./

Learning Report – Embedded Software Design, Development processes & Standards



Course Code: <CODE>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
| **1** | **9/10/2020** | **Deeksha P** |  | **Bhargav N** |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

# 

Table of Contents

[Activity and Tasks 5](#_Toc53174395)

[Activity 1 - Black box, white box and grey box testing and its Application 5](#_Toc53174396)

[Black box testing in Software Engineering 5](#_Toc53174397)

[White box Testing in Software Engineering 7](#_Toc53174398)

[Gray Box Testing in Software Testing 9](#_Toc53174399)

[Activity 2 – Standards fallowed by Aerospace, Rail and Automotive in EDLC. 11](#_Toc53174400)

[Activity 3 - Overview of the diagrams used in SysML 15](#_Toc53174401)

[References 20](#_Toc53174402)

**List of Figures**

[Figure 1: Black Box Testing 5](#_Toc53174440)

[Figure 2: Equivalent Partitioning 7](#_Toc53174441)

[Figure 3: White Box Testing 8](#_Toc53174442)

[Figure 4: Gray Box Testing 10](#_Toc53174443)

[Figure 5: SysML Diagram 17](#_Toc53174444)

[Figure 6: The Four pillar of SysML 17](#_Toc53174445)

# 

**List of Tables**

[Table 1: Black Box Techniques 7](#_Toc53174494)

# Activity and Tasks

# Activity 1 - Black box, white box and grey box testing and its Application

## Black box testing in Software Engineering

Black-box testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as Specifications based testing. Independent Testing Team usually performs this type of testing during the software testing life cycle.

This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.

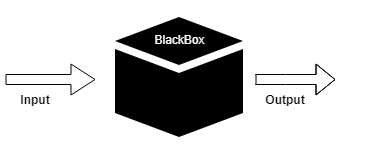


Figure 1: Black Box Testing

Black box testing can be done in following ways:

1. **Syntax Driven Testing –** This type of testing is applied to systems that can be syntactically represented by some language. For example- compilers, language that can be represented by context free grammar. In this, the test cases are generated so that each grammar rule is used at least once.
2. **Equivalence partitioning –** It is often seen that many type of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group. The idea is to partition the input domain of the system into a number of equivalence classes such that each member of class works in a similar way, i.e., if a test case in one class results in some error, other members of class would also result into same error.

**The technique involves the following steps:**

1. Identification of equivalence class
2. Generating test cases
3. Boundary value analysis
4. Cause effect Graphing
5. Requirement based testing
6. Compatibility testing

|  |  |
| --- | --- |
| [Decision Table Technique](https://www.javatpoint.com/decision-table-technique-in-black-box-testing) | Decision Table Technique is a systematic approach where various input combinations and their respective system behaviour are captured in a tabular form. It is appropriate for the functions that have a logical relationship between two and more than two inputs. |
| [Boundary Value Technique](https://www.javatpoint.com/boundary-value-analysis-in-black-box-testing) | Boundary Value Technique is used to test boundary values, boundary values are those that contain the upper and lower limit of a variable. It tests, while entering boundary value whether the software is producing correct output or not. |
| [State Transition Technique](https://www.javatpoint.com/state-transition-technique-in-black-box-testing) | State Transition Technique is used to capture the behaviour of the software application when different input values are given to the same function. This applies to those types of applications that provide the specific number of attempts to access the application. |
| [All-pair Testing Technique](https://www.javatpoint.com/all-pairs-testing-technique-in-black-box-testing) | All-pair testing Technique is used to test all the possible discrete combinations of values. This combinational method is used for testing the application that uses checkbox input, radio button input, list box, text box, etc. |
| [Cause-Effect Technique](https://www.javatpoint.com/cause-and-effect-graph-technique-in-black-box-testing) | Cause-Effect Technique underlines the relationship between a given result and all the factors affecting the result. It is based on a collection of requirements. |
| [Equivalence Partitioning Technique](https://www.javatpoint.com/equivalence-partitioning-technique-in-black-box-testing) | Equivalence partitioning is a technique of software testing in which input data divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior. |
| [Error Guessing Technique](https://www.javatpoint.com/error-guessing-technique-in-black-box-testing) | Error guessing is a technique in which there is no specific method for identifying the error. It is based on the experience of the test analyst, where the tester uses the experience to guess the problematic areas of the software. |
| [Use Case Technique](https://www.javatpoint.com/use-case-technique-in-black-box-testing) | Use case Technique used to identify the test cases from the beginning to the end of the system as per the usage of the system. By using this technique, the test team creates a test scenario that can exercise the entire software based on the functionality of each function from start to end. |

Table 1: Black Box Techniques

**Advantages**

* Tests are done from a user’s point of view and will help in exposing discrepancies in the specifications.
* Tester need not know programming languages or how the software has been implemented.
* Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
* Test cases can be designed as soon as the specifications are complete.

