Revision topics for the Analysis and Design Unit, Level 8, part 1

Outcome 1: Describe the use of analysis and design in the software development process

SDLC stands for Software Development Life Cycle. SDLC is the process consisting of a series of planned activities to develop or alter the software products

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements. The testing team follows Software Testing Life Cycle (STLC) which is similar to the development cycle followed by the development team.

There are following six phases in every Software development life cycle model:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

1) Requirement gathering and analysis: Business requirements are gathered in this phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system? What data should be input into the system? What data should be output by the system? These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analysed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.

Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model. The testing team follows the Software Testing Life Cycle and starts the Test Planning phase after the requirements analysis is completed.

2) Design: In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

In this phase the testers comes up with the Test strategy, where they mention what to test, how to test.

1.1 - Describe conventional and contemporary approaches to software development

Traditional methodologies are characterised by a sequential series of steps like requirement definition, planning, building, testing and deployment. First, the client requirements are carefully documented to the fullest extent. Then, the general architecture of the software is visualised and the actual coding commences. Then comes the various types of testing and the final deployment. The basic idea here is the detailed visualisation of the finished project before the building starts, and working one's way through to the visualised finished structure.

Agile method of developing software is a lot less rigorous than the former. The crux here is incremental and iterative development, where phases of the process are revisited time and again. Agile developers recognise that software is not a large block of a structure, but an incredibly organic entity with complex moving parts interacting with each other. Thus, they give more importance to adaptability and constant compatibility testing.

1.2 - Describe analysis and design tools and models

Describe major Analysis and Design tools and models, including:
User Interface Design
Entity Relationship Model
Data Design
Data Dictionary

Data Dictionary -
A document that outlines your table designs, the data type for each column and a brief explanation of each field.

Entity Relationship -
A diagram showing objects and pieces of data and how they are linked

User Interface -
Making the user's interactions as simple and efficient as possible

Data Design -
Selecting logical representations of data objects identified during the requirements definition and specification phase
[Software Development Life Cycle](http://www.tutorialspoint.com/sdlc/index.htm)

User Interface
(UI) Design focuses on anticipating what users might need to do and ensuring that the interface has elements that are easy to access, understand, and use to facilitate those actions. UI brings together concepts from interaction design, visual design, and information architecture.

Choosing Interface Elements
Users have become familiar with interface elements acting in a certain way, so try to be consistent and predictable in your choices and their layout. Doing so will help with task completion, efficiency, and satisfaction.

Interface elements include but are not limited to:

Input Controls: buttons, text fields, checkboxes, radio buttons, dropdown lists, list boxes, toggles, date field
Navigational Components: breadcrumb, slider, search field, pagination, slider, tags, icons
Informational Components: tooltips, icons, progress bar, notifications, message boxes, modal windows
Containers: accordion

Describe the approaches to software development including: advantages and disadvantages; phases and iteration in the development process; and factors affecting your choice of this approach.

1.3 - Describe the Waterfall Development approach

The best-known and oldest process is the waterfall model, where developers follow these steps in order. They state requirements, analyse them, design a solution approach, architect a software framework for that solution, develop code, test, deploy, and maintain. After each step is finished, the process proceeds to the next step.

Describe the Waterfall Development approach including the production of functional and non-functional requirements specifications.

- Requirements • Design • Implementation • Verification • Maintenance

Requirement Gathering and analysis: All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.

System Design: The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.

Implementation: With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.

Integration and Testing: All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system: Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.

Maintenance: There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Advantages of waterfall model:

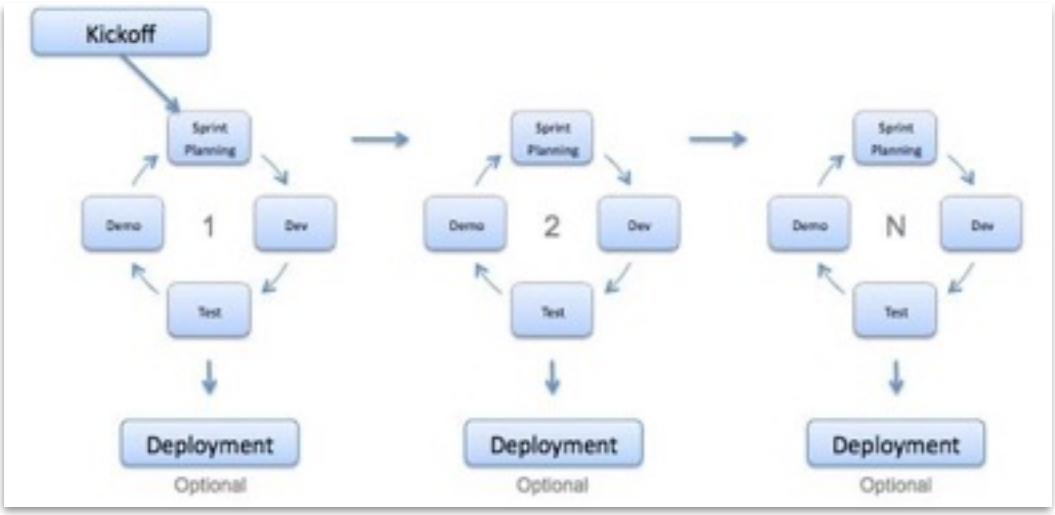
- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

