|  |  |
| --- | --- |
| **Learner Name** | Husnain Ahmed |
| **Assessor Name** | Chris Livesey |
| **Qualification Title** | BTEC - L3 IT Extended Diploma in Computing |
| **Unit Title** | Unit 22 – Systems Analysis and Design |
| **Assignment No./Title** | 22.1 – The Principles of Systems Analysis and Design |
| **Learning Aim(s)** | A: Examine the principles of systems analysis and design |
|  |  |
| **Issue Date** | 22/02/21 |
| **Planned Submission Date** | 12/03/21 |
| **Re-submission Date (if approved)** | 26/03/21 |

|  |  |
| --- | --- |
| **Feedback** | Provided online in the form of rubric, comments, and general feedback. |
| **Reflection** | Once you receive feedback, you should reflect on your performance using the reflection document. This should be recorded on your Pro Portal.  --> ILP --> 3. My Learner Reflections |
| **SMART Actions/Targets** | You should regularly set SMART Actions/Targets on your Pro Portal.  --> ILP --> 4. My SMART Actions |

|  |  |  |
| --- | --- | --- |
| **Evidence of proof-reading and improvements to quality of communication using Microsoft Editor.** | | |
| Print screen of Microsoft Editor before improvements | Print screen of Microsoft Editor after improvements (1) | Print screen of Microsoft Editor after improvements (2)  Re-sub if required. |
|  |  |  |

The Principles of Systems Analysis & Design

Contents

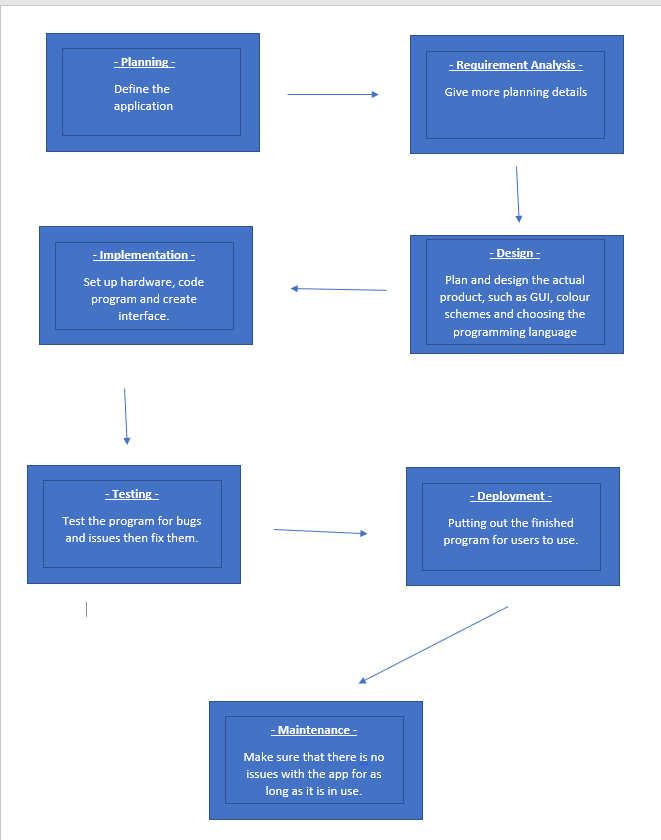
[The Systems Development Lifecycle (SDLC) 2](#_Toc65724663)

[Development Models 4](#_Toc65724664)

[Tools and Techniques 6](#_Toc65724665)

# The Systems Development Lifecycle (SDLC)

Software development lifecycle (SDLC) is the process by which a software is conceptualised, developed, and maintained; its purpose is to produce a high-quality software that is up to the standards of the customer who has commissioned it. It must meet a set of pre-determined requirements and must be completed within a certain timeframe, as well as not going over the allotted budget.



Planning – The planning stage is where you will outline the requirements for your program, this focuses of the scope of the project and deals with things like budgets and schedules. To gain more information on this the stakeholders will undergo a feasibility assessment in many different fields such as finance, laws, politics, and the technical side of things.

Requirement Analysis – This stage is where you will add more specific details to the plans you have created. This includes the specific details of the product and requirements that need to be included inside of it such as a clock app needing the ability to set timers and alarms. There is a need to create detailed diagrams on the product and do some questionnaires so that the planners can know what to add.

Design – This stage is where you will design the program, you create a design specification that will entail the specifics of what the program will do and how it will be achieved and review it to get feedback on what could be improved. Stakeholder feedback is especially important at this stage as they can add more requirements that they may have missed out on earlier. This stage is where you will create algorithms and design the infrastructure.

Implementation – This is the stage where you will produce the program, you will have different teams working on coding the program and designing the graphics and interface (you will also choose whether you will need a database or not). It will be done under a software methodology such as the waterfall model or agile model.

Testing – This stage will have the program tested by people, the feedback they give will be used to improve upon any bugs and issues and ensure that the final program is as perfect as can be. It tests many things from the security of the program to its performance and quality. You will choose the method of testing from Whitebox or Blackbox testing or do both. You will also need to do boundary value analysis and cause-effect graphing to know what to fix. In this stage you identify what went wrong and fix it.

Deployment – This is where the working program is ready to be used by users, it can also be sent to the stakeholders first so that they can test the product and catch any final mistakes before the release of the program.

