Apply the Principles of Designing Computer System Inputs and Outputs

SAQA Unit Standard ID: 115365
NQF Level: 5
Credits: 7
Notional Hours: 70

Version: 1.2

Revision Date: 15 November 2019

Contents

What are Inputs and Outputs?

Inputs and outputs are possibly the most essential ingredients in a computer system. The word "compute" essentially means to take inputs and respond with an output. Every computer system in the world can be thought of as a machine that takes inputs and responds with outputs.

Examples

Here are some examples of systems that illustrate the concept of inputs and outputs:

- A Weather Forecast system takes as inputs all the data produced by sensors, satellite images and climate models and outputs a forecast for the day's weather.
- Twitter takes as inputs all the status updates, images and other activities of people you follow and produces an output of a feed of (hopefully) relevant, interesting tweets.
- A software compiler takes as input all the source files of a program and produces as output a binary executable.
- Google Maps takes as input an address, which can either be written by the user, or based on a current location, or from a voice command, and, using its stored maps, produces as output a travel route, including a screen to display the route, text directions and voice instructions as the user travels.

Inputs

Inputs are the pieces of data that your system receives from the outside in order to perform its operations.

Types of Inputs

There are a large number of types of inputs a computer system can have. Here are some:

- Keyboard
- Ports, such as USB
- Network
- Touch screen
- Mouse
- Modem
- GPS sensor
- Accelerometers
- Disk, such as Hard Disk Drive, or a USB stick
- Voice commands
- Bluetooth

Considerations

When receiving inputs, it's important to consider that each input comes in a different format, and you will need to convert this input data into a format that is usable within the system before using it. Below are some examples of this kind of conversion:

Aggregation

Many input forms provide a lot of data - for example, a GPS sensor might provide continual GPS readings to the application, but our application may only need one reading every 10 seconds in order

to function. We might then aggregate the data over the 10 seconds and produce one reading that will be used by the system.

Conversion

The inputs may arrive in a form that is not usable by our system. For example, a file might be stored on disk in XML form: we may need to convert that XML data into a set of objects in memory before we can act upon it.

Validation

Inputs are often in need of validation before being used - a user may have used a keyboard to enter a number and may have included an invalid character in that number - this might need to be rejected.

Internal Representation

Inputs arrive at the "edge" of a computer system in a certain structure, or format, and need to be converted into an internal representation based on the needs of the system. For example, you may have a web-based screen where a user can enter a set of information. The user may click a date picker to select a date but the system will need to convert this mouse click into a date value in order to use it.

Outputs

Outputs are the pieces of data that your system produces. These outputs are then used by end-users or fed as inputs into a downstream system in order to continue a business process. The process of providing an output takes the internal representation of data and converts it into a form that is useable by an external device, such as a screen, a device, a network packet or a file on a disk.

Types of Outputs

Just as there are a large number of types of inputs, there are also many ways a system can provide outputs:

- Screen
- Printer or Plotter
- Screen reader
- Writing to files on a disk/USB stick
- Data sent over a network or modem
- Bluetooth

Considerations

When producing outputs, it is important to consider the type of output we are provided and who or what is consuming it. If the output is being provided to another computer system then we will provide it in one way, and if it is being viewed by a user then we should provide it in a different way.

Accessibility

The needs of the end user of the output must be considered, first and foremost. If the output is for the benefit of another computer, then the output should be formatted and sent in a way that is machine-readable. If the output, instead, is for the benefit of a human, then the output should be provided in a way that is accessible to that human. For example, if a human is visually impaired, the output could be provided by using a screen reader, or by using really large fonts. The Operating System can be helpful in providing these facilities, but applications still need to consider them to ensure that the OS tools can be used.

Format

Output must always be formatted - we can't simply take the contents of our program's memory and dump it out (unless, of course, the output is for a debugging application). Instead we must format that data in a way that is consumable. For example, our computer may store integer numbers as 32 bits of binary information, but those are not readable by most humans. When outputting an integer number for a human we should rather format it in a way that is readable by humans. Remember also that different cultures format their numbers in different ways, and this must be considered when outputting data such as numbers and dates especially.

Encoding

Output must be encoded for the destination device. For example, if we are writing a software system that is a web-based server with browser-based clients, we will need to encode the output as HTML so that the browser can render it for the user. If we are instead writing code that will write to a printer, we will need to include different formatting instructions that are applicable to printers. When sending data over the network we will need to consider the encoding and protocol used to do so.

Online vs Offline I/O

An important distinction in all forms of input and output are whether they are online or offline.

Online vs Offline Input

An example of an offline input is a manual form, such as when you submit a deposit slip to a bank teller. In this case, a person is writing information onto a paper form, and then this data will be captured into an online system by a trained bank teller. Even more offline are processes found in many municipalities, where you submit a request for a service (such as electricity) on a piece of paper, and that request is accepted by a customer service agent and then later sent to another department for capturing into the management system.

Offline input has the advantage of being simpler for the end user - most people know how to fill in forms and don't need training in order to do so. However, in order to use an online computer system, they may need training or may find it impossible if they have not learnt how to type or use a mouse.

Offline input has disadvantages too - the data is not immediately available for processing and still needs to be captured into the system manually before it can be used. This can cause delays. In addition, it requires physical presence.

These advantages and disadvantages can be illustrated by banking: the offline system of transacting requires you, the account holder, to physically go to the bank, fill in a form and submit it to a teller so that they can do the transaction you want. The online system of transacting requires you, the account holder, to understand how to use the bank's mobile or web-based online-banking system and puts the responsibility on you to do each transaction correctly.

Online vs Offline Output

Offline output is simply output that is standalone outside the system. It is often manifested in some physical form, such as a print-out, but could also be an email with an attachment that can then be printed by the receiver. Offline output is usually human-readable and portable, which is its advantages. Online output is output that is provided directly from the system via a screen. This has the advantage of being immediate, but cannot be taken away from the system easily.

GUI vs Terminal Input/Output

The usual modern method is to provide graphical user interfaces (GUIs) for end users to provide input and to show output. However, there are times where terminal text-based input/output may be preferable.

GUIs have become the standard way of presenting information to users and enabling users to interact with software because they are much easier for users to understand and to learn. A button on a screen that says "Print" is much easier for a user to understand and to do than having to type out a "print" command or learn a shortcut such as Ctrl-P.

However, there are times when text-based terminal-type user interfaces can be better. More specifically, text interfaces can be much faster to use by someone who is experienced in the particular application's use. This is why it is important to consider who your users are, what their level of ability is and what their speed/productivity requirements before deciding on how to design your application's user interfaces.

Designing Inputs and Outputs

Designing the inputs and outputs of a system is key to understanding what the system is that you are trying to build, and why you are trying to build it. The inputs and outputs of the system are what enable it to perform its function, so it's important to consider what its inputs and outputs will be before writing any software system.

Consider the Operating Environment

When you are thinking about the inputs and outputs of a system, you should consider the environment within which the system will be used. Is the user able to input information into the system easily? Is the user able to easily access all of the information they require for the system as output? Let us have a look at some aspects of the environment and the effects that they will have on the way that you design the system inputs and outputs.

- Lighting
    - For a system that is used outside in the sunlight you would need to create a design that has a higher contrast and does not use subtle variations in colour to communicate anything.
    - For a system that is used in a low light environment you would need to create a design that does not hurt the eyes by throwing out too much light.
    - The usage of colour can also have an effect on the user. For example, if the screen emits more blue-toned light then it can actually affect a person's ability to fall asleep. Thankfully this consideration is built into modern devices that engage a night time mode to reduce the light that the screen emits that would keep you awake.
- Noise
    - In a noisy environment, relying on sounds for notifications or alarms may be ineffective. Voice control would also be very difficult.
    - In an environment that is designed to be quiet, using sound as a form of output could also be very invasive and not appreciated by the users. For example, think about a situation where you are sitting in a waiting room, scrolling through social media, and a video starts playing loudly as you scroll past it.
- Equipment
    - Will the operator of the system be wearing gloves? A touch display could be unusable unless special gloves or a stylus is used.
    - Will the user be carrying the device or will it be mounted?
    - Will it be on a phone, tablet, or a desktop?
    - Is it a touch device, or does it require keyboard and/or mouse or any other input peripheral?
    - Will the user be standing or sitting while using the system?
    - Will the user be using the system with both hands, one hand, or with their thumb?
- Access to other Applications
    - In many company environments there is a restriction on other applications (for example, social media, etc.) in order to reduce the number of interruptions that the users experience. Being aware of this is important to ensure that the device used to access your system cannot be used for other purposes.
    - If the device used for your system is single purpose then you may need to design your system is such a way that it cannot be exited intentionally or unintentionally by a non-administrator.

