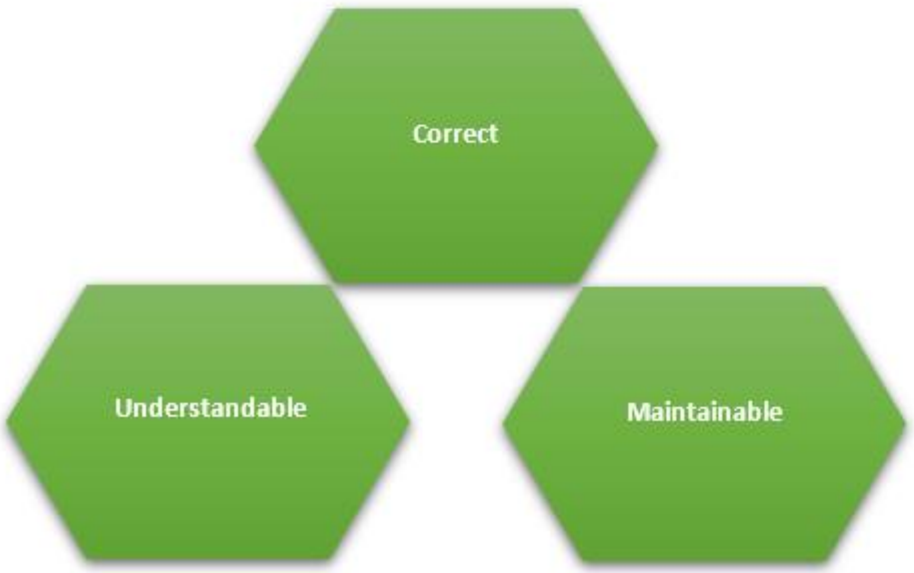
## Embedded Software Design Characteristics

Good software design has three main characteristics. Your software design should be correct, meaning, that everything is correct and nothing is wrong from gathering, requirements (inputs), so your design solution should meet these requirements. The second one, the software design should be understandable, and it means that any stockholder should understand the design regardless of his/her background, however, we do a lot of reviews between the team members to clarify the design. The third characteristic of a good embedded software design is to be maintainable; we mean here that the system should allow editability, in case of changing the requirements, we can change the design without damaging any other part or any other component in the system.



## ESD

The most important terms and their definitions are summarized again in the following table:

| Term | Meaning, definition |
|---|---|
| Embedded system | An embedded system is an electronic calculator or computer, which carries out a defined function and is embedded in a physical environment, is optionally surrounded by other subsystems and has an optional user interface. |
| Design (activity) | The activity that defines how a system is built from several components (architecture elements). |
| System design (activity) | See “Design”— with the specification that it is the design of a system, i.e. various elements such as hardware, software, mechanics. |
| Software design (activity) | See “Design”— with the specification that it is the design of a software (software elements). |
| Architecture | The architecture of a system is the set of structures needed to infer the system, or to ensure that the system satisfies the required properties. The structures include the elements, their relationships and their properties. |
| The design | The structures which, in addition to the architecture-relevant structures (see “Architecture”), also define the structure of the system, but which are not relevant for the required system properties. |
| The system architecture | See “Architecture”— with the specification that it is the design of a system, i.e. various elements such as hardware, software, mechanics. |

| | |
|---|---|
| The software architecture | See “Architecture”— with the specification that it is the design of a software (software elements). |

Tab. 1: Terms and their meaning/definition

## Embedded Software vs Embedded Systems

The hardware components within a device that are running embedded software are referred to as an "embedded system." Some examples of hardware components used in embedded systems are power supply circuits, central processing units, flash memory devices, timers, and serial communication ports. During a device's early design phases, the hardware that will make up the embedded system – and its configuration within the device – is decided. Then, embedded software is developed from scratch to run exclusively on that hardware in that precise configuration. This makes embedded software design a very specialized field that requires deep knowledge of hardware capabilities and computer programming.

## Examples of embedded software-based functions

Almost every device made with circuit boards and computer chips has these components arranged into a system that runs embedded software. As a result, embedded software systems are ubiquitous in everyday life and are found throughout consumer, industrial, automotive, aerospace, medical, commercial, telecom, and military technology.