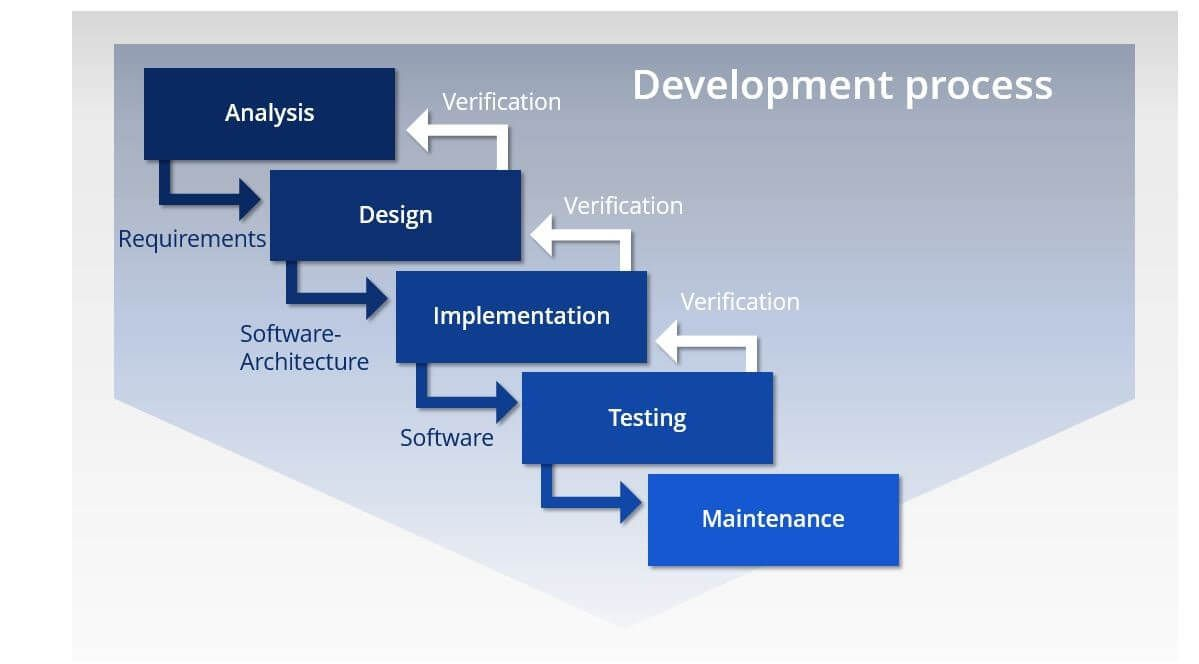
Maintenance – This is where the software is monitored after release to catch any extra issues of bugs that may have been missed in the testing stage, these bugs can be reported by users or caught by staff and then improved upon in an updated version of the application. An example of this happening is with new versions of windows updates to the OS (Operating System) that fix some bugs or add new features.

# Development Models

**Waterfall method –** This software development model works by creating a system in a linear and sequential fashion, it works by splitting up the work into separate phases and then systematically completing them one after another in a downward fashion, hence the name 'waterfall'.

Projects are driven from the top of the hierarchy and completed one after another, the tasks at the bottom of the hierarchy will be completed last.

Planning is required for long term projects as you cannot go back to add to previously completed phases so you will have to have completed the phase to your utmost capabilities the first time around. The complete requirements are clearly documented at the start of the project so that the software development team can know what to do, because of this each phase has its own plan that needs to be followed.



**How it works**

Analysis – this is where you plan and sort out the details of the project, set your budgets and initialise the scope of the project.

Design – this is where you design diagrams for the software, the infrastructure of the software and test plans for all of its components.

Implementation – this is where you create the software and obtain the hardware to run it on for testing purposes.

Testing – this is where you will test the software for bugs and issues and fix them so that there are no issues left over when the software is published and used.

Maintenance – this takes place after the software is published and is where you ensure that there are no new issues popping up with the software and fixing them if they do pop up, also updating the software with new and improved features.

**Pros / Cons**

Pros – All costs and the assigned workload each employee will receive is estimated at the start of the project, this makes it so that all employees are aware of their role in the development of the program.

It is suited for smaller projects where requirements are well defined due to its simple structure. However, due to it being documented in detail it is suitable for larger teams which have people coming and going as they can look at the documentation and know what role they must fulfil.

Projects designed using the waterfall method are properly documented in depth and the time they will take can easily be estimated.

Cons – Error can be fixed only during the phase that they were made in; this is because previous stages cannot be changed or added to once they have been completed and left behind.

It is not desirable for complex project where the requirements change frequently, this is because previous stages cannot be changed or added to once they have been completed and left behind.

Clients valuable feedback cannot be included with ongoing development phase as it is only considered after the development process (testing). This is due to the extremely inflexible linear approach to completing tasks that is enforced by the waterfall method. Due to the user feedback coming too late there may not be enough to go off to improve the project.

**What kind of projects it is suited for?**

The waterfall method is suited for almost all kinds of small-scale projects due to its linear fashion of completing project phases. As it cannot complete complex phases with multiple layers it is not suited for larger projects.

It is better to use the waterfall method for ‘mission critical’ projects such as flight controls, control systems for nuclear facilities / chemical manufacturing plants and payment gateways/processing, this is because these projects do not allow for the slightest bug or error in their use and play particularly important roles. The waterfall method forces the design to be as fool proof as possible as the design will be thoroughly debugged.