Consider the User Demographics

We should be careful as to not exclude a specific demographic of our users from the usage of our system because we have not considered them in designing the system. Some environments will narrow the demographics that we should consider in designing the inputs and outputs of the system. For example, if we are designing a system for a factory environment we would not need to consider that the system would be used by an infant! Here are some specific factors to consider with regard to the user:

- Age
    - Sight or hearing loss
    - Lack of fine grained control (see dexterity)
    - For infants, does the ability to close the application need to be restricted
- Familiarity with Technology
    - Consider the level of computer literacy that you expect your users to have. You may have to simplify your design or add more guidance to assist the user.
- Skin colour
    - For example, a face recognition system using AI may have only used training sets that included people with lighter coloured skin. This could cause the system to unknowingly exclude functionality for users with darker skin.
- Dexterity
    - If your system can only be operated by a mouse it is excluding users that may have limited dexterity.
    - Providing very small input controls could exclude users with larger fingers or that lack accuracy in their movements.
- Disability
    - Users with bad eyesight would need the ability to increase font size and contrast to use your system.
    - Blind users would need the ability to use screen readers or similar assistants to use your system.
    - Deaf users would need to be considered when your system uses sound as a form of output. Closed captions can be used on video and notifications or alarms should always have a visual component.
    - Epileptic users would have difficulty using a system with flashing lights. Although the use of flashing lights is not a design trend, it is considerate to ease the transition between screens of a system to limit the possibility of your system triggering an epileptic seizure.

Principles of Good Design

There are six basic principles of good input/output design, which can be summarized as:

- User Involvement
- User first, computers last
- Minimize human efforts
- Remember human limitations
- Convention standardization
- Cultural bias

These are described briefly as follows:

User Involvement

- The user interacts with a system through the interfaces (input/output)
- To increase the satisfaction of the user, the analyst should consult the user throughout the design of the user interfaces
- The primary goal of input/output design is to assist the user in their tasks
- If the design is good, the users perform their task easily and make few mistakes. If the design is bad, the users grumble and make mistakes
- Good design enhances user productivity
- To develop an acceptance/rejection system

User First, Computers Last

- The design choices must be made to create the best design for the users rather than the easiest design for the programmers to code
- The interfaces must be formulated to assist the user, but if the system performance will be severely degraded, compromises may be necessary

Minimize Human Efforts

- Whenever possible, the interaction of the human with the computer should be minimized
- The number of keystrokes by the user should be minimal
- Avoid manual data entry to minimize data error
- Do not key in data that can be retrieved from the file or database
- Do not ask for a value that can be computed by the system
- For example, using the zip code to identify a client's city and state is more appropriate than having the user enter the city and state values

Remember Human Limitations

- Humans are prone to errors, and hence checks should be placed in the system to prevent the entry of erroneous data
- Screens should not be cluttered with too much information that can overwhelm the user
- Information carried over to more than three screens should be avoided. This is to reduce human empowerment of short-term memory and pattern recognition
- The system menu and the screens should be linked such that screens can be called up without going through multiple screens

Convention Standardization

- If other systems of an organization have some standard that the users are used to, implement the same to the new system
- Standardization assists the user by eliminating the need to remember many operating procedures
- For example, if function F1 is used to display help features of other information systems, adopt the same for the new system

Cultural Bias

- Design of input and output should be consistent with the natural human expectation
- Screens, forms, and reports should follow data representation from left to right and then from top to bottom (or as necessary for the relevant cultural conventions)
- Computer messages such as error messages should use common terms used in the work area rather than expressions preferred by a particular user or programmer

Design of System Input

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OCR, etc. Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties −

- It should serve specific purpose effectively such as storing, recording, and retrieving the information.
- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on user's attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding −
- What are the inputs needed for the system?
- How end users respond to different elements of forms and screens.

Objectives for Input Design

The objectives of input design are −

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.

Data Input Methods

It is important to design appropriate data input methods to prevent errors while entering data. These methods depend on whether the data is entered by customers in forms manually and later entered by data entry operators, or data is directly entered by users on the PCs.

A system should prevent user from making mistakes by −

- Creating a clear form design by leaving enough space for writing legibly
- Giving clear instructions to fill in the form
- Reducing key strokes or clicks required in order to complete the form
- Giving immediate error feedback when the user has provided invalid input

Some of the popular data input methods are −

- Batch input method (offline data input method)

- Online data input method
- Computer readable forms (using OCR or as an editable PDF)
- Interactive data input

Design of System Output

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

Objectives of Output Design

The objectives of output design are –

- To develop output design that serves the intended purpose and eliminates the production of unwanted output
- To develop the output design that meets the end users' requirements
- To deliver the appropriate quantity of output
- To form the output in appropriate format and direct it to the right person
- To make the output available on time for making good decisions

External Outputs

Manufacturers create and design external outputs for printers. External outputs enable the system to leave the trigger actions on the part of their recipients or confirm actions to their recipients.

Some of the external outputs are designed as turnaround outputs, which are implemented as a form and re-enter the system as an input.

Form Design

A form is a formatted document containing blank fields that users can fill in with data. It is a basic business tool (whether printed or electronic) for collecting and transmitting information.

The following guidelines primarily describe the design of a physical form, but these principles do transcend just the physical realm because they address efficient communication with users in order to gather or convey information.

Classifications of Forms:

1. Action form: It requests the user to do something that is get action. It goes from one place (person) to another. Examples are purchase order, shop order.
2. Memory form: A memory form is a record of historical data that remains in a file, is used for reference, and serves as a control on key details. E.g. purchase records, inventory records.
3. Report form: It guides the supervisors and one administrators in their activities. It provides data on a project or job. E.g. profit and loss statements, sales analysis report etc.

Requirements of a Form Design

The purpose of a form is to communicate effectively through form design. There are several major requirements:

1. Identification and wording:
   The form title must clearly identify its purpose. Columns and rows should be labelled to avoid confusion. The form should also be identified by firm name or code number to make it easy to reorder.
2. Maximum readability and use:
   The form must be easy to use and fill out. It should be legible, and uncomplicated. Ample writing space should be provided for inserting data.
3. Physical factors:
   The form composition, colour, layout should lead themselves to easy reading. Pages should be numbered when multipage reports are being generate for the user.
4. Order of data items:
   The data requested should reflect a logical sequence. Related data should be in adjacent position. Data copied from source document should be in the same sequence on both forms.
5. Ease of data entry:
   If used for data entry, the form should have field position indicated under each column of data and should have indication of where decimal points are.
6. Size and arrangement:
   The form must be clearly stored and filed. It should be provided for signatures.
7. Use of instructions:
   The instructions that accompany a form that shows how it is used and handled.
8. Efficiency considerations:
   The form must be cost effective. This means eliminating unnecessary data.
9. Type of report:
   Form design should also consider whether the content is executive summary, intermediate managerial information or summary data. The user requirements for each form determine the final form design.