**Disadvantages**

* Only a small number of possible inputs can be tested and many program paths will be left untested.
* Without clear specifications, which is the situation in many projects, test cases will be difficult to design.
* Tests can be redundant if the software designer/developer has already run a test case.
* Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

**Example**

Assume, we have to test a filed which accepts a Mobile Number of ten digits.

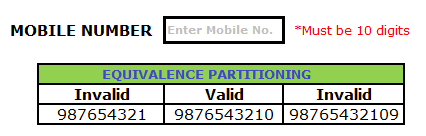


Figure 2: Equivalent Partitioning

Valid input: 10 digits

Invalid Input: 9 digits, 11 digits

Valid Class: Enter 10 digit mobile number = 9876543210

Invalid Class Enter mobile number which has less than 10 digits = 987654321

Invalid Class Enter mobile number which has more than 11 digits = 98765432109

## White box Testing in Software Engineering

White box testing techniques analyse the internal structures the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing.

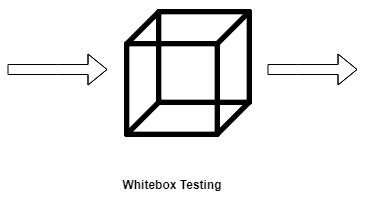


Figure 3: White Box Testing

**Working process of white box testing:**

* **Input:** Requirements, Functional specifications, design documents, source code.
* **Processing:** Performing risk analysis for guiding through the entire process.
* **Proper test planning:** Designing test cases so as to cover entire code. Execute rinse-repeat until error-free software is reached. Also, the results are communicated.
* **Output:** Preparing final report of the entire testing process.

**White Box Testing Techniques:**

* **Statement Coverage -** This technique is aimed at exercising all programming statements with minimal tests.
* **Branch Coverage -** This technique is running a series of tests to ensure that all branches are tested at least once.
* **Path Coverage -** This technique corresponds to testing all possible paths which means that each statement and branch is covered.

**White Box Testing Tools**

Below is a list of top white box testing tools.

• Parasoft Jtest

• EclEmma

• NUnit

• PyUnit

• HTMLUnit

• CppUnit

**Advantages of White Box Testing**

• Code optimization by finding hidden errors.

• White box tests cases can be easily automated.

• Testing is more thorough as all code paths are usually covered.

• Testing can start early in SDLC even if GUI is not available.

**Disadvantages of White Box Testing**

* + - White box testing can be quite complex and expensive.
    - Developers who usually execute white box test cases detest it. The white box testing by developers is not detailed can lead to production errors.
    - White box testing requires professional resources, with a detailed understanding of programming and implementation.
    - White-box testing is time-consuming, bigger programming applications take the time to test fully.

**Example**

Consider the following piece of code

Printme (int a, int b) {

------------ Print me is a function

int result = a+ b;

If (result> 0)

Print ("Positive", result)

Else

Print ("Negative", result)

}

----------- End of the source code

The goal of WhiteBox testing in software engineering is to verify all the decision branches, loops, statements in the code.

To exercise the statements in the above code, WhiteBox test cases would be

* A = 1, B = 1
* A = -1, B = -3

**Applications**

White box testing encompasses several testing types used to evaluate the usability of an application, block of code or specific software package. There are listed below:

* Unit Testing: It is often the first type of testing done on an application. Unit Testing is performed on each unit or block of code as it is developed. Unit Testing is essentially done by the programmer. As a software developer, you develop a few lines of code, a single function or an object and test it to make sure it works before continuing Unit Testing helps identify a majority of bugs, early in the software development lifecycle. Bugs identified in this stage are cheaper and easy to fix.
* Testing for Memory Leaks: Memory leaks are leading causes of slower running applications. A QA specialist who is experienced at detecting memory leaks is essential in cases where you have a slow running software application.