The waterfall method is also suited for the building of machines and buildings, this is because it is done in a sequential manner from the ground up and follow a rigid plan that focuses less on the individual workers but more on the processes and tools on hand. (in this case it is called the ‘sequential stage gated system engineering process’)

**What kind of project it is not suited for?**

One type of project that the waterfall method is not suited for is a flexible and less documented project without many requirements and needs such as updating a mobile game application, this is because the waterfall method is extremely strict in its execution and needs all of its requirements to be mapped out with little room for change. This would be more suited to using the agile methodology.

**Conclusion**

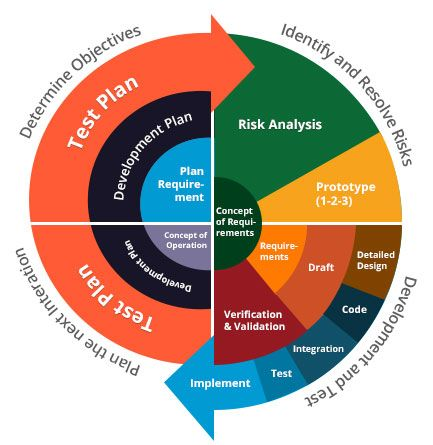
Overall, the waterfall method is best suited to for small scale projects due to its linear fashion of completing project phases. Because it offers less flexibility when compared to other SDLC methods it must be completely documented and decided on beforehand, this makes it so that it is impossible to make any changes to it later in the development.

The use of this method would be a better option for more traditional industries who are already familiar with better structured techniques that offer a rigid, linear, and highly organised structure that can be followed to a tee without the need to take any creative liberties.

Modern technology industries, on the other hand may prefer a more flexible method such as the agile methodology or RAD that is very adaptable to changes and allows them to make changes later in the development if needed.

**Spiral method -** This software development model is designed on the unique risk patterns of a given project, it can adopt the methods of some other development models such as the waterfall method or incremental method. Which is why it is referred to as the most flexible SDLC model.

There is an emphasis on repetition as the planning, design, implementation/development, and testing phases are re-done over and over again with slight changes each time to improve the final program.



**How it works**

Identification – this is where you plan and sort out the details of the project, set your budgets and initialise the scope of the project. Unlike waterfall method you do not have to do it all at once as there is a focus on repetition so on your 2nd time doing the identification stage you can add more details to it.

Design – this is where you design diagrams for the software, the infrastructure for the software and test plans for all its components. Unlike waterfall method you do not have to do it all at once as there is a focus on repetition so on your 2nd time doing the design stage you can add more details to it.

Development/build – this is where you create the software and obtain the hardware to run it on for testing purposes. As it is not done all at once and is appended to over and over again you can send this to the user for feedback and make use of the user feedback you have gained to improve the build quality. The later spirals can be used to add more precise changes and details to the project. Each build has a build number: build 1, build 2 etc.

Evaluation / risk analysis – Identifying the risks with the program and having a user evaluate it and provide feedback that will be used to improve the program in further iterations or the cycle.

**Pros / Cons**

Pros – Continuous or repeated development helps in risk management, as the development is done in several small parts the riskier bits can be coded first so that on further iterations they can be appended to and fixed if there are any issues with them.

Users can see an early model; of the system in the first iteration and give their feedback on it which allow you to fix any problems they have with it in further iterations and add new features if needed.

Additional functionality or changes can be done at a later stage, this means that you a can fix problems that popped up without too much effort.

Cons – Not suitable for small projects as it will take a long time to complete due to the need to repeat the development, this can increase the cost of the project far beyond the budget allocated to such a small project.

For its smooth operation spiral model protocol needs to be followed strictly, the stages need to be followed in the same order in each iteration.

Documentation will be increased exponentially in further stages, depending on the users' requirements the spiral could go on indefinitely.

**What kind of projects it is suited for?**

The spiral method is suited for high-risk projects that are of a medium to large scale, this is because every risk needs to be accounted for and it allows for new features to be added at a further date. It also receives a lot of client feedback which allows he project to keep on track.

It is suitable for the development of ‘mission critical’ projects such as flight controls, control systems for nuclear facilities / chemical manufacturing plants and payment gateways/processing, this is because these projects do not allow for the slightest bug or error in their use and play important roles.

The spiral method is also suited for games development, this is because it can produce a prototype model of the game very quickly and carefully in a way that minimises errors and bugs. Games development focuses heavily on early game models known as alpha and beta stages and they are the played and appended to add more features as well as removing errors with feedback from users, this is done repeatedly in an iterative fashion and is why the spiral method is suited for it.

**What kind of project it is not suited for?**

One type of project that the spiral method is not suited for is a small project with low risks that does not have the need for prototypes such as a CLI quiz or a link that redirects you to another page. Due to the extremely complex nature of the spiral method, it is better suited to large projects with a need for more details. This would be more suited to using the agile methodology.

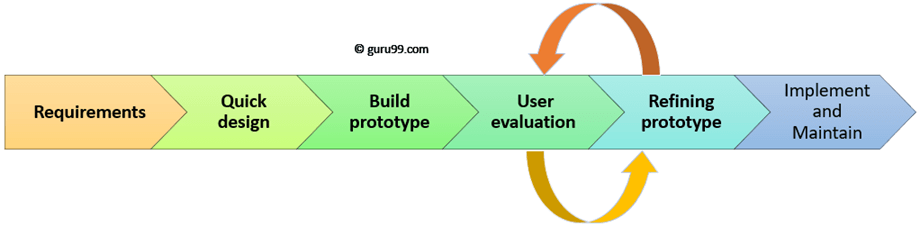
**Conclusion**

Overall, the Spiral method is best suited to for medium to large scale projects with high-risk as every risk will be accounted for in the development of the program, the software is produced in a way that minimises errors and bugs which is particularly good for high-risk projects as they do not allow for any leeway in their function. Because it is very complex as compared to other SDLC methods, such as the waterfall method, it is better if the project is refined a few times in its development which would take far too long for the usual time constraints of a small-scale project.

