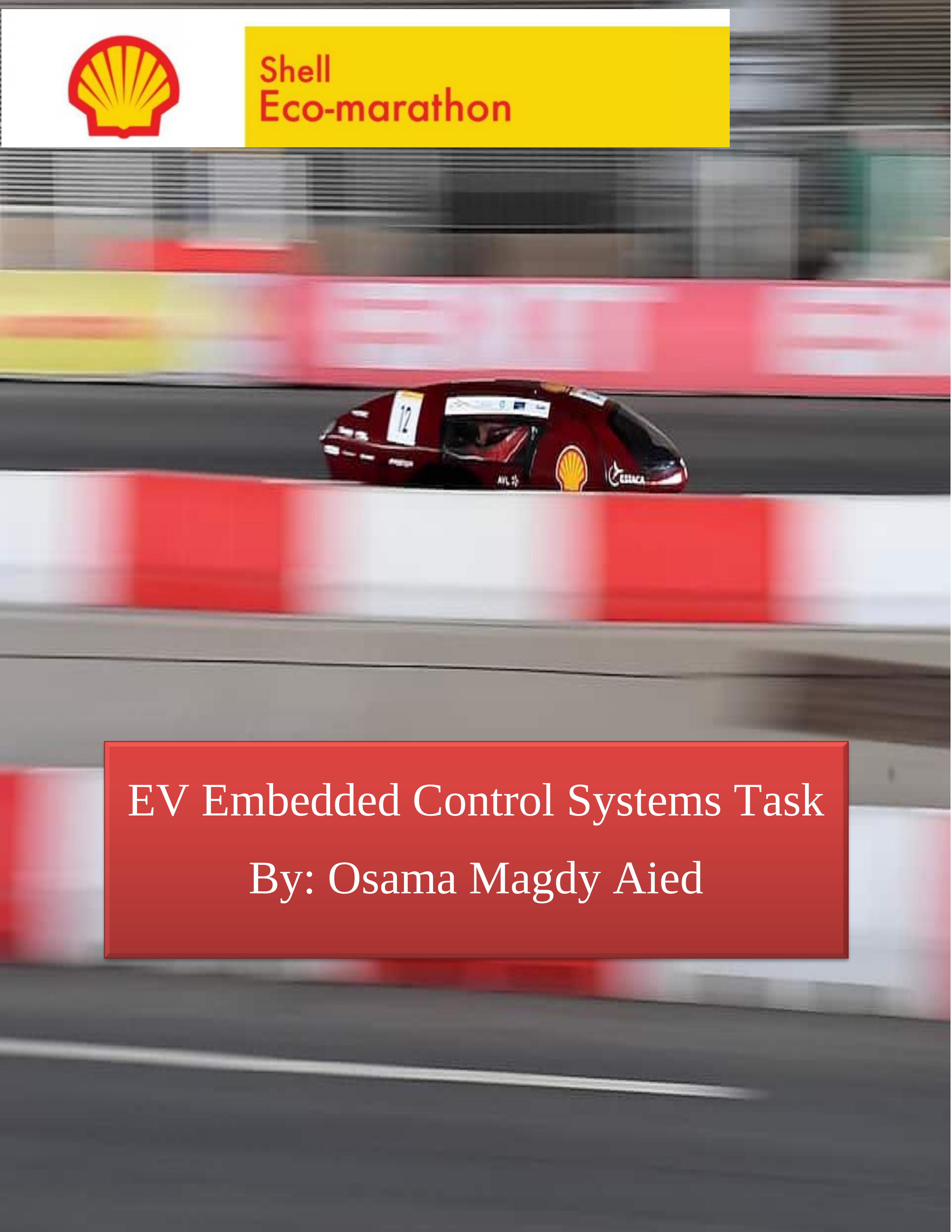


EV Embedded Control Systems Task

By: Osama Magdy Aied

Contents

Question 1:

Github repo: https://github.com/osamamagdy/PWM-Task

Question 2:

a) Direct memory access:

It is used when we need to transfer data from the memory to the I/O ports without using the CPU first. The normal process of accessing data memory is that we ask the CPU to get that data and give it back to the port, but there is a DMA controller that we can set to make the ports get the needed data directly from the memory. This process takes only one clock cycle instead of two (by using the CPU).

b) ADC comparison:

The Analog-to-Digital Converter is mainly using a voltage comparator to read the input values. So, it will be affected by any change in the voltage coming to it even if the signal itself remains the same. In this question we're using voltage divider in both cases which means that the voltage of the signal will not directly go to the ADC but it will be divided between the resistor and the ADC. The percentage of the voltage entering each one will be changed according to the resistor value (as the ADC resistance is the same).

c) Keywords in C:

I. Volatile:

We mainly are using “volatile” variables whenever their values can be changed unexpectedly. This word is written in the variable definition to avoid side effects of the compiler optimization, in many applications we assign a value to a variable only one time but keep reading it every time. So, the compiler simply optimizes this code and stops reading the value every time and stores the original value we used in the first place. This is not good in the embedded world as there are many variables that could change after the programming process like variables accessed by interrupt service routine or RTOS applications. The word volatile prevents this type of optimization.

II. Static:

A “static” key word can be used before functions and variables. With variables declared inside any function, it means that the value assigned in declaration is executed one time no matter how many times the function is called. Example:

```
int counter_statice(void)
  {
    static int count1 = 0 ;
    count1++;
    return count1;
 }


int counter(void)
 {
    int count 2= 0 ;
    count2++;
    return count2;
 }
```

Now, if we run a code that calls both functions twice, the return value of the static variable count1 will be 1 then 2 and the non-static variable count2 will be 1 then also 1 because it’s given 0 every time we call it. Using static with function declared inside specific file means that it can’t be used (called) in any other files even when we use #include “file_name.h”. For the previous example, if we write “static” before the function return type “int” it means that we can’t call it anywhere outside this file.

III. Extern:

The “extern” keyword can also be used with functions and variables. With functions it’s the opposite of the “static” functions. As the static functions can’t be used outside the file it’s written in it, the “extern” function extends the use of it to the whole program (of course you have to include the file where the function is written first) and functions are by default “extern” unless you make them static. With variables we use it to declare the existence of a variable but it’s not defined or allocated in the memory yet.

```c
extern int x;
int main(void)
{
x = 10;
return 0;
}
// This will give us an error
```

So, we can’t use it or give it any values unless we define it first and values are not “extern” by default.

d) Baud rate: $\frac{1}{bit\ period}$

bit period = 1 ms //the time for every bit change

Baud rate = 1000

The number:

The 8-bit number is being transmitted between the 0 start bit and the 1 stop bit (note that the LSB is being transmitted first). So, the number is:

In binary: 0b10110010

In hexadecimal: 0xB2
In decimal: 178

e) What do you know about:

1) Vector table:

By enabling the interrupt of a certain interrupt source, we want to tell the CPU what to do in case that interrupt takes place. This is what we call Interrupt Service Routine (ISR), it contains the instructions that are executed when a certain interrupt flag is raised. But this service routine can be of different sizes and can be stored in different memory locations. So, we use vector tables to store the address of each service routine function located in the memory. Those addresses are stored in program address of the vector tables. When interrupt occurs, the CPU stores the last address from the program counter in the stack and goes to the vector table to see what program address it should access depending on the interrupt source. After that we see the address stored in the program address that will take us to the service routine function. The CPU executes this function and then returns back to the address stored in stack and puts it in the program counter to continue working.

2) Startup code:

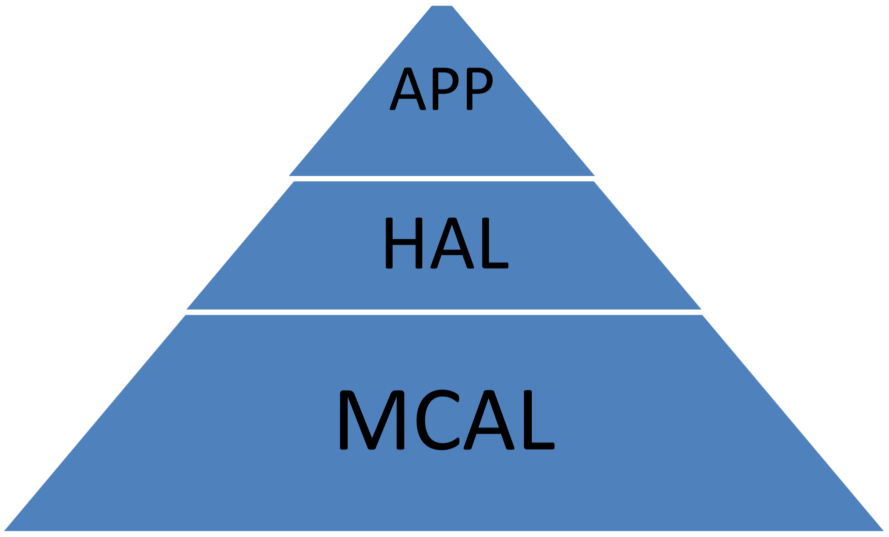
Startup code is the program code that is executed after reset is enabled or at connecting the power to the MCU. This code includes four main operations:
1 - Reset Vector (sets all registers to initial values)
2 – Setup the bus configuration registers
3 - Clearing the memory
4 – Initializing global variables