## Gray Box Testing in Software Testing

**Gray Box Testing** is a software testing technique which is a combination of [Black Box Testing](https://www.geeksforgeeks.org/software-engineering-black-box-testing/) technique and [White Box Testing](https://www.geeksforgeeks.org/software-engineering-white-box-testing/) technique. In Black Box Testing technique, tester is unknown to the internal structure of the item being tested and in White Box Testing the internal structure is known to tester. The internal structure is partially known in Gray Box Testing. This includes access to internal data structures and algorithms for purpose of designing the test cases.

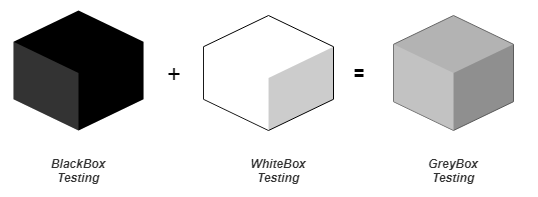


Figure 4: Gray Box Testing

Gray Box Testing is named so because the software program is like a semi-transparent or grey box inside which tester can partially see. It commonly focuses on context-specific errors related to web systems.

**The objective of Gray Box Testing is:**

1. To provide combined advantages of both black box testing and white box testing.
2. To combine the input of developers as well as testers.
3. To improve overall product quality.
4. To reduce the overhead of long process of functional and non-functional testing’s.
5. To provide enough free time to developers to fix defects.
6. To test from the user point of view rather than a designer point of view.

**Gray Box Testing Techniques:**

1. Matrix Testing
2. Pattern Testing
3. Orthogonal Array Testing
4. Regression Testing

**Advantages of Gray Box Testing**

• Users and developers have clear goals while doing testing.

• Gray box testing is mostly done by the user perspective.

• Testers are not required to have high programming skills for this testing.

• Gray box testing is non-intrusive.

• Overall quality of the product is improved.

• In gray box testing, developers have more time for defect fixing.

• By doing gray box testing, benefits of both black box and white box testing is obtained.

• Gray box testing is unbiased. It avoids conflicts between a tester and a developer.

• Gray box testing is much more effective in integration testing.

**Disadvantages of gray box testing**

• Defect association is difficult when gray testing is performed for distributed systems.

• Limited access to internal structure leads to limited access for code path traversal.

• Because source code cannot be accessed, doing complete white box testing is not possible.

• Gray box testing is not suitable for algorithm testing.

• Most of the test cases are difficult to design.

**Application**

* Grey-box testing is a perfect fit for Web-based applications.
* Grey-box testing is also a best approach for functional or domain testing. Functional testing is done basically a test of user interactions with may be external systems. Gray-box testing is well-suited for functional testing due to its characteristics; it also helps to confirm that software meets the requirements defined for the software.

# Activity 2 – Standards fallowed by Aerospace, Rail and Automotive in EDLC.

**AEROSPACE:**

SAE International has been a leading provider of aerospace standards for much of its 106-year history. With more than 22,000 aerospace standards (AS) and aerospace materials specifications (AMS) available, SAE standards are recognized and used globally by manufacturers and suppliers throughout the aerospace industry.

SAE International has been a leading provider of aerospace standards for much of its 106-year history. With more than 22,000 aerospace standards (AS) and aerospace materials specifications (AMS) available, SAE standards are recognized and used globally by manufacturers and suppliers throughout the aerospace industry.

Universally Accepted: SAE standards are used extensively by companies and consultants around the world

Industry Relevant: SAE develops aerospace standards rapidly, allowing users to stay current with the latest trends and technology in the industry.

Always Current: SAE standards are frequently reviewed and updated, so users are assured of having the most current information available.

Historical Perspective: With standards available back to the early 1900s, users can trace the progression of key technologies and components to gain a broader understanding of market needs and changes.