The use of this method would be a better choice for more traditional industries as the iterative fashion of its development allows for a better understanding of it and allows for the software to be more thoroughly developed, this can calm some people who would be concerned with its competency due to not being familiar with it.

Modern technology industries would also receive help from this as it is very thorough, it also receives a lot of user feedback during development and the final outcome will be what the user was expecting of the final program as they can see the progress made on the software after every iteration of the development.

**Prototype method -** This software development model works by creating a ‘prototype’ version of the final software and showcasing it to the stakeholders to get their feedback on it and so you can know what to improve on for the final design, you will refine this ‘prototype’ and get feedback multiple times until you have your final product.



**How it works**

Requirements – this is where you plan and sort out the details of the project, this does not need to be detailed and more specific information on security and the performance of the project can be put aside until later.

Quick design – this is where you design diagrams for the software, the infrastructure for the software and test plans for all its components.

Build prototype – this is where you build the first prototype model of the software, it does not have to be perfect as it will be refined later but it needs to be able to give the client an estimate of how the final software will turn out.

User evaluation – this is where you showcase the current prototype to the user / client and get their feedback on it to know how to improve it in late r stages. You will also get more details on the project from the clients / users such as budget constraints, time constraints and how feasible the implementation of the software is.

Refining prototype – this is where you use the user feedback to improve the prototype, add new features and fix any issues with it. You will then receive feedback again and then refine it again repeatedly until you are satisfied with your final product.

Implement and maintain – in this stage you will publish the software / program and perform regular maintenance on it to ensure that there are no issues with it such as errors or bugs. You will also perform further updates in this stage if there is a need to add new features again.

**Different types of prototyping**

There are several types of software prototyping, these are some of them:

Incremental prototyping – this is where you build many prototypes of different sub-systems and then merge them to create the complete system.

Throwaway / rapid prototyping – this uses the initial prototype to understand the projects requirements, then scraps it and attempts to develop the final program.

Extreme prototyping – this is used in web development; develops a completely working user interface with no regards to services at all. It is made up of 3 phases. Phase 1: a html prototype of the pre-existing pages. Phase 2: a prototype service layer that simulates data processing. Phase 3: implementing the services to the final prototype.

There can be 2 types of dimensions to prototypes, vertical and horizontal. A ‘Horizontal prototype’ gives a broad overview of the entire product with the UI and is used to gain insights on the surface level of the program for things, such as showcasing the product to other people. The ‘Vertical prototype’ is a detailed breakdown of a specific function or a sub system in the software and is used to gain exact details on how the systems work, such as obtaining the requirements of the database.

**Pros / Cons**

Pros – Reduced time and costs as any issues with the program are detected early on in one of the first iterations of the prototype's development, this allows you to spend your time more efficiently on developing the program and not worrying about errors and bugs.

Improved and increased user involvement and feedback, user feedback is obtained multiple times in the development of this program as each prototype is shown to the user to get their opinion on it. The program is then altered and improved with new functions based on the user’s suggestions and feedback. So, this is a methodology that can easily create exactly what the user is envisioning without any room for dissatisfaction as they can have a better understanding of the system from prototype models.

Functions can be easily identified and understood with the vertical and horizontal dimensions, this allows missing functions to be identified and created as well as allows confusing or complex functions to be identified and better understood.

Cons – There is a risk of insufficient levels of analysis being made due to over reliance on the prototype for the development of the software. Using the user’s feedback as the main method of gaining information on how to develop the program can cause you to ignore the analysis of requirements and missing out on some features that are needed.

This runs the risk of increasing the complexity of the system far beyond the initial scope as users may want to keep adding new features to the program, this could cause it to go over budget and time constraints.

There is also a chance for the users to confuse the prototypes displayed to them as the final product which would cause them dissatisfaction as it would appear to be either unfinished or inadequate in their eyes.

**What kind of projects it is suited for?**

The prototype method is suited for projects that have an exceedingly high degree of user interaction such as an online messaging platform or social media platforms. Users can fill out various forms, the data from which is processed and creates a rough prototype of what the software will look like when it is developed.

It is better to use the prototype method for online projects such as file sharing services, social media platforms and streaming services, this is because these projects all have a high degree of user interaction which can be used to gain feedback on how to improve the software in further updates.

The prototype method is also suited for Q&A forums and sites such as ‘Quora’ for the previous reason that this site has a lot of users and they interact with it a lot which can be used to obtain feedback.

**What kind of project it is not suited for?**

One type of project that the prototype method is not suited for is a ‘mission critical’ project that comes with high risks such as controlling flights and dangerous areas such as power plants, this is because the prototype method usually implements its software first and then fixes tis issues after receiving feedback from the suer which cannot happen in key areas such as these and could have drastic consequences. This would be more suited to using the Spiral methodology.