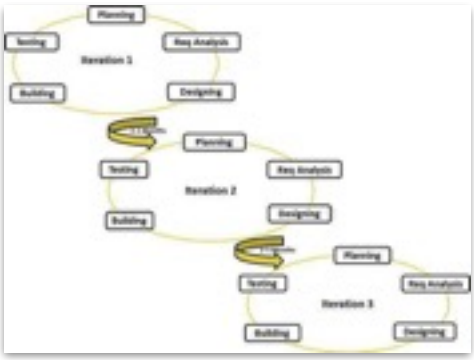
Disadvantages of waterfall model:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

1.4 - Describe the Agile Development approach

Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications. Extreme Programming (XP) is currently one of the most well known agile development life cycle model.





pros:
- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasised rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

cons:
- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

Select an Agile approach such as Scrum or Kanban, produce an Agile Product Roadmap, use User Stories to capture requirements, produce a Product Backlog, manage and track progress, present results, seek feedback and reflect on future improvements.

….

Whats the difference between a product roadmap and a product backlog?

A backlog is a list of features or technical tasks which the team maintains and which, at a given moment, are known to be necessary and sufficient to complete a project or a release:

- if an item on the backlog does not contribute to the project's goal, it should be removed;
- on the other hand, if at any time a task or feature becomes known that is considered necessary to the project, it should be added to the backlog.

These "necessary and sufficient" properties are assessed relative to the team's state of knowledge at a particular moment; the backlog is expected to change throughout the project's duration as the team gains knowledge.

The backlog is the primary point of entry for knowledge about requirements, and the single authoritative source defining the work to be done.

Describe Software Prototyping

The Software Prototyping refers to building software application prototypes which display the functionality of the product under development but may not actually hold the exact logic of the original software.

Software prototyping is becoming very popular as a software development model, as it enables to understand customer requirements at an early stage of development. It helps get valuable feedback from the customer and helps software designers and developers understand about what exactly is expected from the product under development.

• Prototype is a working model of software with some limited functionality.

• The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation.

• Prototyping is used to allow the users evaluate developer proposals and try them out before implementation.

• It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely:

Throwaway/Rapid Prototyping: Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

Evolutionary Prototyping: Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. Using evolutionary prototyping only well understood requirements are included in the prototype and the requirements are added as and when they are understood.

Incremental Prototyping: Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.

Extreme Prototyping : Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

pros
- Increased user involvement in the product even before implementation
- Since a working model of the system is displayed, the users get a better understanding of the system being developed.
- Reduces time and cost as the defects can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified

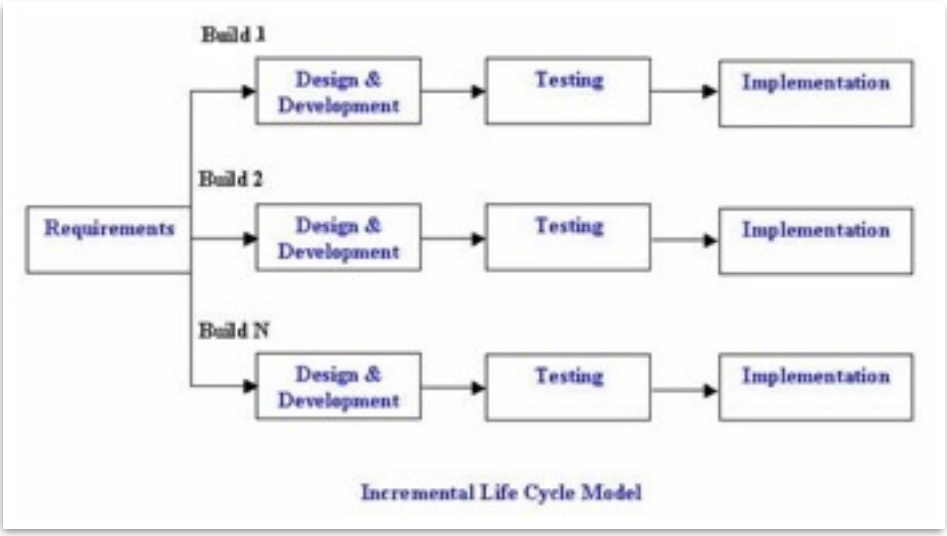
cons
- Risk of insufficient requirement analysis owing to too much dependency on prototype
- Users may get confused in the prototypes and actual systems.
- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans.