Types of Forms

1. Flat forms:
   A flat form is a single copy form prepared manually or by a machine and printed on any grade of paper. For additional copies of the original, carbon paper is inserted between copies. It is the easiest form to design, print and reproduce. It is the least expensive.
2. Unit-set/ Snap out form:
   These form have an original copy and several copies with one-time carbon paper interleaved between them. The set is glued into a unit for easy handling. The copies are perforated at the glue margins for tearing out, although the carbon is not perforated. The carbon paper is approximately 3/8/ inch shorter than the copies. Because of the perforations and the shorter carbon, the forms can be easily snapped out after completion.
3. Continuous strip/ fanfold forms:
   These are multiple unit forms joined together in a continuous strip with perforation between each pair of form. One-time carbon is interleaved between the copies, which are stacked in a fanfold arrangement. The fanfold is the least expensive construction for large volume use.
4. NCR (No Carbon Required) form:
   Several copies can be made by pressing a chemical coating into the top sheet into claylike coating on the top of the second sheet. The writing pressure forms an image by the coating material. The same process is repeated on the back of the second sheet for producing a carbon copy on the succeeding sheet. It offers cleaner and long lasting copies. One problem with NCR is that it is costly.

Layout Considerations of a Form

When a form is designed, a list is prepared of all the items to be included on the form and the maximum space to be reserved. The list should be checked by the firm to make sure it has the required details:

1. Form Title and Number:
   The first consideration in the form design is a brief, descriptive title that tells us what the form is and what it does. Since we read from left to right and top to bottom, the upper left corner of the form is an appropriate place for a title. On forms, that goes outside the organization the title is placed in the centre at the top of the form. Long title should be divided. Good titles are often no longer than two words.
2. Data classification and zoning:
   All items in the form should be logically grouped. Then the data is placed in the appropriate zones. Thus"
   a. A form is divided into zones (zones represent a group of similar information)
   b. The zone uses Upper Left Corner (ULC) method
3. Rules and Captions:
   In designing forms, use rules (lines) to guide the human eye to read and write data groups. A caption is similar to column heading. It specifies what information to be written in the space provided rules and captions go together. Rules guides and separates whereas captions guides and instructs.
   A form is designed with a combination of rules and captions. Rules can also be used to make boxes in which the users places data. The caption tells the user what information goes in a particular position.
4. Box Design:
   Whenever possible, it is advisable to design the form using the box design rule with captions in the upper left corner. The box design gets the caption out of the way. It also makes data entry interrupted from left to right.
5. Spacing requirement:
   In a form there must be sufficient space to allow for data capture. A most commonly used standard is 3/5 spacing. the 3 refers to the number of lines per vertical inch, while 5 refers to the number of characters that fit in one horizontal inch. There are times when the number of lines have to be maximum.
6. Ballot box and check off design:
   Using ballot or check off boxes for questions that can be answered by a "Yes" or "No" can greatly reduce the amount of required writing. The user indicates the preference by simply by checking off the desired box or placing a mark in the appropriate space.
7. Form instruction:
   A well designed form with clearly stated captions should be self-instructing. A form becomes self-instructing by means of clearly stated captions and brief procedural instructions.
8. Paper selection:
   Forma may be printed on papers of different colours, grades and weights. Coloured papers or coloured printing on white papers is used to distinguish among copies and facilitate sorting copies.

Form Controls:

The first step in form control is to determine whether a form is necessary. Managing hundreds of forms in a typical organization requires control program. Form control is a procedure for:

1. Providing improved and effective form.
2. Reducing printing cost
3. Securing adequate stock at all times

Before launching a form control program, the designer needs the consider several questions:

1. Who will be responsible for improving form design?
2. Should the form be produced within the organization or assigned to outside printer?
3. What quantity should be printed?
4. What should be the life of a form?
5. Where and how should the form be stored?

Implementing Inputs and Outputs

Once you have your system planned to some degree, it's time to implement it in code.

When it comes to building your system in code, one key aspect to select is your frameworks, or "stack". Most user-centric applications these days are built using either web technologies or mobile technologies, with web being the most common because it is usable on all types of devices.

Here is a sample stack that can suit a typical web development project:

- Front End (the user interface):
  - HTML5/CSS
  - Bootstrap for layout
  - ASP.NET MVC with Razor for dynamic elements
- Back End (the server side):
  - ASP.NET Core
  - Entity Framework for persistence
- Data storage:
  - SQL Server

This is a simple stack and won't suit every application, but will fit many small to mid-size applications.

The details of these different technologies are covered in other unit standards, so this section simply provides you with some ways in which these technologies can help you deal with data.

Front End

HTML5/CSS

HTML5/CSS is the universal standard for web layout. These technologies include some key features related to inputs and outputs:

- HTML includes many controls that are used for input. In particular, the <INPUT />
- Web browsers also provide automated readers and ways of customising layout for hearing impaired and visually impaired people
- CSS includes ways of responding to different screen sizes to allow one page to be rendered clearly on many different screens. You can also provide a specific CSS file to use for printing in the case that the user would prefer the output to be printed.

Bootstrap

Although you can build your site using just HTML5/CSS without any help, there are some really good UI frameworks that helps you to layout pages in a way that takes a lot of the effort away by bootstrapping your site.

Bootstrap templates are a great way to start - these are just structural and don't contain much design elements: https://startbootstrap.com/templates/

Bootstrap themes go one step further and include full design elements like backgrounds and colours. https://startbootstrap.com/themes/

ASP.NET MVC with Razor

ASP.NET MVC is a web framework created by Microsoft that makes it easy to create pages that display data retrieved from a database, and to receive data from the browser so that it can be processed.

When using MVC, a full HTML page is built from a template on the server and sent to the browser on the client to be rendered for the user. While the page is being built, the server can load data from the database and populate fields, or perform calculations. Once the page is ready, it is sent to the user. Then, once the user has entered data, or performed some action, the page content (including the data entered) is sent back to the server for processing.

The Razor layout engine includes many tools for creating editors for data as well as validating data that is entered by the user.

Back End

ASP.NET Core

ASP.NET Core is the server-side technology that hosts the ASP.NET MVC engine. Once the ASP.NET MVC engine has received and validated the data you can now process it in whatever way you need to. Very often this will mean persisting it to a data store, but it can also mean sending it on to another application, sending an email or logging an event in a file. This can all be implemented in C# using ASP.NET Core.

Entity Framework for persistence

The most common data operation is to save and load to and from a data store, and the most common data store in practice is the SQL database. Entity Framework is a powerful library provided by Microsoft that can be used to persist your in-memory objects into database tables.

Data Storage

SQL Server

SQL Server is a very powerful SQL Database Management System and a very reliable way of storing your system's data. It is important to consider the structure of your data, as well as how it will be accessed. For example, if you will need to search on a particular field, then that field should be indexed in order to make searches on that field fast.

SQL databases are also excellent storage methods for reporting from - many reporting tools work well with SQL databases, so if one of your key outputs is daily, weekly, or monthly reports, this can be done directly from your SQL Database using a report builder tool (such as, for example, Microsoft's RDLC).

Design Diagrams

In this section we will look at how you convert the high-level analysis you have done into a system design. Most of the tools here utilise diagramming. A diagram can be more useful than a description because they can provide more clarity and precision than text can. They are also often much easier for non-technical audiences to understand and interact with than text on a page is.

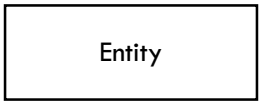
A note about diagrams: only create them if they are useful - they are a tool to allow people to think, to collaborate, to discuss and decide. They are not an end in themselves - they are a means to more understanding and clearer communication.

Data Flow Diagram

The Data Flow Diagram tries to represent the way data flows through a process. So, for example, if there is a purchase requisition process, the DFD for that will show how a purchase requisition is initiated, where it goes for approval, where it goes next for budget approval, and then where it goes for purchasing, and so on. This fairly quickly identifies the people that are involved in the process and what their parts of the process are, which is essential if we are intending to digitise a process.

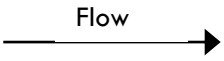
A data flow diagram consists of the following components:

External entities, represented by rectangles. These are the sources or destinations of data external to the system



Processes, represented by rounded rectangles. These transform, or process data, taking in data and outputting the transformed data

Data flows, represented by arrows. These represent data flowing from one place to another - note that data can flow as physical items (documents, forms) or electronic data



Data stores, represented by open-ended rectangles. These are places that data is stored either physically or electronically (e.g. a filing cabinet, or a SQL database)

Example

Below is an example Data Flow Diagram that depicts the data flows for the assessment and moderation process of this Learnership. This was drawn using http://draw.io with their standard drawing shapes.