**Conclusion**

Overall, the prototype method is best suited to projects that have an exceedingly high degree of user interaction. This is because it receives a lot of user feedback in its development and uses that to plan out its next steps when further refining the prototype of the program. It is not good for very important roles such as in ‘mission critical’ projects as it can have a lot of errors in the initial stages, even if they do get fixed later on it will cause a lot of problems at first which can have irreparable consequences.

The use of this method would not be a very good option for traditional industries as they do not interact with the clients much beyond the initial requirement analysis and the clients input and feedback is important in the software’s development.

Modern technology industries, on the other hand would be an excellent choice to use the prototype method for as they make their Software to cater to the user's needs and wants.

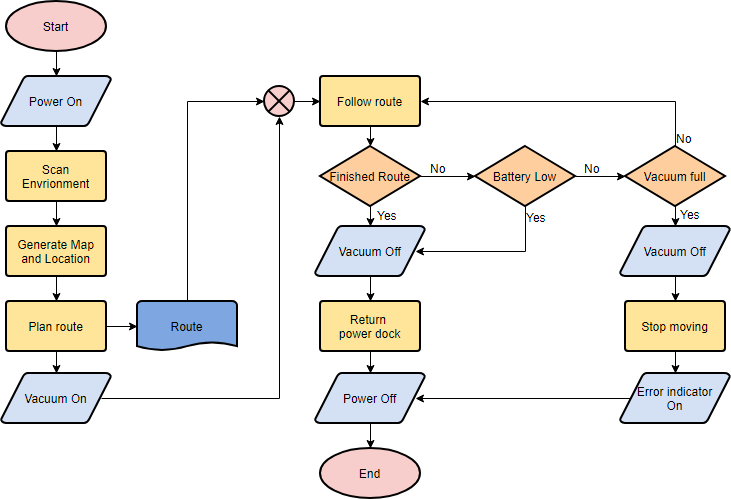
# Tools and Techniques

The general purpose of systems analysis tools and techniques is to study a system to obtain knowledge about it and to identify its objectives and the purpose of it.

The tools and techniques I am going to talk about in this section are: Flowcharts, Data flow diagrams (DFD), Context diagrams, Computer Aided Software Engineering (CASE Tools), Structured Systems Analysis and Design Method (SSADM) and Unified Modelling Language: Class Diagrams & Activity Diagrams.

**Flowcharts** – these are a type of diagram that are used to represent a process or a workflow in a step-by-step manner. Each step is shown as a different type of box. There are different boxes for inputs / outputs, processes, decisions and start / end commands.

They are used quite a lot in IT and computer science fields to design an algorithm for a software.



Flowcharts are very logically driven and are very easy to understand due to the clear and concise statements written inside of the symbols. The symbols follow a standard convention that all flowcharts do so it is easy to understand. It also uses some basic keywords and terminology that show up in other diagrams and coding languages, such as ‘read’ ‘write’ ‘input’ ‘output’ and ‘print’.

Flowcharts show their steps in differently shaped boxes with arrows called ‘connectors’ in between them to show the order that they are in.

**Suitability**

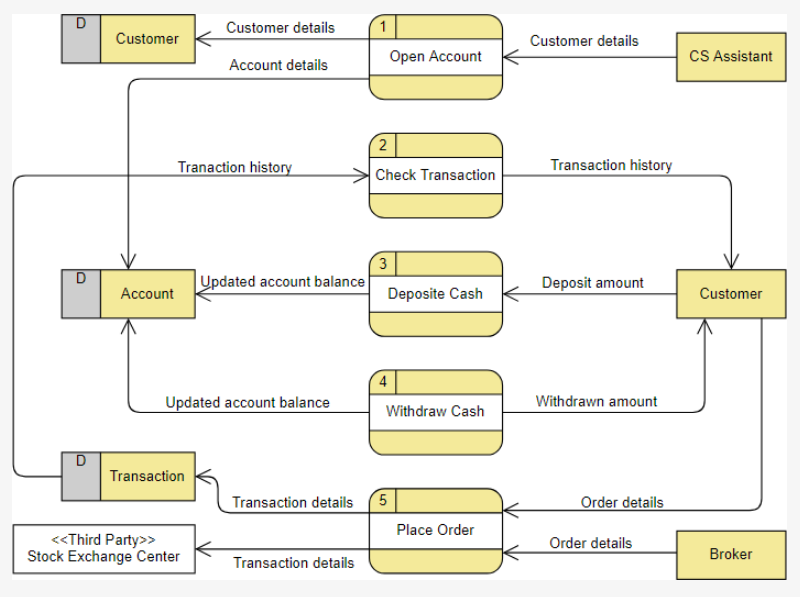
For the Waterfall method this is suitable as it can be used in the design stage to help you to properly understand how you will code the program. It is also very clear and logically driven which can make understanding it easy which fits with the waterfall method which is very rigid.

For the Spiral method this is suitable and can be implemented in the design phase to increase your understanding on how the program will work before you code it, the flowchart can be updated and enlarged in further iterations to make it more complex and add more features.

For the Prototype method this is not suitable as there is not much of a need to design the software since it works by creating a low-quality prototype. The design phase is not detailed at all in this and does not need a flowchart.

**Data flow diagrams** – these are a type of diagram that are used to represent the flow of data through a process or system, it provides information on the inputs and outputs of each entity with it and the processes themselves. There is no control flow, loops or decision rules to follow.

It includes inputs, outputs, data stores and various sub-processes that the data flows through.



It can isolate the collection of data (data stores) and identify data sources that are outside of process boundaries.