Incremental/ Iterative model

In incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a “multi-waterfall” cycle. Cycles are divided up into smaller, more easily managed modules. Incremental model is a type of software development model like V-model, Agile model etc.

In this model, each module passes through the requirements, design, implementation and testing phases. A working version of software is produced during the first module, so you have working software early on during the software life cycle. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.



Incremental Life Cycle Model

Advantages

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it’d iteration.

Disadvantages

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

Spiral

Spiral model has four phases
• Identification
- Design
- Construct or Build
- Evaluation and Risk Analysis

Identification
This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

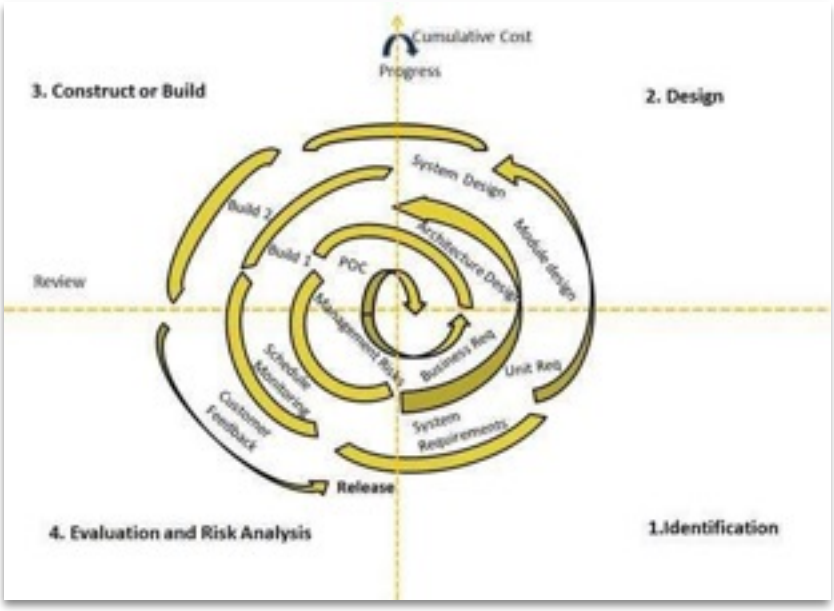
Design
Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

Construct of Build
Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Evaluation and Risk Analysis
Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The Pros
- High amount of risk analysis hence, avoidance of Risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.

Cons
- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.



RAD

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

Business Modelling: The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

Data Modelling: The information gathered in the Business Modelling phase is reviewed and analysed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

Process Modelling: The data object sets defined in the Data Modelling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding , deleting, retrieving or modifying a data object are given.

Application Generation: The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

Testing and Turnover: The overall testing time is reduced in RAD model as the prototypes are independently tested during every iteration. However the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

pros

- Reduced development time.
- Increases reusability of components
- Quick initial reviews occur
- Encourages customer feedback
- Integration from very beginning solves a lot of integration issues.

cons

- Depends on strong team and individual performances for identifying business requirements.
- Only system that can be modularised can be built using RAD
- Requires highly skilled developers/designers.
- High dependency on modelling skills
- Inapplicable to cheaper projects as cost of modelling and automated code generation is very high.