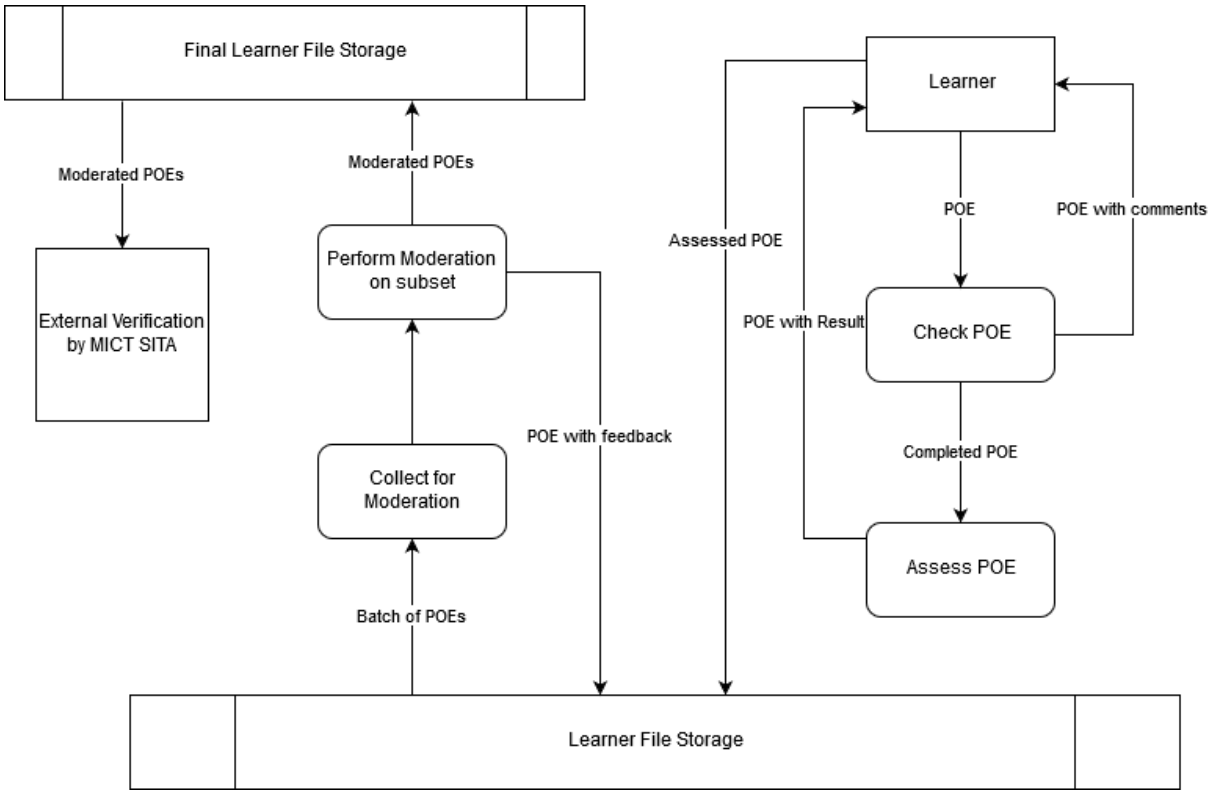


Figure 1: Example DFD depicting the assessment process for this learnership. Drawn using draw.io

Guidance

Here is some general guidance for creating Data Flow Diagrams, taken from Scott Ambler (see below)

- "All processes must have at least one data flow in and one data flow out.
- All processes should modify the incoming data, producing new forms of outgoing data.
- Each data store must be involved with at least one data flow.
- Each external entity must be involved with at least one data flow.
- A data flow must be attached to at least one process."

Exercise: does the example diagram above conform to these "rules"? If not, where does it not conform, and how would you change the diagram to better conform?

Further content

How to create Data Flow Diagrams (YouTube video):
https://www.youtube.com/watch?v=KA4rRnihLlI&t=185s

Scott Ambler's notes on DFD's (where the guidance rules are from):
http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm

Structured Analysis Process Modelling: (excellent detail about how to build DFDs)
https://www.youtube.com/watch?v=b5tVT3n9aec

Wireframes for User Stories

When working with user stories that involve input and output screens it is very helpful to draw up wireframes. Wireframes are hand-drawn (or tool-drawn) pictures of screens that show how screens are laid out and how the interaction works. The idea is to focus on the structure and content of screens, not on how they will look (no colours or other details).

Example

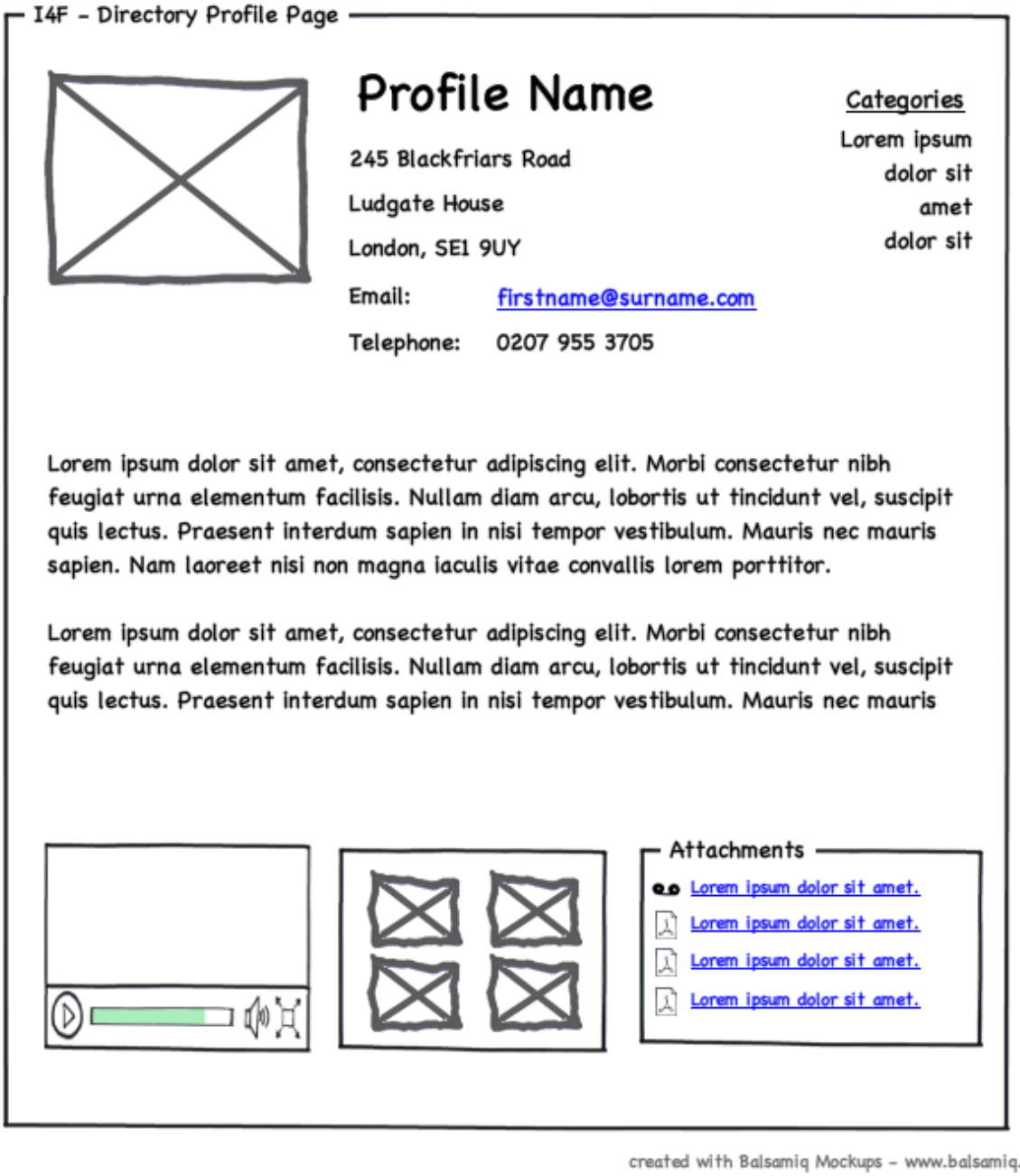


Figure 2: Example wireframe showing a profile page. From the Wikipedia page about wireframes (https://en.wikipedia.org/wiki/Website_wireframe)

More examples can be found at: https://balsamiq.com/wireframes/

Guidance

Some general guidance for using wireframes:

- Focus on structure and content
  - general layout is important but specific location isn't
  - colours are not normally used - pure grayscale means people are not distracted by choosing colours
- Use a lo-fi tool, such as hand-drawing, or a diagramming tool that doesn't make the UI elements look like real ones.