Common examples of embedded software-based features include:

- Image processing systems found in medical imaging equipment
- Fly-by-wire control systems found in aircraft
- Motion detection systems in security cameras
- Traffic control systems found in traffic lights
- Timing and automation systems found in smart home devices

## What are the different types of embedded software and their purposes?

- Operating System – An operating system (OS), in its most general sense, is software that allows a user to run other applications on a computing device. The operating system manages a processor’s hardware resources including input devices such as a keyboard and mouse, output devices such as displays or printers, network connections, and storage devices such as hard drives and memory. The OS also provides services to facilitate the efficient execution and management of, and memory allocations for software application programs.

- Firmware – Firmware is a type of software that is written directly for a piece of hardware. It operates without going through APIs, the operating system, or device drivers—providing the needed instructions and guidance for the device to communicate with other devices or perform a set of basic tasks and functions as intended.
- Middleware – Middleware is a software layer situated between applications and operating systems. Middleware is often used in distributed systems where it simplifies software development by providing the following:
    - Hiding the intricacies of distributed applications
    - Masking the heterogeneity of hardware, operating systems and protocols
    - Providing uniform and high-level interfaces used to make interoperable, reusable and portable applications.
    - Delivering a set of common services that minimizes duplication of efforts and enhances collaboration between applications
- Application – The end-user develops the final software application that runs on the operating system, uses or interacts with the middleware and firmware, and is the primary focus of the embedded systems’ target function. Each end application is unique while operating systems and firmware can be identical from device to device.

## What are the different UML diagrams?

UML Diagrams are divided into **two broad categories first one is structural and second one is behavioral**. In Unified Modelling Language of Objects Oriented Analysis and Design, there are **14 types of UML diagrams, which are divided into these two categories**.

1. **Structural (Static) view:** emphasizes the static structure of the system using objects, attributes, operations and relationships. It includes class diagrams and composite structure diagrams. Its specify the structure of the object.

Structure diagrams depict the static structure of the elements in your system. i.e., how one object relates to another. It shows the things in the system – classes, objects, packages or modules, physical nodes, components, and interfaces.

- **Composite Structure Diagram:** It shows the internal structure of a classifier, classifier interactions with the environment through ports, or behavior of a collaboration.
- **Deployment Diagram:** It shows a set of nodes and their relationships that illustrates the static deployment view of an architecture.
- **Package Diagram:** It groups related UML elements into a collection of logically related UML structure.

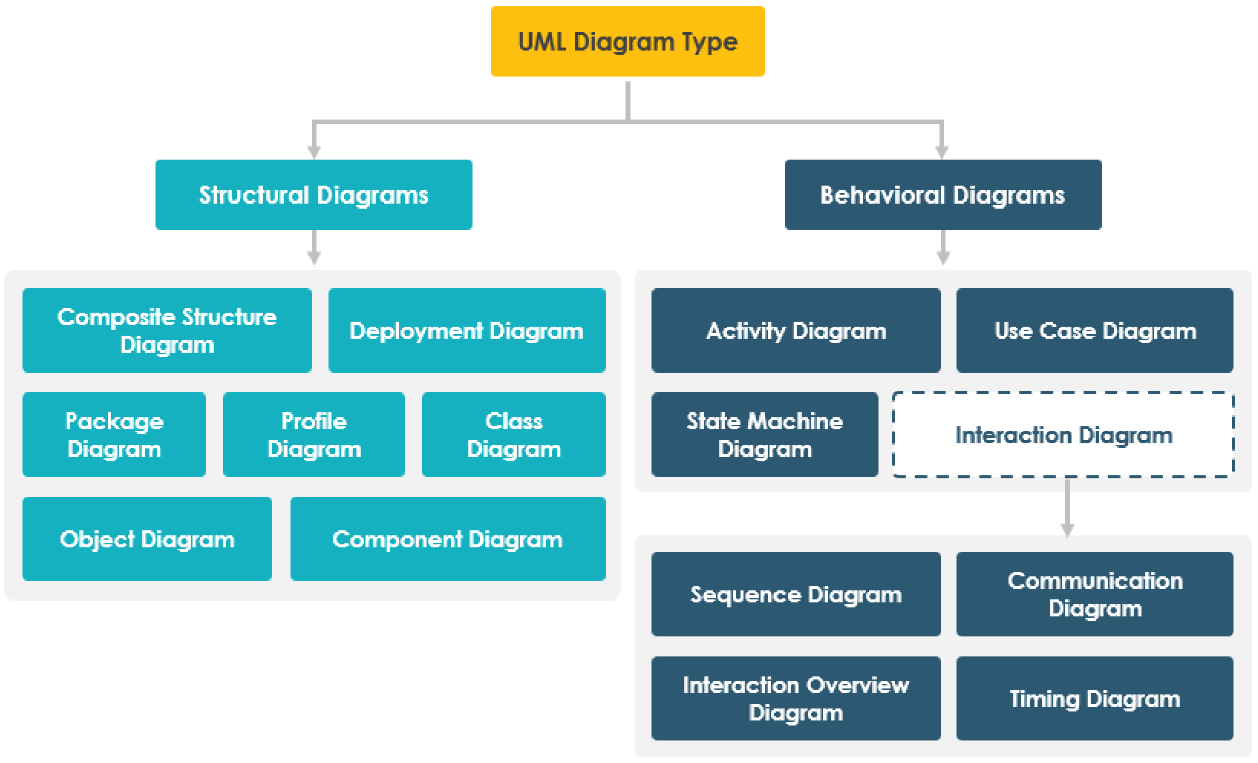
- **Class Diagram:** It shows a set of classes, interfaces, and collaborations and their relationships, typically, found in modeling object-oriented systems.
- **Object Diagram:** It shows a set of objects and their relationships, which is the static snapshots of instances of the things found in class diagrams.
- **Component Diagram:** It shows a set of components and their relationships that illustrates the static implementation view of a system.
- **Profile Diagram:** It shows the basic profile and work of an object.