Entity Relationship

An entity relationship model, also called an entity-relationship (ER) diagram, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organisation of data within databases or information systems. An entity is a piece of data-an object or concept about which data is stored.

one to one
one to many
many to many

[Data Design slides 4+8](http://www.peter-lo.com/Teaching/CS213/L04.pdf)

[Data Dictionary](http://www.cs.unb.ca/profs/fritz/cs3503/dd.htm) and [Data Dictionary](https://blog.udemy.com/data-dictionary-example)

Identify the characteristics of good program design

The six main quality characteristics (according to an international standard for evaluating software quality) are:

Functionality

the degree of which the software fulfils its purpose without waste of resources

Reliability

The frequency and criticality of software failure, where failure is unacceptable effect of behaviour

Usability

how easy the program is for clients and customers to understand and learn

Efficiency

the suitability of software for serving its intended purpose

Maintainability

The ease with which changes can be made to satisfy new requirements or to correct deficiencies

Portability

Good practice in program design refers to the ease with which software can be used on computer configurations

Data Dictionary -

A document that outlines your table designs, the data type for each column and a brief explanation of each field.

Entity Relationship -

A diagram showing objects and pieces of data and how they are linked

User Interface -

Making the user's interactions as simple and efficient as possible

Data Design -

Selecting logical representations of data objects identified during the requirements definition and specification phase
[Software Development Life Cycle](http://www.tutorialspoint.com/sdlc/index.htm)

Implementation and Unit Testing

1.1 - Describe structured programming constructs

Programming constructs: Expression, sequence, selection, iteration, predefined function, file handling.

Expression
in a programming language is a combination of one or more explicit values, constants, variables, operators, and functions that the programming language interprets (according to its particular rules of precedence and of association) and computes to produce ("to return", in a stateful environment) another value. This process, as for mathematical expressions, is called evaluation.

In simple settings, the resulting value is usually one of various primitive types, such as numerical, string, and logical; in more elaborate settings, it can be an arbitrary complex data type. In functional programming, the resulting values are often functions or expressions, which can themselves be further evaluated.

Sequence
is a set of instructions in order, one instruction after the other, once one instruction has been executed, it will take a few seconds for it to be executed. These instructions are made by what are “programmers.”

Selection
is one of the main three basic structures. A selection is when a question is asked and depending on the answer, there may be more than one outcome. To put this in a more simpler way to understand selection is also known as an **IF STATEMENT.**

Iteration
An iteration is a computer **LOOP** that goes through a group of instructions which gets executed over and over again. Iteration is also one of the main three basic programming constructs. Iteration involves the use of two variables which are called the **while** and the do while.

Predefined Functions
A predefined function is a set of routines that the computer carries out to perform functions in a programming language.

An example of a predefined function would be: System.out.println("This is my work");
System.out.println allows to the computer to print what ever words you put in the speech marks on to the screen. However, you must remember to put a semi colon at the end or it will show as an error.
Another example of a predefined function would be concat() this lets you combine the text of two strings and returns to a new string.

public String concat(String str) would be an example of two strings.

File Handling
Another use of files is reading data into your program, think about when you load a saved game, or open a spreadsheet. Both of these things involve reading a file.

Files might have lots of strange file extensions like .doc .txt .html .css .odt. In fact there are thousands of them and they tell the Operating System what program to use to read the file. However, if you open each file with a text editor such as notepad++ you might be able to see underlying structure of the file. Some of them will be plain text and some of them gobbledygook. We are going to learn how to read and write data directly to these files.

Reading files A common function needed in programs is to load data: saved games, text documents, spreadsheets. To do this you'll need the StreamReader, a collection of functions used to read data from files into our program. To get this program to work you are going to need a file called myfile.txt and put it into a location you know the full address of, for example the root of the C:\ drive:

1.2 - Describe simple data types, data structures and algorithms

Data types: String, integer, float, boolean

Data structures: Array, hash, etc

String consists of one or more characters. String represents **ALPHANUMERIC DATA**
this means a string can contain many different characters but they are all considered as if they were text even if the characters are numbers.

numeric data - **integer** which is a numeric value without a decimal **whole numbers** that can be positive and negative. Short integer is 16 bits Long integer 32 bits. Number with a decimal is referred to as a **decimal, float and double.**
Boolean is primary result of conditional statement which controls work flow in program

Symbols
Symbols are light-weight strings. They're often used as identifiers, in places other languages would often use strings. They're used instead of strings because they can take up much less memory (it gets complicated, so I'm trying to keep it simple; if you want to read more check out this article).

To create a symbol, simply precede a word with a colon; you can see a few examples of this in the code snippets above.

Nil is Ruby's "nothing" value, although it is an object.

Simple algorithms: such as algorithms for linear search, input validation, find maximum, find minimum, and count occurrences.
a set of steps to a computer program accomplish a task
Audio and video Algorithms
A route finding Algorithms
Rendering Algorithms
Optimisation and scheduling Algorithms
minimax search Algorithms - for games
What makes a good algorithm - two most important criteria that is solves a problem and it does so **efficiently**.
Give us an answer that is always correct. Measuring efficiency - scientists use an **asymptotic analysis** which allows algorithms to be compared independently of a particular programming language or hardware that say yes some algorithms are more efficient than others.