1. **AS9100 Revision Number**: C (Quality Management Systems - Requirements for Aviation, Space and Defense Organizations)

This International Standard specifies requirements for a quality management system where an organization needs to demonstrate its ability to consistently provide product that meets customer and applicable statutory and regulatory requirements.

|  |  |
| --- | --- |
| **Date Published:** | 2009-01-15 |
| **Issuing Committee:** | G-14 Americas Aerospace Quality Standards Committee (Aaqsc) |

1. **ARP4761**;(Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment)

This document describes guidelines and methods of perfomring the safety assessment for certification of civil aircraft. It is primarily associated with showing compliance with FAR/JAR 25.1309. The overall aircraft operating environment is considered. When aircraft derivatives or system changes are certified, the processes described herein are usually applicable only to the new designs or to existing designs that are affected by the changes. In the case of the implementation of existing designs in a new derivation, alternate means such as service experience may be used to show compliance.

|  |  |
| --- | --- |
| **Date Published:** | 1996-12-01 |
| **Issuing Committee:** | S-18, Aircraft And Sys Dev And Safety Assessment Committee |

1. **AMS2750 Revision Number**: E (Pyrometry)

This specification covers pyrometric requirements for thermal processing equipment used for heat treatment. It covers temperature sensors, instrumentation, thermal processing equipment, system accuracy tests, and temperature uniformity surveys. These are necessary to ensure that parts or raw materials are heat treated in accordance with the applicable specifications.

|  |  |
| --- | --- |
| **Date Published:** | 2012-07-27 |
| **Issuing Committee:** | Ams B Finishes Processes And Fluids Committee |

1. **ARD9000** (Aerospace Basic Quality System Standard)

To assure customer satisfaction, aerospace industry manufacturers must produce world class quality products at the lowest possible cost. This document standardizes, to the greatest extent possible, the quality system requirements of the aerospace industry. Standardization of compliance requirements results in cost savings due to the elimination or reduction of unique requirements developed for each different customer.

|  |  |
| --- | --- |
| **Date Published:** | n/a |
| **Issuing Committee:** | n/a |

1. **AS9006 Revision**: A(Deliverable Aerospace Software Supplement for AS9100A, Quality Management Systems - Aerospace - Requirements for Software (based on AS9100A))

The basic requirements of AS9100A apply with the following clarifications. This document supplements the requirements of AS9100A for deliverable software. This supplement contains Quality System requirements for suppliers of products that contain deliverable embedded or loadable airborne, space borne or ground support software components that are part of an aircraft Type Design, weapon system, missile or spacecraft operational software.

|  |  |
| --- | --- |
| **Date Published:** | 2013-07-09 |
| **Issuing Committee:** | G-14 Americas Aerospace Quality Standards Committee (Aaqsc) |

**AUTOMOTIVE:**

The requirements to system and software development tools brought up by the automotive industry differ from the requirements that other customers have.

* **AUTomotive Open System ARchitecture (AUTOSAR):** is a global development partnership of automotive interested parties founded in 2003. It pursues the objective to create and establish an open and standardized software architecture for automotive electronic control units (ECUs). Goals include the scalability to different vehicle and platform variants, transferability of software, the consideration of availability and safety requirements, a collaboration between various partners, sustainable use of natural resources, and maintainability during the whole product.
* **Automotive Industry Standards ( AISs) Passive Safety**
  + **AIS-098: Offset frontal crash:** Requirements for the Protection of the Occupants in the event of an Offset Frontal Collision
  + **AIS-100: Pedestrian protection:**
  + **AIS-99: Side mobile deformable offset**: Approval of Vehicles with regards to the Protection of the Occupants in the event of a Lateral Collision.
* **Automotive Industry Standards ( AISs) Tyres / Brakes / Stability**
  + **AIS- 050**: Requirements for Vehicles with regard to Selection and Approval for use of Tyres for Two and Three-Wheelers.
  + **AIS-091**: Automotives Vehicles-Mechanical Coupling Components of Combinations of Vehicles other Than Agricultural Tractors Requirements
* **Automotive Vehicles-Mechanical Coupling Components of Combinations of Vehicles other Than Agricultural Tractors Requirements.**
  + **AIS-002 / 2001**: Automotive Vehicles – Rear View Mirrors - Installation Requirements.
  + **AIS-074:** Approval of Vehicles with regard to their Protection against Unauthorized use –Two and Three Wheeled Vehicles.
* **Automotive Industry Standards** ( AISs) Miscellaneous Testing
  + **AIS-055**: Automotive CNG / LPG Vehicles – Test Method to Evaluate the Range.
  + **AIS-059**: Automotive Vehicles – Recording Equipment in Road Vehicles (Tachograph).
* **Automotive Industry Standards ( AISs)Safety and Procedural Requirements for Type Approval of LPG operated Vehicles ( Version 3)**
  + **AIS-025 (Version 3)** Amd. 2, 3, 4 & 5: Safety and Procedural Requirements for Type Approval of LPG operated vehicles.
  + **AIS-026 (Version 3)**: Code of Practice for use of LPG Fuel in Internal Combustion Engine to Power 4 Wheeled Vehicles.