DFD’s can summarise the characteristics of the dataflow of a specific process on a single page, system-related processes are shown in this. This is called a ‘two-dimensional summary’.

They can also check how much of the model is complete and generate questions based on the missing data that is required by the system.

They are good at analysing and spotting patterns. such as spotting where lots of data is generated, collected, used, transferred, stored and delivered.

There are multiple levels to the DFD:

Level 0 – These are known as context diagrams; they provide a brief overview of the system but do not go into much detail. This includes a single node (such as the system) and its connections to external entities such as the user.

Level 1 – This is slightly more detailed than the previous level, in this level the single node is broken down into many subprocesses that need to be linked with data stores and flows.

Level 2+ – This breaks down the previous level into much more detail by adding more subprocesses.

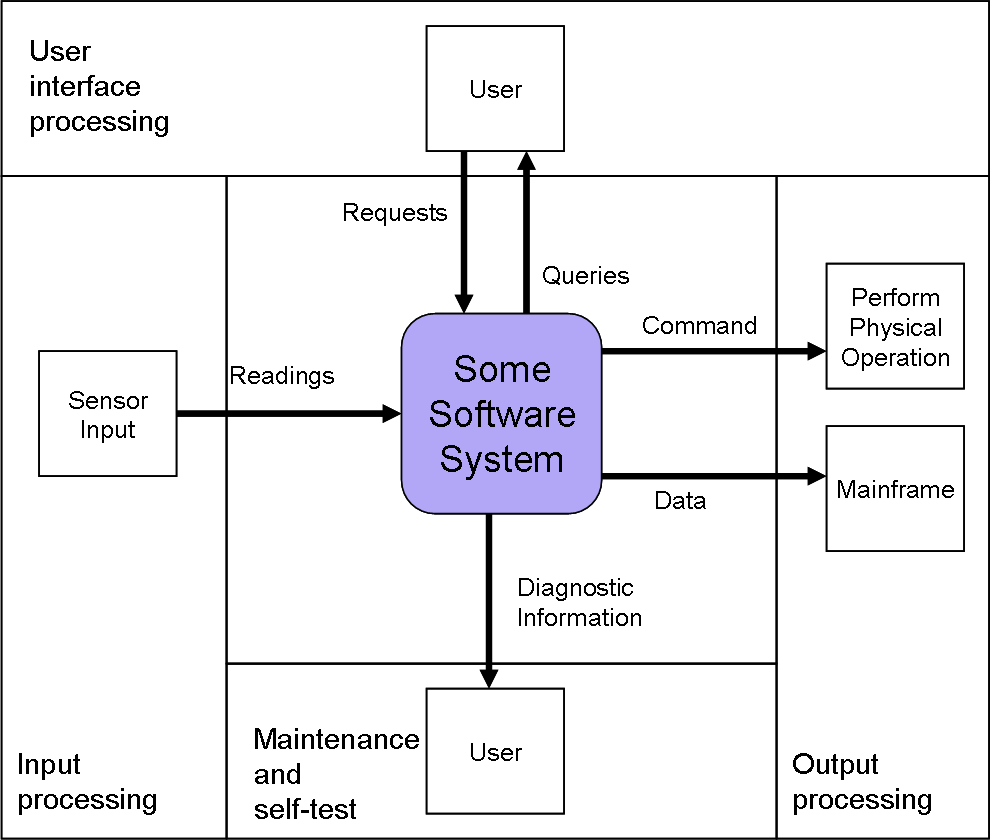
**Suitability**

For the Waterfall method this is not very suitable as it does not contain much detail about the requirements and properly define them, this method requires a comprehensive analysis of requirements because it cannot be added to later.

For the Spiral method this is not suitable as it can provide a small amount of information which can be used in the first iteration of development. It cannot provide any more information in further iterations which is why it is not suitable as it does not give the developer enough information to use as a basis to create their program.

For the Prototype method this is very suitable as although it is not very detailed it has enough to create the first low quality prototype that will be improved later on.

**Context diagrams** – As explained above, this is also the 0th level of the DFD, it is a brief overview of the system that does not go into much detail. It shows the flow of data between the system (node) and the external components such as the user, sensors, mainframe etc.



It is made up of a ‘context bubble’, which is a large circle in the middle of the diagram that represents a boundary that contains the processes and details of a project including its activities. These processes are hidden and only the name of the context bubble is shown.

Unlike other diagrams this is not for the use of engineers and software developers but for the use of the stakeholders. Due to its low level of detail, it will not confuse them with a bunch of technical jargon and would be understandable for them to know what the project will entail.