Algorithm question example

The algorithm questions will present some lines of Ruby code and ask you to answer the following: “State the purpose of this algorithm, describe what it does and what will be returned when it is run”

```
def bubble(array)
  n = array.length
  loop do
    swapped = false

    (n-1).times do |i|
      if array[i] > array[i+1]
        array[i], array[i+1] = array[i+1], array[i]
        swapped = true
      end
    end

    break if not swapped
  end

  array
end

bubble([1,5,2,8,4,6])
```

State the purpose of this algorithm:

This is a bubble sort algorithm that will put the items of an array into ascending order.

Describe what it does:

The bubble function takes in an array. The length of the array is assigned to the variable ‘n’, in this case ‘6’.

In the loop the variable ‘swapped’ is set to false. There is then another loop that runs five times (the length of the array -1) with a counter variable ‘i’ to go through the items of the array.

If the item in the first position of the array is greater than the item in the next position then they are swapped and the swapped variable is set to true. This keeps looping for each number, moving it up until it isn’t swapped. The loop then repeats with the next item of the array.

What will be returned when it is run:

The array returned will be [1,2,4,5,6,8]

1.3 - Describe basic software testing methods

Software testing methods: Static testing, Dynamic testing, White box testing, Black box testing

Static Testing
Static testing is a software testing method that involves examination of the program's code and its associated documentation but does not require the program be executed. Dynamic testing, the other main category of software testing methods, involves interaction with the program while it runs. The two methods are frequently used together to try to ensure the functionality of a program.

Static testing may be conducted manually or through the use of various software testing tools. Specific types of static software testing include code analysis, inspection, code reviews and walkthroughs.

Dynamic Testing
Dynamic testing is a method of assessing the feasibility of a software program by giving input and examining output (I/O). The dynamic method requires that the code be compiled and run. The alternative method of software testing, static testing, does not involve program execution but an examination of the code and associated documents.

Types of dynamic testing include unit testing, integration testing, system testing and acceptance testing.

Black-box Testing - IMPORTANT
Black-box testing (also known as functional testing) treats software under test as a black-box without knowing its internals. Tests are using software interfaces and trying to ensure that they work as expected. As long as functionality of interfaces remains unchanged, tests should pass even if internals are changed.

Some of the advantages of black-box testing are:
- Efficient for large segments of code
- Code access is not required
- Separation between user's and developer's perspectives

Some of the disadvantages of black-box testing are:
- Limited coverage since only a fraction of test scenarios is performed
- Inefficient testing due to tester's luck of knowledge about software internals
- Blind coverage since tester has limited knowledge about the application

White-box Testing - IMPORTANT
White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) looks inside the software that is being tested and uses that knowledge as part of the testing process. If, for example, exception is thrown under certain conditions, test might want to reproduce those conditions. White-box testing requires internal knowledge of the system and programming skills. It provides internal perspective of the software under test.

Some of the advantages of white-box testing are:
- Efficient in finding errors and problems
- Required knowledge of internals of the software under test is beneficial for thorough testing
- Allows finding hidden errors
- Programmers introspection
- Helps optimising the code
- Due to required internal knowledge of the software, maximum coverage is obtained

Some of the disadvantages of white-box testing are:
- Might not find unimplemented or missing features
- Requires high level knowledge of internals of the software under test
- Requires code access

Conclusion
Both white and black-box testing are necessary for the successful software delivery. In many cases black-box testing is done by dedicated testers while white-box testing is performed by developers. Emergence of TDD, ATDD and BDD processes and supporting tools allows early defects detection and shifts the focus from QC towards QA.

Software testing levels - IMPORTANT

There are four levels of Software Testing - Unit >> Integration >> System >> Acceptance
(unit testing, integration testing, component testing, system testing)

Unit Testing - A level of the software testing process where individual units/components of a software/ system are tested. The purpose is to validate that each unit of the software performs as designed.

Integration Testing - A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

System Testing - A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.

Acceptance Testing - A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

Component Testing

Component testing is that in which we test those test objects which are separately testable as a isolated unit without integrating with other components (e.g. modules, programs, objects, classes, etc.).