3) Bootloader:

The Bootloader is the first code to be executed when we program our microcontroller or power it up. It contains code that uses the Hex file to download it into memory of the MCU. It also uses the startup code to initialize registers and clear the memory. The Bootloader is responsible for initializing the interface modules like RAM, flash and the interrupt controller. The Bootloader is executed only once at the power up but the Startup code is executed every time we press the reset button.

4) Hardware Abstraction Layer:

In embedded, **Hardware abstraction layer** is a layer of programming that allows the microcontroller to interact with hardware devices connected to its ports. It is represented by writing functions to operate separate modules like LCD or Keypad without taking into consideration the type of the **MCU**. This layer uses another low level layer functions called **MCAL** (microcontroller abstraction layer), the **MCAL** contains drivers that is dependent on the MCU like GPIO initializing and protocols settings. HAL itself is used by a higher level layer called APP layer that is the final layer to make the application we need works (this layer uses HAL and HAL uses MCAL)



Layered architecture

5) Bare metal programming:

Bare-metal programming is the term we use to describe programming without any abstraction layer like we stated previously in HAL or APP. It means we make our code by using the registers of the MCU directly like DDRx or SREG, here we're communicating with MCU at the hardware level and taking into consideration the special hardware architecture of each microcontroller. It can be called like we code in the MCA layer (the first layer of layered architecture). One main advantage of Bare-metal programming is by using it we ensure high execution speed of the code. Many instances of Bare-metal programming focus on the working of the processor and other system components.

6) Polling vs interrupt:

There are two main ways to work with multiple devices using the same microcontroller, the first way is by Polling which means that the MCU is going to monitor the state of each device one at a time to see if this device needs to be served or not and then goes to the next device and repeat that process. But if a certain device needs to be served and it's not its turn, it will wait until the MCU monitor its state again (this is extremely dangerous in applications like car air-bags). Another disadvantage is that polling wastes the time of the CPU by making it monitor every device even when there is no need to use it.

The second method to service multiple devices is using interrupts. We program the MCU registers to always monitoring the devices by special registers like MCUCR. When the device needs to be serviced, it will immediately raise a flag to tell the CPU that this device needs to be served. So, the CPU finishes the current execution and goes to service the device no matter what operation it is doing. But in case of many interrupt happening at the same time, they will be serviced according to their priority (it will be like pooling in this case but we can use Direct Memory Access like we stated in the first point in this question). Interrupt will buy us the wasted time in checking for every device and will service the most important devices that need to be serviced at this time.

7) Pre-Scaler :

There are many applications where we need to count time while executing the code (e.g. generating PWM waves with a specific periodic time). For that, every microcontroller has internal clock that is ticking the timer counter register with a certain frequency related to real time. But if the counter register overflows before it reaches the periodic time we need, then we use pre-scaler to slow down the clock and lower it's frequency to make the Timer Counter registers continues counting within the period. Many MCUs have an internal clock of 1MHz and have two 8-bit Counter registers that counts from 0-255 and one 16-bit Counter register that counts from 0-65535. So, we use pre-scalers to extend the time the register reaches their top.

8) How to select ADC conversion time :

The ADC module needs (in some controllers) 13-cyles to convert the input into binary number. It uses 10 cycles in the operation of successive approximation (10-bit ADC register) and 3 cycles to store and get data. So, if we need to control the ADC conversion time we can first set a pre-scaler to its clock before using the internal CPU clock and in this way the time of each cycle will be changed. This way can be achieved by setting the ADPS2:0 bits in the ADCSRA register. Another way to manipulate the ADC conversion time is by reducing its accuracy. This happens if we make the successive approximation process applies to only the 8-bits of the ADCH. This way, the ADC will take only 11 cycles to convert the input value and we can do this by setting the ADLAR bit in the ADMUX register to ONE (but then we need to multiply the value in ADCH by four).