**RAIL:**

The following codes, standards and specifications applies to all systems and equipments, as the case maybe, forming part of the project:

* **EN 50119 Railway applications** : Fixed installations - Electric traction overhead contact lines

**EN 50121**: Railway applications- Electromagnetic compatibility: This is a multi-part document divided into the following parts:

* **EN 50121-1**: Railway applications. Electromagnetic compatibility. General.
* **EN 50121-2**: Railway applications. Electromagnetic compatibility. Emission of the whole railway system to the outside world.
* **EN 50121-3-1**: Railway applications. Electromagnetic compatibility. Rolling stock. Train and complete vehicle.
* **EN 50121-3-2**: Railway applications. Electromagnetic compatibility. Rolling stock. Apparatus.
* **EN 50121-4**: Railway applications. Electromagnetic compatibility. Emission and immunity of the signalling and telecommunications apparatus.
* **EN 50121-5**: Railway applications. Electromagnetic compatibility. Emission and immunity of fixed power supply installations and apparatus
* **IEC 60364 (4-41):** Electric installation of Buildings - Electric Shocks having following as applicable releases.
* **IEC 60364-4-41 Amd.1 Ed. 5.0 b**:2017 (Amendment 1 - Low voltage electrical installations - Part 4-41: Protection for safety - Protection against electric shock)

# Activity 3 - Overview of the diagrams used in SysML

**What is SysML?**

The Systems Modelling Language (SysML) is a general purpose modelling language for engineering systems. SysML supports the analysis, design and verification of complex systems including hardware, software, information, personnel, procedures, and facilities in a graphical notation. SysML provides graphical representations based on a semantic foundation for modelling system behaviour, requirements, structure, and parametrical elements which are used to integrate with other engineering analysis models using the OML XML Metadata Interchange (XMI).

SysML includes a graphical construct to represent text based requirements and relate them to other model elements. The requirements diagram captures requirements hierarchies and requirements derivation, and they satisfy and verify relationships allow a modeler to relate a requirement to a model element that satisfies or verifies the requirements. The requirement diagram provides a bridge between the typical requirements management tools and the system models.

**The SysML Diagram-**

1. A graphical modelling language developed in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233a
2. Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities
3. Is a visual modelling language that provides – Semantics = meaning, connected to a meta model (rules governing the creation and the structure of models) – Notation = representation of meaning, graphical or textual.
4. Is not a methodology or a tool (SysML is methodology and tool independent).

**SysML vs UML**

UML is a general-purpose graphical modeling language aimed at Software Engineers

* Diagrams not used
* Object diagram,
* Deployment diagram,
* Component diagram,
* Communication diagram,
* Timing diagram and
* Interaction overview diagram

Diagrams from UML

* Class diagram (Block Definition Diagram - Class→ Block)
* Package diagram
* Composite Structure diagram (Internal Block Diagram)
* State Machine Diagram
* Activity Diagram
* Use Case Diagram
* Sequence Diagram

• In addition, SysML adds some new diagrams and constructs

* Parametric diagram,
* Requirement diagram
* Flow ports,
* Flow specifications
* Item flows.
* Allocation

**SysML Extensions**

* Blocks
* Item flows
* Value properties
* Allocations
* Requirements
* Parametrics
* Continuous flows