Further content

Wireframing for Newbies: https://www.youtube.com/watch?v=KnZrypOaVCg

Using http://draw.io to create wireframes: https://www.youtube.com/watch?v=GDPDIPj5XWY

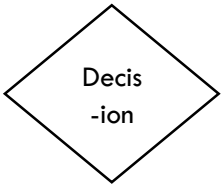
Activity Diagram

An activity diagram is a graphical representation of a workflow. In some ways it is similar to the Data Flow Diagram, but it also allows for choice, loops and split flows within it, so it is more suitable for detailed sequences. Data Flow Diagrams tend to be useful at the context level - how the overall system fits into its environment - the activity diagram is useful at the granular, detail level.

Activity diagrams are also similar to classic flowcharts, but they provide more richness by adding some extra shape types. An Activity diagram (as defined in the Unified Modelling Language 2.x), has the following types of shapes:

Actions, represented by ellipses or rounded rectangles.

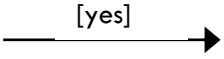
Decisions, represented by diamonds



Start (initial node), represented by a black circle with no text inside it

End (final node), represented by a black circle with another circle drawn around it

Sequencing, represented by arrows. Arrows indicate the sequence of activities within the process. They can have labels next to them when there is more than one path (i.e. out of a decision) to indicate which path is followed in which case.



Split and join of concurrent activities, represented by bars.

Example

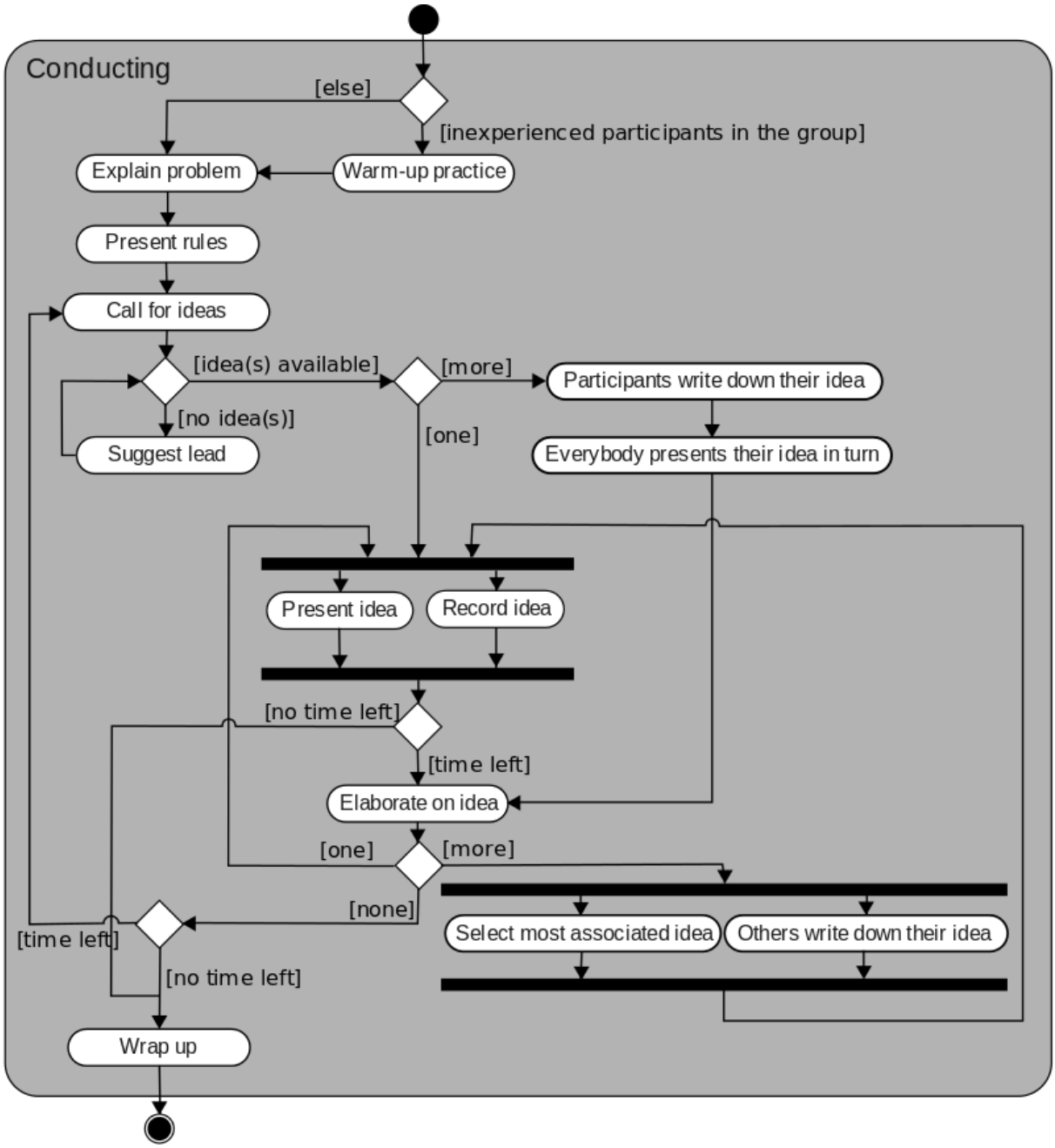


Figure 3: Activity diagram example showing the process for a brainstorm session. From Wikipedia

Guidance

- Keep the activity diagram small - focus on the specific process you are modelling and us the start and end symbols to bound this process.
- You can link together different activity diagrams to show larger processes - do this by linking one diagram's end node with another diagram's start node through labels
- Think of the activity diagram as a way of understanding a process in order to simplify it or break it into smaller pieces. The fact that you need an activity to make sense of a process means it is reasonably complex and could possibly be broken into smaller sub-processes that are each simpler.

Further Content

Systems Analysis and Design: UML Activity Diagram:
https://www.youtube.com/watch?v=FouObgZc6oA

Scott Ambler's Activity Diagramming Guidelines:
http://www.agilemodeling.com/style/activityDiagram.htm

Class Diagram

The Class Diagram is a way of representing the object model of your application solution space. It is also part of the Unified Modelling Language, a set of diagram specifications designed specifically for software development.

The Class Diagram focuses on the logical relationships between entities, as well as what data each entity contains. It is a way of thinking about how the various entities you are trying to model relate to one another. Being a Class Diagram, it includes concepts like inheritance and composition.

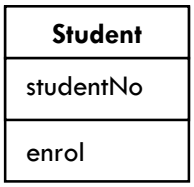
A Class Diagram (as defined in the Unified Modelling Language 2.x), has the following types of shapes:

Classes, represented by boxes containing the name of the class.

Classes can have an optional *stereotype* - the most common are *interface* and *value type*, which are constructs most OO languages support. Stereotypes are written above the class name in double angle-brackets.