Testing and documenting solutions - IMPORTANT

Testing

To thoroughly test a program, you should test it using normal, extreme and exceptional data. The data that falls into each of these categories depends on what your program is designed to do.
For example, if you designed a program to process students' test scores out of 50, then normal, extreme and exceptional data might be as follows:

Test Case	Explanation	Example where a score should be between 0 and 50
Normal	Data that you would expect to work or be accepted and that lies within the range	2, 45
Extreme	Data at the lower and upper limits of the range	0, 50
Exceptional	Data that should not be accepted by the program	-7, Yaney

When testing programs, it is good practice to set up a test plan where you plan to test at least two cases of data from each category.

It is also important to know that extreme test data is NOT boundary testing. Sometimes extreme data is referred to as 'boundary testing' but this is a little inaccurate. In the above example, boundary testing would test the extreme data of 0 and 50 but would also include -1 and 51.

Extreme data is only concerned with the lower and upper values in a range, in this case 0 and 50.

Identifiers, indentation, providing internal commentary

How readable is your code?

Often programmers will work on each other's code. So you need to write your code so that it can be read and understood by others. You also need to be able to understand it if you come back to it months or years later. There are a few techniques you can use to help with this:

- Internal commentary
- Meaningful identifiers (variable names)
- Indentation
- White Space

Internal commentary lets a programmer add comments to their code that will not be translated when the program runs. These comments can be used to add descriptions, notes or explanations that can be read by anyone with access to the code. This is useful when working as part of a team as other programmers can get an idea of what your code is doing and why you made certain decisions.

Meaningful identifiers
Making sure you use variable names that describe what the variable contains is very important. Poor variable names can make it difficult to work out what is going on in your program.
For example, the use of the word ‘data’ is not meaningful:
DECLARE data AS STRING INITIALLY “ “

Indentation
Indentation means moving parts of your code that form the content of constructs such as loops or if statements to the right so it's easier to see the overall structure of the code. Using indentation can make your code much easier to read.

White Space
White space is similar to indentation in that is it is used to help make it clear where code is placed. However, white space is used to separate different subprograms and functions so that it is easier to see each module (section) of code. The listing shown below contains no white space to separate each section of code.

Programming paradigms

Two of the most important programming paradigms are **procedural programming** and **object-oriented programming**

Procedural Programming

list of instructions to tell the computer what to do step by step. (routines and sub routines) also referred to imperative programming.

Event Driven Programming

Event-driven programming is a paradigm used to structure a program around various events. These events include user input events in graphical user interfaces and networking requests from websites and other online properties.

The creation of graphical interfaces and the windows paradigm, however, forced developers to rethink their earlier strategies. Because users can click virtually anywhere, the program must be read for nearly any input. In addition, information isn't always entered in the same manner; event-driven programming makes it easier to handle a diverse range of inputs.

Object-Oriented Programming

approach to problem solving where all computations are carried out using objects. An object is a component of a program that knowns how to perform certain actions and how to interact with other elements of the program.

Ternary Operator

The ternary (or conditional) operator will evaluate an expression and return one value if it's true, and another value if it's false. It's a bit like a shorthand, compact if statement.

Ternary Operator Example

Let's look at this example:

```
def find_largest_num(num1, num2)

  num1 > num2 ? num1 : num2 <— —short hand for the if statement in blue

  if ((num1 > num2) == true)
    return num1
  else
    return num2
  end

end

puts find_largest_num(300, 100)
```

Here, the conditional operator is being used to select between two strings. The entire operator expression is everything including the conditional, question mark, two strings and the colon. The general format of this expression is as follows: conditional ? true : false.

If the conditional expression is true, then the operator will evaluate as the true expression, otherwise it will evaluate as the false expression. In this example, it's in parentheses, so it doesn't interfere with the string concatenation operators surrounding it.

Array and empty hash

So for example input of ["fish", "chips", "chips"] would output {"fish": 1, "chips": 2}

```
def word_count

  word_list = ["I", "am", "a", "list", "a", "list", "list", "of", "words"]

  word_counts = Hash.new(0)

  word_list.each do |word|
    word_counts[word] += 1
  end

  puts word_counts
end

word_count
```

so returns
I : 1
am : 1
a : 2
list : 3
of : 1
words : 1

Example of Syntax Error

unexpected keyword
SyntaxError: (eval):1: syntax error, unexpected '=', expecting $end

Example of Execution Error

could not add variables string, hash, fixnum etc..
The Exception class handles nearly every kind of hiccup that might occur during runtime, including syntax screwup and incorrect type handling. - does not crash