9) Sampling frequency and its relation to Nyquist theorem:

While sampling data from any wave we receive (e.g. sound waves) we need to avoid the Aliasing when a signal is digitized (converted to digital), it is the effect of having insufficient sampling times of the wave so we cannot fully characterize it or reconstruct it when needed. This effect could happen in many applications in real life like bad sound quality or the wagon will effect when filming a moving wheel, it appears to us like to stand still or moving slow because of its high radial velocity.

The Nyquist theorem suggests that the sampling rate/frequency of the receiver has to be the double of the frequency we receive because of quantization errors. The Nyquist criteria guarantees that there is enough sample points provided the signal is a sin wave and if the signal consists of more than one sin wave, we make the sampling rate twice the highest frequency in these waves.

10) Code refactoring :

It will be a naïve believe to think that any application has ever been programmed once and for all without rewriting it and correcting some errors. But if the code works fine, that doesn't mean you're all done. Code refactoring takes place in many software engineering projects (especially big ones) which actually work but to make the code of a better quality without any effect on its main functionality. This includes increasing its modularity and reusability by covering some corner cases and making the code more generic, optimization of memory usage by getting rid of unused variables, reducing code complexity and execution time by using more efficient algorithms or even writing comments to make it more readable to anyone reads the code.

11) Unit testing and test driven development:

Before deploying your code, you will need to test it and discover what errors could happen with it. But incase we're working on mega projects that contains many functions or modules, we need to test each Unit of them independently. Also, we need to make this test in a repeatable and automated way.

Unit Tests:

To solve the previous issues, we'd run a test program that contains functions to test the outputs of every function in the project. It lets us know which units failed the test and raised unexpected outputs or errors and which ones succeeded. The C language itself has many unit testing frameworks like Check and AceUnit.

Test Driven Development:

TDD is to write tests for your project before even writing the code inside it. This is very helpful as you're beginning to think of every different corner cases that you want the project to pass in the first place. And when you start implementing the code, you can get an immediate feedback on whether it works or not. This also makes the refactoring of your code much easier that you will be assured the rest of your code doesn't break while you make changes. It has some main stages as follows:

I. We write tests to cover all different scenarios we can think of.
II. Those tests fail at first as the code is not complete yet.
III. After more work on implementation, the code passes all tests.
IV. The final stage is to start Code Refactoring and make sure we don't break it.

12) C programming build process:

This process is how the high level source code we write in C is converted into binary executable file that is really understood by the MCU. It has many operations and multiple tools that can be different from one project to another but almost they're all common in the following stages:

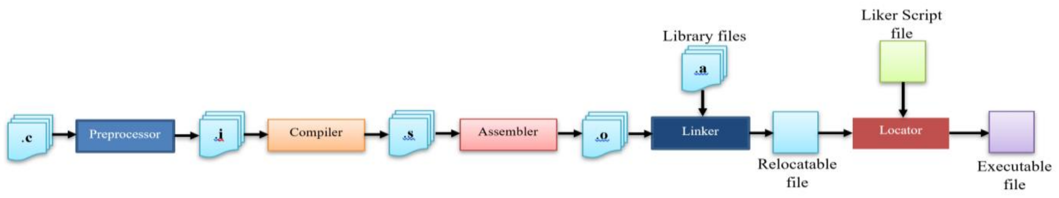
I. Preprocessor stage:
In this stage we evaluate all the preprocessor directives like striping out comments and spaces and making substitution for macro definitions used in the code (any line starts with '#'). The preprocessor takes file.c and output file.i for the next stage.

II. Compiler stage:
In this stage we start to deal with the code depending on the architecture we program. Almost any specific architecture in the world has its special assembly language. So, we write our code in C language that is general for all of them. Then, the compiler analyzes the code and the architecture we use to generate assembly instructions and in many cases, the compiler optimizes the code (like we stated in explaining "volatile" keyword). The compiler takes the file.i and outputs file.s or file.asm to enter the next stage.

III. Assembler stage:
After taking the assembly instructions from file.asm the assembler (which can be a part of the compiler sometimes) takes this file and converts it into an object file that contains opcodes (also known as machine code) which are instructions to determine which operations to be performed by the machine itself. It also allocates the code in memory and allocates used data in data sections. This stage takes the file.asm and outputs file.o or file.obj.

IV. Linker stage:
In this stage the linker gets the object files (in projects that contains many files) and standard C libraries we used and converts them to one relocatable file. It first collects all the files referenced in our main functions (if there is something referenced more than one time it will throw redefinition error) and then changes the addresses assigned to labels. The output of this stage is Relocatable ELF object file

V. Locator Stage:
This stage mostly happens inside the Linker itself but can be completed in a separate tool. The locator takes the Relocatable file from the previous stage and the Linker Script file (it provides the locator with information about actual memory layout) to output the final binary executable binary image, file.exe that is ready to be loaded to the MCU.