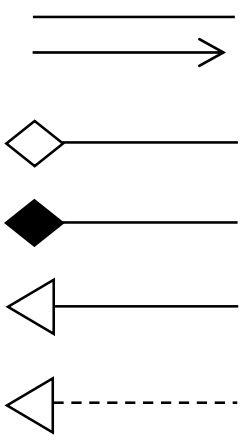
Class boxes can have three compartments which provide more detail:

- Top: the name of the class
  Format: bold and centred
- Middle: the attributes of the class
  Format: camel-cased, left-aligned
- Bottom: the operations the class can execute
  Format: camel-cased, left-aligned



Relationships, represented by arrows of various types. These can be named; each end can be named and each end can have different multiplicity:

- Association: simple line represents a bi-directional relationship). Standard arrow represents a one-way relationship
- Aggregation: empty diamonds denote that the relationship is a "has a" relationship.
- Composition: filled-in diamonds denote that one class is contained within another
- Inheritance: triangle denotes that one class inherits from another
- Implements: triangle with dashed line denotes that one class implements another (that has the interface stereotype)



Example

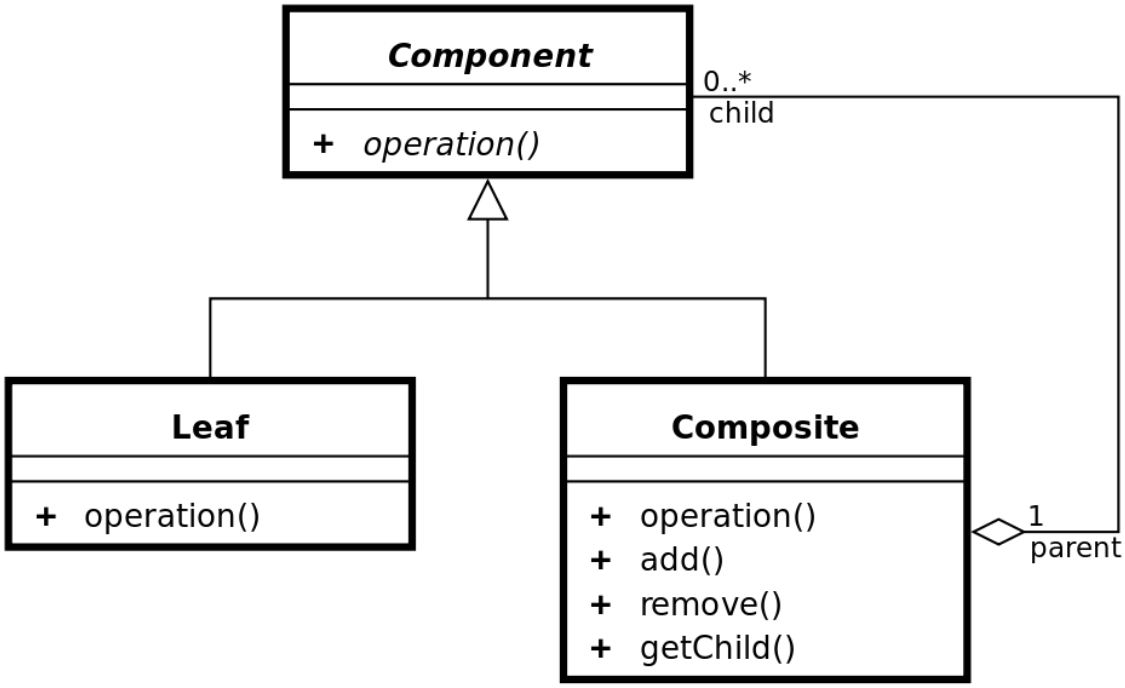


Figure 4: Class diagram showing the "Composite" pattern. From Wikipedia

Guidance

- In general, avoid using inheritance. It can be useful, as shown in the example, but it presents difficulties in high coupling and in persistence (there is no such thing as inheritance in database tables). (Note: in the example, the Component class is shown in italics, which means it is an abstract class - this would be better denoted with a stereotype <<abstract>>, and even better, as an interface with dashed lines to the implementing types - interface implementation creates much less coupling than inheritance).
- Remember that diagrams are tools for thinking, they are not a blueprint for the code. As you code you will learn more about the solution domain than you knew when you created the class diagram, so when writing the code, don't be afraid to change the design to better match your changing mental model.

Further Content

Systems Analysis and Design: Class Diagrams: https://www.youtube.com/watch?v=IGqAbuxCOXI

UML Class Diagram Tutorial: https://www.youtube.com/watch?v=UI6lqHOVHic

Scott Ambler's Class Diagram Introduction: http://agilemodeling.com/artifacts/classDiagram.htm

Relational Database Model

When it comes to designing data inputs and outputs, your logical data storage model is a key component - understanding how you are going to store your data can help with building your system.

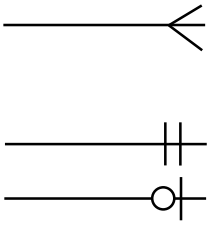
As with all of these diagrams, they are thinking tools to help you reason about your problem and solution domains; they are not specifications for coding up. When writing the software itself, if you find that you missed something in the database model, change it, adapt it to your deeper understanding.

A Relational Database Model (aka an Entity-Relationship Diagram) can have different forms - the one we will use focuses on the logical tables and relationships using crows-feet notation:

Tables, represented by boxes with at least two sections: one, the heading, containing the name of the table, and the other containing the fields of the table.

Relationships, represented by lines between tables. These have different endings to indicate multiplicity:

- *Many*: indicated by crow's feet on this side of the relationship
- *One (mandatory)*: indicated by two lines
- *One (optional)*: indicated by a line and a circle
- The mandatory (single line) and optional (circle) modifiers can also added to a *many* relationship



Example

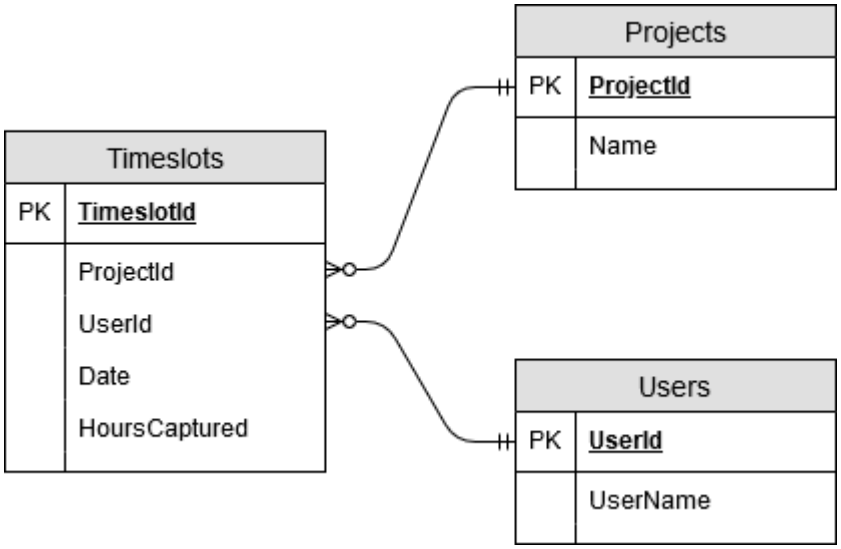


Figure 5: ERD for a simple time capturing system

Guidance

- In your storage model you cannot have many-to-many relationships as no relational database engine supports them. One side of each relationship will need to be a "one" side. This means that if you do have a many-to-many, you will need to add a table in between and replace the original many-to-many relationship with two relationships: a one-to-many and many-to-one to the new table, also known as an *associative* table.

Further Content

Entity Relationship Diagram (ERD) Tutorial, Part 1: https://www.youtube.com/watch?v=QpdhBUYk7Kk

Entity Relationship Diagram (ERD) Tutorial, Part 2: https://www.youtube.com/watch?v=-CuY5ADwn24

System Analysis and Design: Logical Database Design:
https://www.youtube.com/watch?v=lBrHtvGPtcg

Appendix A: Analysis Process and Methods

In this section we will look at some methods you can use to discover, or analyse, what your inputs and outputs should be.

Note: this unit standard is not specifically about analysis and eliciting requirements, it is about designing your system's data requirement. However, in order to do this, some high level analysis will be required, so here we provide brief overview of this topic.

Process

It is fundamentally important that all stakeholders, including users, of the potential system are included in the process. A computer system is typically being built to tackle a problem or opportunity that an organisation has identified, and the people that have identified that problem or opportunity are perfectly placed to provide input into how that problem or opportunity is to be tackled.

As software developers, we bring our professional expertise to bear upon that problem/opportunity in order to create a system that meets the needs of its stakeholders. This should be our top priority when designing a system.

