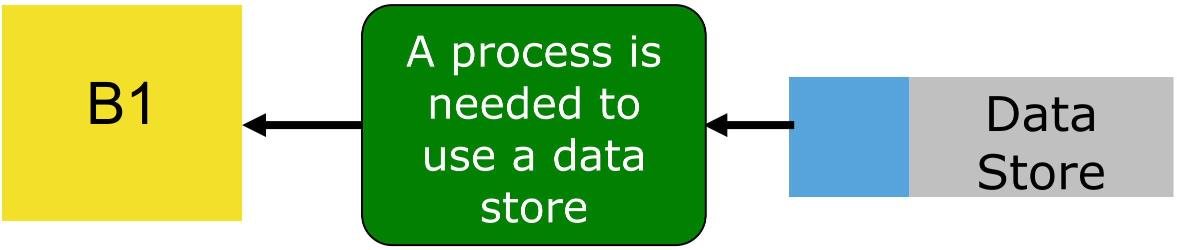


External Agent accessing information from a Data Store

Similarly, an external agent accessing information from a data store directly is also illegal.

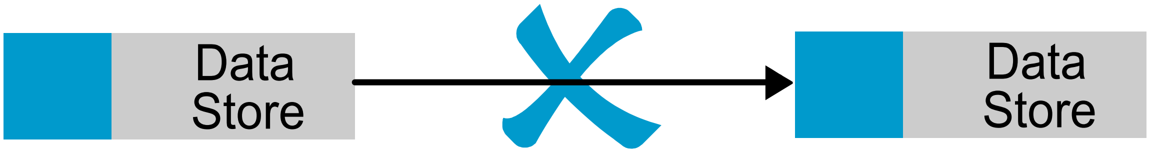


Again a data transform/process if needed in this communication. It should be able to retrieve information from the data store and then pass it on to the external agent.

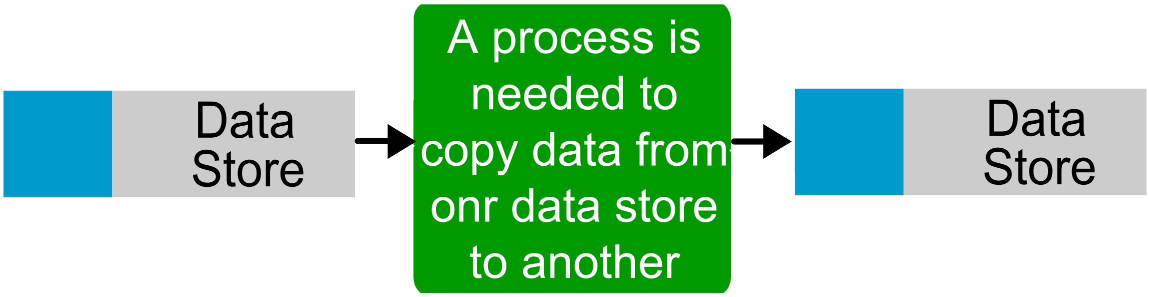


Copying data to a data store

In the following diagram, a data store is shown copying data directly to another data store. This is again illegal as there is not any intermediate process/data transform mentioned.



So, the correct method is again to use a data transform/process between the two data stores. It should retrieve data from one data store and after transforming that data, store it into another data store.


Data Store
A process is needed to copy data from onr data store to another
Data Store