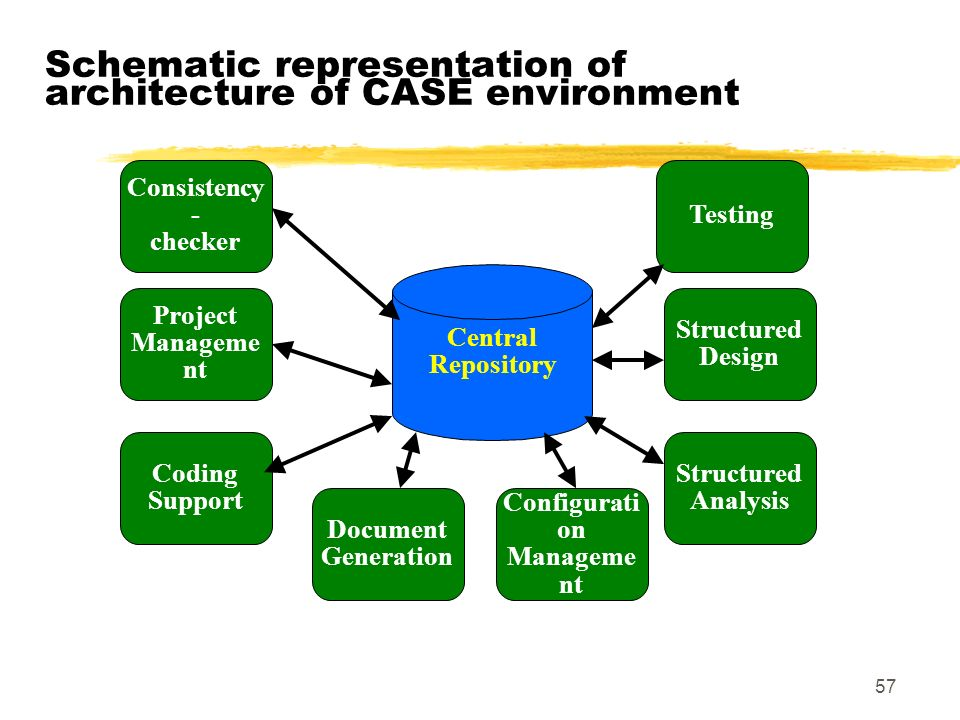
**Suitability**

For the Waterfall method this is not very suitable as it does not contain much detail about the requirements, as such it will not grant the required amount of data needed to prepare for the next phases of this method.

For the Spiral method this is not suitable as it does not provide much information, while it can be expanded upon in later iterations it is still inferior to more in-depth analysis techniques.

For the Prototype method this would be suitable as it does not require a high amount of analysis to take place and this would suffice, it would however be better to have slightly more information.

**Computer Aided Software Engineering** – this is a type of software that helps developers create new complex software. It is similar to CAD (Computer aided design) which designs objects on a computer to be manufactured.



CASE is a broad term to describe all methods of designing software on a computer including the aforementioned DFD and flowcharts.

It helps by automating some parts of the software development phase to save time and effort as well as eliminating the risk of human error thereby increasing the quality of the final product. It can also provide information on the software being developed to help you in the non-automated parts of the development phase.

It makes use of upper CASE and lower CASE tools. Upper CASE tools are used for front end development such as requirement analysis and designing. Lower CASE tools are used for back-end development such as the actual software development phase, testing, debugging and maintenance.

The more common tools that you see are called integrated CASE tools (I-CASE). These tools offer support for the full product life cycle, both back end and front end. They share information from a common area, changes in information from the design documents (design brief) will be reflected in many other parts of the system.

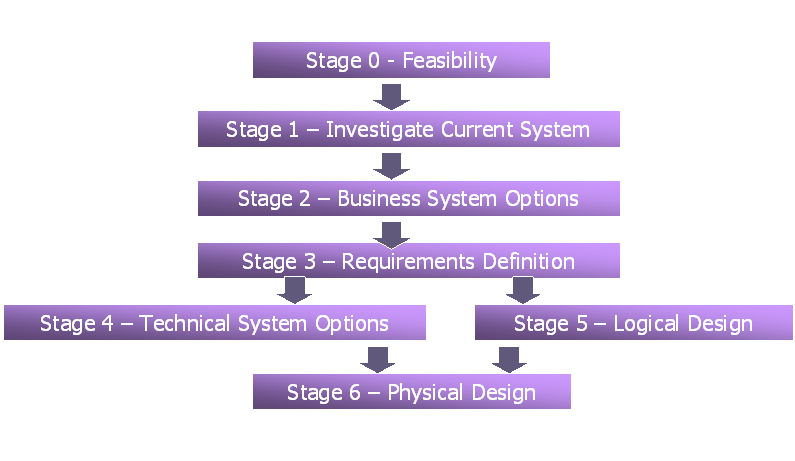
**Suitability**

For the Waterfall method this is suitable as it can sue the computer to generate questions that will help you research the requirements needed for the software, as these requirements need to be highly detailed for this methodology this is a very good choice.

For the Spiral method this will be suitable as it provides a lot of information, due to this it will remove the need to add to the design stage in further iterations and will save you time and effort.

For the Prototype method this is suitable as it can provide more than enough information for the first prototype and will cut down development time significantly as there will be less iterations of developing new prototypes since the first one will be of a good quality.

**Structured Systems Analysis and Design Method** – SSADM is a set of standards for software design based on the waterfall method. It is said to be the best document driven method of system designing.



There are 3 techniques that come with SSADM:

Logical data modelling – this is where you identify, model and document the requirements of the system. The data model will include entities, attributes and relationships.

Data flow modelling – this is where you will identify, model and document how data flows around in the system. It processes data store, data flows and external entities (similar to a DFD).

Entity event modelling – this process can be broken down into 2 separate stages. Entity behaviour modelling and event modelling. Entity behaviour modelling is where you identify, model and document the events that affect each entity and the history of when this happened. Event modelling is where you design the process to coordinate entity life histories for each event that you have documented in the previous stage.