The process you follow will be different in each situation, but there are a set of tools that you will generally use, which we cover in some detail below. The general process is to gather information, analyse that information, then begin designing a solution, and then implement that solution. In Agile methodologies this is done iteratively, so that you feedback learning from implementing (coding) back into the next phase of design.

Workshops

It is fundamental to any system to understand what the problem is that you are trying to solve. A workshop with key business stakeholders is a way of gathering this information. Workshops can also be used during the information gathering process and throughout the software development life-cycle.

A workshop is simply "a meeting at which a group of people engage in intensive discussion and activity on a particular subject or project". From this definition, we can see that we need:

- People: a successful workshop needs at least two people. Preferably these people should be decision makers.
- A subject, or project: in order to run a workshop, we need a subject. Initially, this will perhaps be a vague question, but the workshop should seek to clarify this.
- A facilitator: a group of people engaging in intensive discussion needs someone to facilitate it; that is, to lead the conversation or to at least set boundaries of time and to organise a venue

It is also clear that some outcome is required. A workshop should have a specific goal, and enough time should be allocated to achieve that goal. The initial project workshop should have the goal of answering the following questions:

- What is the system? Why is it needed?
- Who are the key stakeholders?
    - Sponsor - the person (or people) paying for the system
    - Champion - the person tasked with ensuring the system is implemented
    - Subject matter experts - the people that are most knowledgeable about the subject matter - these people may or may not be users of the system. For example, they might

be a legal expert, or the accountant, who will not be using the system but will understand the rules that need to be applied
- Key users - the people that will be the main users of the system

Example: Your customer, a local sports club, has asked you to build a member management system. To be able to start to plan how to go about doing this, you might schedule a workshop with the following parameters:

- People: The club's chairperson and key committee members and yourself
- Subject: Initial workshop to discuss the goals of the proposed member management system
- Facilitator: yourself
- Planned outcome: The goals of the member management system; that is: why is it needed, and what is expected to be achieved by introducing an IT system to replace the current manual system? In addition, who will be the key stakeholders involved with the system definition and implementation?

It is very important that good documentation is completed once the workshop is over. Key documentation that is required is the following:

- Attendees
- Date
- Subject
- Key points discussed
- Key decisions made
- Next steps

This documentation is intended to provide a reference for all attendees so that any important decisions made are documented. This helps to ensure that all attendees agree on the outcomes: it may even help to write the decisions down within the meeting so that all can agree that what is documented is what they have agreed upon.

JAD Sessions

One of the questions that should be clarified in the initial project workshop is that of "who are the key stakeholders?" These are the people that you will be engaging with throughout the project in order to deliver on the goals of the project. They are also the people that will be the most useful to engage with in defining the inputs and outputs of the system.

One way to do this is through JAD sessions: Joint Application Design sessions. These are facilitated workshops that include many different stakeholders and are used to elicit the requirements of the system.

These JAD sessions usually include many different stakeholders so that all the right people are in the room to make the key software requirements decisions. It's important that they are facilitated so that all voices are heard. It's also important that they are fully documented - appointing someone to be a scribe, or recording the conversations, can help a lot here.

The method used in a JAD session can be one of many methods - these are discussed in the Documentation Methods section on page 30.

Example: Now that you have the overall goals of the Member Management system and you have information about the key stakeholders, you can plan a JAD session with those stakeholders. To do so you will need to:

- Set a date: You will need to find out when all participants are available for enough time
- Arrange a venue: if all participants are from one company you could use a meeting room. Alternately you could use an off-site venue, but this is usually more costly
- Invite the participants: Remember to invite the sponsor, the subject matter experts and the identified key users, as well as a scribe and a facilitator (you could take on facilitator role)
- Prepare: to prepare, you will need to decide on your method, and prepare to explain the method to the participants so that they know how to contribute. Then you will also need to prepare a list of questions that have already come up so that you can address them.
- Agenda: It is really helpful to participants if they have a clear agenda for the sessions ahead of time so that they know what the times of different activities will be and what will be expected of them. An example agenda for this project might be:
    - 8h00: Welcome and introductions
    - 8h15: Project goals presented by the project sponsor
    - 8h45: Brainstorm requirements
    - 9h30: Tea
    - 9h45: Prioritise requirements
    - 11h00: Brainstorm risks
    - 11h45: Define next steps
    - 12h00: End

Brainstorms

A brainstorm is simply an activity where a group of people come up with ideas together. For a successful brainstorm the following is necessary:

- The question, or boundary of the brainstorm: what are we brainstorming about?
- The process: how is the brainstorm going to work?
- The time: how long is it going to take?

Brainstorms are used to generate ideas: the initial phase of a brainstorm is creative, where people come up with ideas. At some point this phase ends, and the second phase begins, where people try to determine which ideas are the most applicable, or best. During the first phase is all ideas are included, during the second ideas are pruned away to leave the ideas the group decides are the best ideas.

In the context of requirements gathering, a brainstorm can be used to enable all stakeholders to come up with their requirements. This can be done by using sticky notes and pens. After each person has written down the requirements they think are needed in order to achieve the goals of the project, they can stick them up. If there are stickies that need explanation then the person who wrote that item can explain what they mean so that all people understand each sticky. Once all stickies are up on the board, you, as facilitator, can then group these stickies so that duplicates are put together. Now that you have set of requirements that this group has discovered, it is really helpful to prioritise them - to do this, take a pair of stickies and ask the following question "which of these requirements is more important for reaching the goal of this project?". The group can vote on which they see as more important. Usually there will be consensus - if there isn't then this should be unpacked as this has discovered a difference in understanding between stakeholders and is an opportunity to learn about this and work towards a more common understanding. Once each pair is decided, go on to another pair, and continue this until all the stickies are ordered from most important to least important. This gives you as the analyst a lot of information about where to focus your efforts.

Further Content

Requirement Gathering | Workshop: https://www.youtube.com/watch?v=6J-_0PCkNfE

Interviews

Interviews are an excellent way of understanding the viewpoint of different people independently. Where a JAD session can generate a lot of information fast, it can be wasteful if it is too detailed. This is because not all stakeholders in a JAD session are interested in all processes or areas. Smaller JAD sessions can be scheduled in different areas if that will help, otherwise interviews can be used to discuss specific processes in more detail, and to address questions that have arisen in the JAD session, or in other interviews.

An interview is essentially a discussion between two people - the interviewer and the interviewee. It can include more than one interviewee if they are sufficiently similar in outlook - an interview with a credit controller and the CEO will not be likely to be fruitful as they will have vastly different knowledge areas - one will be knowledgeable in the detail of the rules and exactly how things are done in the credit control area day in and day out, and the other will be knowledgeable in the overall strategy of the company. If such different are to be included, it is better to run a JAD session instead of an interview.

Ensure that you are prepared for the interview with questions that you need to answer. Write notes, and write down questions that still need to be answered in future discussions. If necessary, record the interview (with the permission of the interviewee), but note that listening to recordings is very time-consuming so it is better to take notes while you are talking. Take the time to note down things, and let the interviewee know that you are doing so. After the interview, read through the notes and type them up in a way that is as fleshed out as possible right now. If, in the fleshing out process, you find that you do not fully understand something, write down your question and go back to that person later: an interview is not a once-off activity.

Further Content

Requirement Gathering | Interview: https://www.youtube.com/watch?v=I1RIhmf0IlI

Observation

Finally, but one of the most important for generating requirements: observation. If you are digitising or automating an existing process, then observing how that process actually works in practice is very helpful. This requires shadowing, or observing a person in their daily work for a reasonable period of time (e.g. a few days) so that you see how they go about their job, what kinds of problems can come up and who they interact with to get things done.

Observation helps especially with understanding what kind of software is going to work - will a mobile application work, will it need a desktop solution, or can it be delivered as a web solution? Considering the actual end user and what they are trying to achieve can really help with user adoption, change management and implementation success.