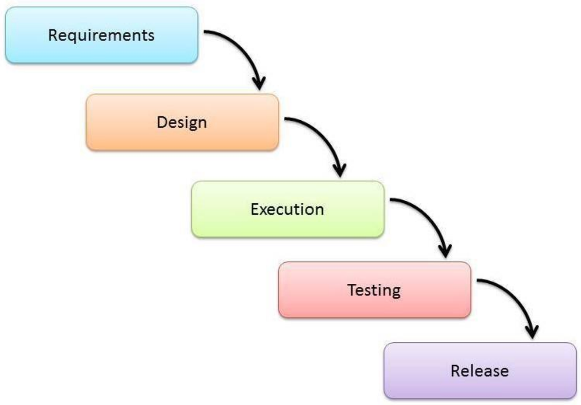


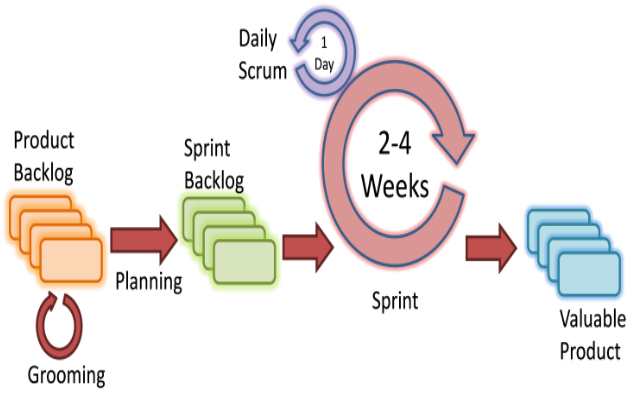
13) Waterfall and Scrum frameworks:

Both frameworks are Project management methodologies to describe the way we build a software project.

Waterfall	Scrum
It is a linear process flow that is seen as flowing downwards through the stages of work. It means that working on some	Scrum methodology is driven from Agile category where there are many cross-functional teams collaborating on the

phase cannot be started till the working on the previous phase is complete. It is one of the first used approaches in software development as it is easy to explain and helps to make plans with specific requirements in each stage	project. It focuses on working in a simultaneous way between each sub-team to complete their work. As software applications are getting more complex, Scrum is most efficient to use and make the target in time due to its adaptability in changing the frequency of work.





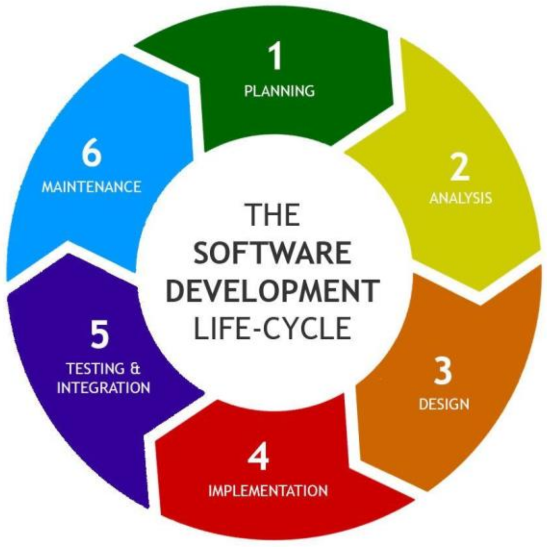
14) Software Development Life Cycle:

SDLC is a framework we put to describe the methodology in developing software applications with high quality to achieve market goals. It is the process that any Software application should go through and contains a specific plan with tasks and goals in each stage of this plan. The stages are mainly as follows (it can be different for each project in some details):

I. Planning and Requirement definition
II. Requirement analysis
III. Designing the product structure
IV. Building the product and implementation

V. Testing and integration of the product
VI. Maintenance and Deployment in market

Also, notice that there are many SDLC models in the industry these days. As an example: Waterfall Model, Iterative Model, Spiral Model, V-Model and Big Bang Model. We choose between them depending on the project and its goals.


1
PLANNING
2
ANALYSIS
3
DESIGN
4
IMPLEMENTATION
5
TESTING & INTEGRATION
6
MAINTENANCE
THE SOFTWARE DEVELOPMENT LIFE-CYCLE