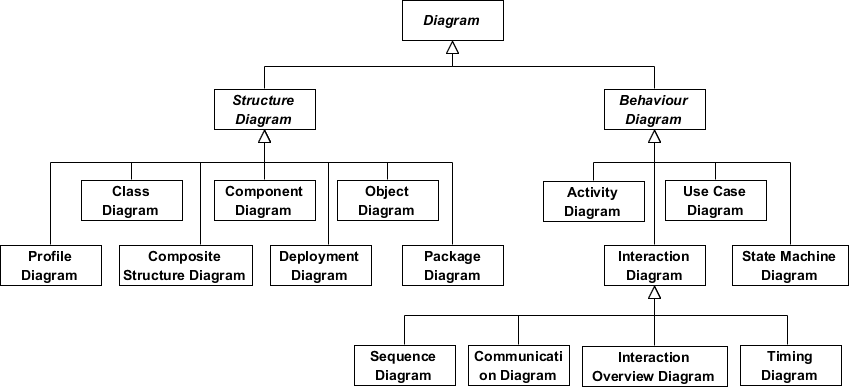
**Suitability**

For the Waterfall method this is very suitable as it was based on the waterfall method and derived its rules from it, due to this it will be very compatible with it.

For the Spiral method this is very suitable as it provides a very detailed software design, this will be helpful later on as you will not need to spend time on the design stage as it is already complete and can focus on the development of the software.

For the Prototype method this is not suitable as it is far too detailed and is overkill. There is not a high requirement for detail in the requirement analysis and design of the prototype model so this may end up being a waste of time.

**Unified Modelling Language** – UML is a modelling language for software that lets you visualise the design of a system before developing it.



There are many types of UML diagrams but they can be split into 2 sections, Structural UML diagrams and Behavioural UML diagrams.

Class Diagrams: This is a structural UML diagram that describes the static structure of a system by showing relationships between classes, objects, attributes, and operations.

Activity Diagrams: This is a Behavioural UML diagram that shows the dynamic nature of a system by modelling the flow of control in between different activities (similar to a flowchart). Each activity is something that results in a change in the state of the system.

**Suitability**

For the Waterfall method this is not suitable as it is not detailed enough and will not be suitable as it cannot be appended to at a later time due to the sequential manner in which the phases of this method are completed.

For the Spiral method this is suitable as although it contains a low amount of data it can be appended to in further iterations and made more complex.

Hi

For the Prototype method this is suitable as it is not detailed and is very brief which can be used to create an initial prototype to work with.

**Programming paradigms**

**Procedural programming –** this is a step-by-step approach to completing tasks in a logical manner. It divides the program into procedures such as functions and routines to be executed.

Flowcharts: These are very suitable for this paradigm as they follow the same sequential method of completing tasks, it also makes use of subroutines.

DFD: this is suitable as it can show inputs/outputs as well as subroutines and is very easy to understand.

Context diagram: this is suitable as it can provide the minimum amount of data required for this paradigm, but it would be better to use one that would provide more information to use in the development of the software.

CASE diagram: this is suitable as it can divide the planning and development phases which allows you to manage your time more efficiently and develop a higher quality software. However, it will be better if the program was larger as procedural programming is generally small-scale and over complicated due to the many requirements that they have. It is also designed for more complex programs so it may not fit procedural programming in this regard.

SSADM: as this was based off of the waterfall method which is very straightforward and rigid in its execution it will also be suitable for procedural programming as it also completes tasks in a linear and sequential fashion.

UML: UML is suitable for procedural programming as it can accurately display all of the aspects of it, the fact that procedural programming is very simple and straightforward in its methods makes it a lot easier to display than some other paradigms.

**Event-driven programming –** this is a programming paradigm where the program is driven by events such as inputs and outputs. They use a lot of if statements and conditions that are activated by certain events.

Flowcharts: These are very suitable for this paradigm as there are dedicated symbols for inputs/outputs and each possible event can be portrayed in a flowchart without problem.

DFD: this is not suitable as while it can easily display all of the events it cannot properly show how conditions work and only shows the flow of data in the background.

Context diagram: this is not suitable because it does not have enough detail to properly give the requirements for a detailed software.

CASE diagram: this is suitable as it can help make complex programs without sacrificing the overall quality of them, however it is suited to more general paradigm such as procedural programming due to them being less specialised.

SSADM: this is very suitable for even driven programming as it has a sub-technique called ‘entity event modelling’ which is based around identifying changes to event and logging when they happen.

UML: activity diagrams are very suitable for this paradigm as they can show the flow of data and can properly represent all functions of it.

**Object-oriented programming –** this is a programming paradigm based on objects (anything that contains data, usually a variable) in the form of procedures (subroutines), procedures within an object can modify the value of the objects themselves. It is coded by making different objects interact with each other.

Flowcharts: These are suitable for this paradigm and can show all of its functions but there are many better options such as activity diagrams.

DFD: this is suitable as it can show the flow of data between all objects and can show when the data is changed such as the value of a variable being overwritten.

Context diagram: this is not suitable because it does not have enough detail to properly give the requirements for a detailed software.

CASE diagram: it is suited as it is designed for more complex programs such as object-oriented ones. It will help to make sure that the programming stage does not get too confusing and can be easily understood and followed.

SSADM: this is suitable as it has a sub-technique called data flow modelling that can keep track of the flow of data throughout the program which is similar to a DFD and is a good choice as all the objects in this paradigm are made up of data, keeping track of data is important as it will help you know about any changes in the software.

UML: class diagrams are very suitable for this paradigm as they directly deal with objects and classes, as this paradigm is centred around objects this technique is an obvious choice.