Further Content

Requirement Gathering | Observation - Gather Requirements by Observing: https://www.youtube.com/watch?v=XWhxsrtxF6c

Questionnaires

Questionnaires are excellent ways of getting information from a lot of people at once. With questionnaires you can ask very specific questions and see how people respond. Questionnaires are not going to be useful to understand business processes, but rather they will be useful in understanding

which parts of a system are most important to users, or what features they would most like to see in the systems they use. This can be a useful data point in preparation for a JAD session, or in preparation for interviews, or to help with prioritising requirements.

Studying Documents

Many businesses have paper documents or manually created reports already in use for information flows - if these do exist then they are very useful to study in order to help elicit requirements and determine data inputs and outputs.

Further Content

Systems Analysis and Design: Alternative Methodologies: https://www.youtube.com/watch?v=PN7GSsqKJAQ

Systems Analysis and Design: Determining System Requirements: (covers interviews, workshops, studying documents and observation) https://www.youtube.com/watch?v=nKWQh7mp5GI

Systems Analysis and Design: JAD, Agile, etc: (covers JAD sessions): https://www.youtube.com/watch?v=yJpxArLb2i4

Documentation Methods

There are many methods for documenting what a system's inputs and outputs are, and the method selected depends upon the type of system, its context and even the methodology of the organisation.

Business Use Case Diagram

A Business Use Case Diagram is used to define who the actors of a system are and what their goals are. It is a very high level diagram and can be used early in the process to clarify stakeholder understanding of the system, who will use it and what they will need to achieve with the system.

Use case diagrams have the following components:

- *Actors*: these are people in different roles who need to interact with the system to achieve a goal. Sometimes an actor can be a non-person such as another system. In other cases actors are people fulfilling a role with respect to the system (e.g. Administrator, Customer, Member).
- *Use cases*: A use case is a goal identified by an actor that will require the use of the system. For example, if we are implementing an online shopfront for our company, a Customer (actor) might want to Place an Order.
- *Interactions:* Lines link the actors with the various goals (use cases) those actors are needing to fulfil
- *Contexts*: Each use case can be placed within a context box which identifies a system boundary. For example, some use cases may be placed outside the system boundary box to indicate that that goal is fulfilled outside the system, or it could be placed in a different boundary box to indicate that a separate system (or later extension) will fulfil that use case.

Example

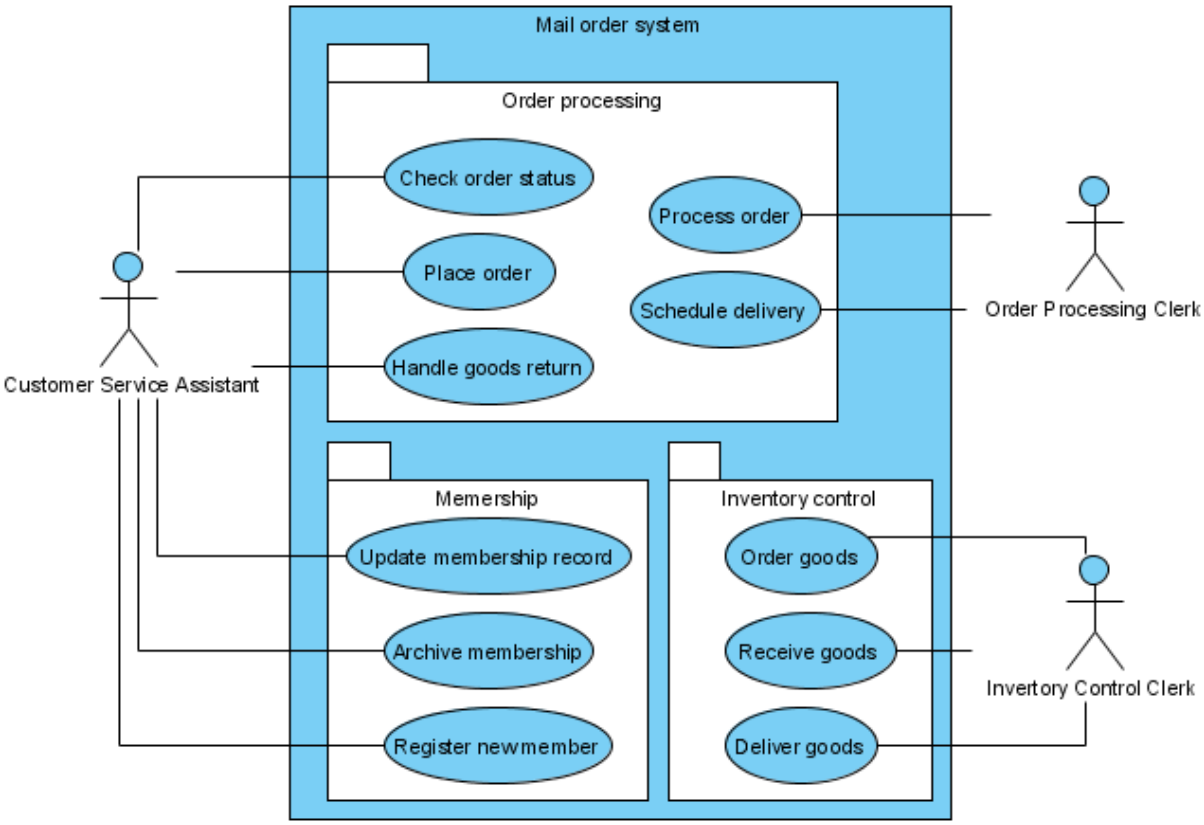


Figure 6: Use Case Diagram showing different actors and different subsystems of a mail order system (https://en.wikipedia.org/wiki/File:UseCaseDiagramExample.png)

Guidance

Although Use Cases can be used to detail the requirements of a system, agile methodologies typically use User Stories instead, and then detail those more with whatever tool makes sense for each user story. However, the Use Case diagram is very useful for understanding the system context at a high level, and which actors are going to do what things in the system.

Further Content

Systems Analysis and Design: Use Case: https://www.youtube.com/watch?v=oXsRdQq4f_A

UML Use Case Diagram Tutorial: https://www.youtube.com/watch?v=zid-MVo7M-E

User Stories

In agile methodologies, the main documentation method of system functionality is through User Stories. User Stories are a short description of a piece of system functionality, with the emphasis on small - user stories should be able to be developed within a few days at most. Typically each use case identified in the business level use case diagram will be broken into many user stories.

User stories are written from the perspective of the user, and have a specific format:

- As a/an <role/user/system>
- I want <required functionality/feature>
- So that <reason/benefit/why>

User stories can be fleshed out further with documentation, but this short description acts as a placeholder for all the rules or other documents associated with it.

Typical documentation associated are:

- Wireframe of the user interface (if the story requires a UI)
- Business rules
- Acceptance criteria

Other documentation that may be helpful, depending on context:

- Activity diagram,
- Class diagram

Example

- As a software developer I want to be able to capture my hours worked on each project for a given day so that the company can bill our customers accurately at the end of the month
    - Acceptance criteria:
        - Able to select the day to capture hours
        - Able to capture hours against each project for that day
- As a user I want to be able to log in to the app using my fingerprint so that I don't need to remember a password
    - Acceptance criteria:
        - Able to log in using the fingerprint reader **or** my password
        - If I put my finger on the fingerprint reader, it should automatically log me in

Guidance

Although acceptance criteria and business rules are not part of the story itself and don't usually appear on the Kanban or Scrum board, they are essential as additional information for the developer and tester.

Further Content

How to write good user stories using 3 key components:
https://www.youtube.com/watch?v=ctFzjMygaRo

How to write user stories: https://www.youtube.com/watch?v=nE8ALJ2M004

User Story in Details for Agile Software Development:
https://www.youtube.com/watch?v=R62lu_4zcpU

User Story Mapping: https://www.youtube.com/watch?v=QYkYdfiwsCM

Further Content

Visual Modelling Techniques: https://www.youtube.com/watch?v=Y_-f2CZELrw

Another useful form of diagram is the Business Process Model and Notation - a way of graphically representing business processes. This can be useful when you need to understand an existing business process or develop a new business process before building software that implements that process.
https://en.wikipedia.org/wiki/Business_Process_Model_and_Notation