**Types of SysML Diagrams**

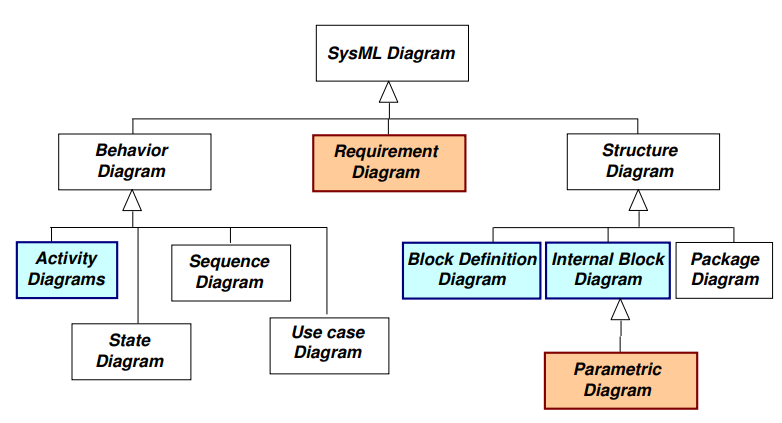


Figure 5: SysML Diagram

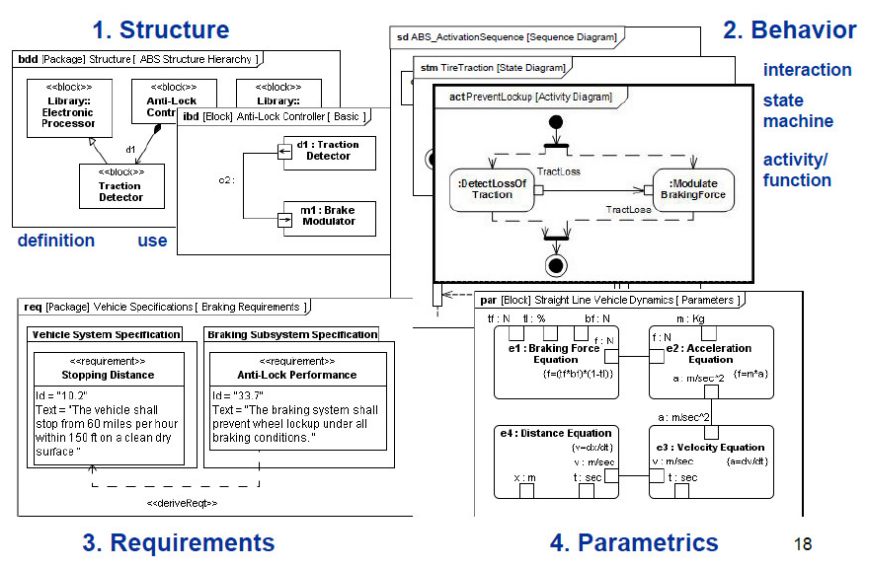


Figure 6: The Four pillar of SysML

* 1. **Behavior Diagram**

The behavior diagrams include the use case diagram, activity diagram, sequence diagram, and state machine diagram.

* **Activity Diagram (act)**

The purpose of Activity diagrams is to specify dynamic system behaviors that Satisfy («satisfy» Dependency) system Functional Requirements using both Control and Object (data) Flows. When properly applied (See Usage Notes below) Activity diagrams are recursively scalable and simulatable. An Activity diagram shows system dynamic behavior using a combined Control Flow and Object (data) Flow model.

* **Sequence diagram (seq)**

A Sequence diagram is a dynamic behavioral diagram that shows interactions (collaborations) among distributed objects or services via sequences of messages exchanged, along with corresponding (optional) events. The purpose of Sequence diagrams is to specify dynamic system behaviors as message-passing collaborations among prototypical Blocks (Parts). When properly applied Activity diagrams are recursively scalable and simulatable.

* **State Machine diagram (smd)**

A State Machine diagram is a dynamic behavioral diagram that shows the sequences of States that an object or an interaction go through during its lifetime in response to Events (a.k.a. "Triggers"), which may result in side-effects (Actions). The purpose of State Machine diagrams is to specify dynamic system behaviors for time-critical, mission-critical, safety-critical, or financially-critical objects. When properly applied. State Machine diagrams are recursively scalable and simulatable.

* **Use Case diagram (uc)**

A Use Case diagram shows communications among system transactions (Use Cases) and external users (Actors) in the context of a system boundary (Subject; notation: rectangle). Actors may represent wetware (persons, organizations, and facilities), software systems, or hardware systems. Defining relationships between the system Subject and the system Actors is an effective informal way to define system scope. The purpose of Use Case diagrams is to provide a high-level view of the subject system and convey the top-level system requirements in non-technical terms for all stakeholders, including customers and project managers as well as architects and engineers. Additional more rigorous SysML diagrams are needed to specify a scalable and simulatable System Architecture Model (SAM).