**2. Behavioral ( Dynamic) view:** emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams, and state machine diagrams. Its represent the object interaction during runtime.

UML's five behavioral diagrams are used to visualize, specify, construct, and document the dynamic aspects of a system. It shows how the system behaves and interacts with itself and other entities (users, other systems). They show how data moves through the system, how objects communicate with each other, how the passage of time affects the system, or what events cause the system to change internal states.

There are seven behavioral diagrams that you can model the dynamics of a system as:

- **Activity Diagram:** It is a graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.
- **Use Case Diagram:** It describes a system's functional requirements in terms of use cases that enable you to relate what you need from a system to how the system delivers on those needs.
- **State Machine Diagram:** It shows the discrete behavior of a part of a designed system through finite state transitions.
- **Sequence Diagram:** It shows the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.
- **Communication Diagram:** It shows interactions between objects and parts using sequenced messages in a free-form arrangement.
- **Interaction overview Diagram:** It depicts a control flow with nodes that can contain other interaction diagrams.
- **Timing Diagram:** It shows interactions when the primary purpose of the diagram is to reason about time by focusing on conditions changing within and among lifelines along a linear time axis.



**What are some of the most important considerations when designing embedded software?**

When designing embedded software, there are several important considerations to keep in mind:

1. Hardware constraints: The hardware platform on which the software will run can have a significant impact on the design of the software. You will need to consider the available processing power, memory, and other resources, as well as any hardware-specific features or constraints that may affect the design.
2. Performance requirements: Embedded systems often have strict performance requirements, such as real-time constraints or the need to operate for long periods without interruption. You will need to design your software to meet these requirements, which may involve optimizing algorithms, minimizing resource usage, or designing for fault tolerance.
3. Reliability: Embedded systems often operate in mission-critical or safety-critical environments, where reliability is of the utmost importance. You will need to design your software to be as reliable as possible, which may involve using redundant systems or designing for fault tolerance.
4. Security: Embedded systems may be vulnerable to cyber attacks or other security threats, so you will need to design your software to be as secure as possible. This may involve implementing security measures such as encryption, authentication, or access controls.

5. Reusability: Reusable code is often easier to maintain and can save time and resources in the long run. You should aim to design your software to be as reusable as possible, which may involve creating modular code or designing for flexibility.

6. Maintainability: Embedded systems may need to be maintained or updated over time, so you should design your software to be as maintainable as possible. This may involve using a clear and consistent coding style, providing detailed documentation, and using version control to track changes to the code.

## When is UML used?

UML is a language to describe a program. It has modelling languages for various things: interactions with the program, program design, etc. This usually forms the documentation about a program, for the program's programmers.

If you are describing what the program is able to *do*, you might write user stories, then draw out **use case diagrams** to elaborate on them.

Once you have an idea what the program is able to do, you might design the structure of the program. Things like **sequence diagrams** and **class diagrams** help structure your program.

Describing the state of a program's execution might help explain a program's implementation of a feature, so **object diagrams** help in this by giving a snapshot of a program's state.

All in all, UML helps explain a program, and is useful especially for new developers in a team. This saves time during the onboarding process.