* 1. **Requirement diagram (req)**

SysML predefines the following stereotype specializations of NFRs:

* «performanceRequirement»
* «interfaceRequirement»
* «designConstraint»
* «physicalRequirement»

A SysML Requirement diagram is a static structural diagram that shows the relationships among Requirement («requirement») constructs, model elements that Satisfy («satisfy» Dependency) them, and Test Cases that Verify («verify» Dependency) them.

The purpose of Requirement diagrams is to specify both Functional and Non-Functional Requirements within the model so that they can be traced to other model elements that Satisfy them and Test Cases that Verify them.

* 1. **Structure Diagram**
* **Block Definition Diagram (bdd)**

A Block Definition Diagram is a static structural diagram that shows system components, their contents (Properties, Behaviors, Constraints), Interfaces, and relationships.Blocks can be recursively decomposed ("nested") into Parts by alternating between Block Definition Diagram (BDD) definitions and Internal Block Diagram (IBD) usages.

Behaviors can either be encapsulated by Blocks (e.g., Operations, Signals, and State Machines) or Allocated (via «allocate» Dependency) to Blocks (e.g., Activities/Actions) directly or indirectly (via Interfaces).

Blocks can be mathematically constrained via Constraint Blocks to produce mathematically simulatable Parametric diagrams. Compare and contrast: UML 2 Class and Component diagrams; SA/SD System Context & Structure Chart diagrams; IDEF IDEF1X diagrams.

**Purpose:**

The purpose of Block Definition Diagrams is to specify system static structures that be used for Control Objects, Data Objects, and Interface Objects. When properly applied Block diagrams are recursively scalable and mathematically (parametrically) simulatable

* **Internal Block Diagram (ibd)**

An Internal Block Diagram is a static structural diagram owned by a particular Block that shows its encapsulated structural contents: Parts, Properties, Connectors, Ports, and Interfaces. Stated otherwise, an IBD is a "white-box" perspective of an encapsuated ("black-box") Block.

Blocks can be recursively decomposed ("nested") into Parts by alternating between Block Definition Diagram (BDD) definitions and Internal Block Diagram (IBD) usages.

Behaviors can either be encapsulated by Blocks (e.g., Operations, Signals, and State Machines) or Allocated (via «allocate» Dependency) to Blocks (e.g., Activities/Actions) directly or indirectly (via Interfaces).

**Purpose:**

The purpose of Internal Block Diagrams (IBDs) is to show the encapsulated structural contents (Parts, Properties, Connectors, Ports, Interfaces) of Blocks so that they can be recursively decomposed and "wired" using Interface Based Design techniques. When used correctly BDDs + IBDs are recursively scalable and mathematically (parametrically) simulatable.

* **Parametric diagram (par)**

A Parametric diagram is a specialization of an Internal Block Diagram (IBD) that enforces mathematical rules (Constraints) defined by Constraint Blocks across the internal Part Value Properties bound by the Constraint Block Parameters.

Binding Connectors (keyword = «equal») between Constraint Block Parameters and internal Part Value Properties effect constraint satisfaction (propagation)

**Purpose:**

The purpose of Parametric diagrams (PARs) is to enforce mathematical rules across Block Value Properties. When used correctly BDDs + IBDs + PARs are recursively scalable and mathematically simulatable.

* **Package diagram (pkg)**

A Package diagram is a static structural diagram that shows the relationships among packages and their contents. Package can be stereotyped (customized) for organizing model elements into models, views, model libraries, and frameworks.

**Purpose:**

The purpose of Package diagram is to support the organization and management of large, complex System Architecture Models (SAMs).

# References

1. <https://inf.mit.bme.hu/sites/default/files/materials/taxonomy/term/445/13/07b_UML-SysML-Overview.pdf>
2. <http://retis.sssup.it/~marco/files/lesson21_SysML.pdf>
3. <https://www.bsigroup.com/en-IN/AS910091109120-Aerospace/Introduction-to-AS910091109120-Aerospace/>
4. <https://www.sae.org/standards/>
5. <https://dzone.com/articles/difference-between-black-box-white-box-and-grey-bo>