QUESTIONS

1. With examples of different programming languages show how programming languages are organized along the given rubrics:

i. Unstructured, structured, modular, object oriented, aspect oriented, activity oriented and event oriented programming requirement.

ii. Based on domain requirements.

iii. Based on requirements i and ii above.

2. Give brief preview of the evolution of programming languages in a chronological order.

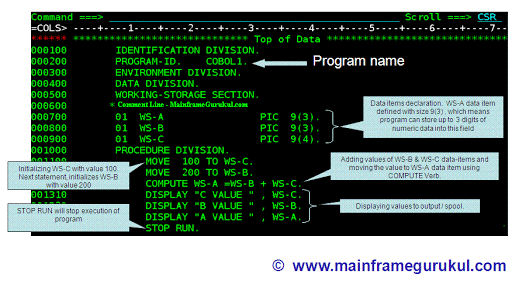
3. Vividly distinguish between modular programming paradigm and object oriented programming paradigm.

SOLUTION

**What is Unstructured Programming?**

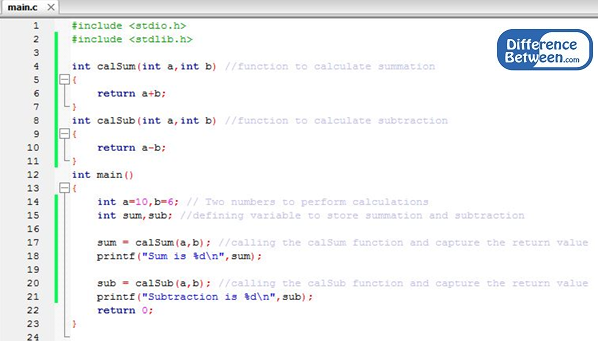
In Unstructured Programming, the code is written as a single whole block. The whole program is taken as a single unit. It is harder to do changes in the program. This paradigm was used in earlier versions of BASIC, COBOL, and FORTRAN. Unstructured programming languages have a limited number of data types like numbers, [arrays](https://www.differencebetween.com/difference-between-arrays-and-vs-arraylists/), strings.

Example of such a programming language using COBOL is shown below:



**What is Structured Programming?**

In Structured Programming, the code is divided into functions or modules. It is also known as **modular programming**. Modules or functions are a set of statements which performs a sub task. As each task is a separate module, it is easy for the programmer to test and debug.  It is also easy to do modifications without changing the whole program. When changing the code, the programmer has to concentrate only on the specific module. [C language](https://www.differencebetween.com/difference-between-java-and-c-language/) and Pascal are some examples of Structural Programming languages.

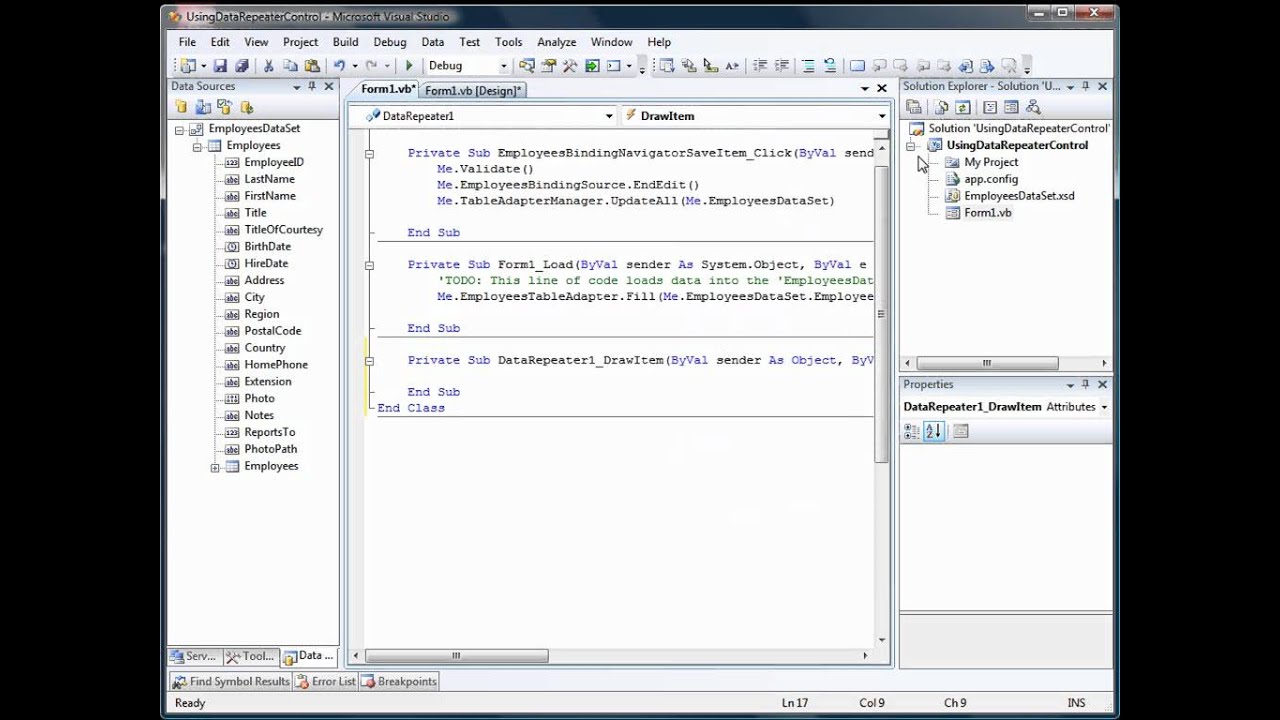


**: Functions using C program**

A programming language like C can use user-defined functions. Functions are called by the main program. Variables in the functions are called [local variables](https://www.differencebetween.com/difference-between-instance-variable-and-vs-local-variable/), and global variables can be accessed by all the functions. Structured programming languages also use selections (if/ else) and iterations (for /do, while). The program above shows the functions using Structured programming language C. Program was written and executed using Code Blocks Development Environment.

**What is Modular Programming?**

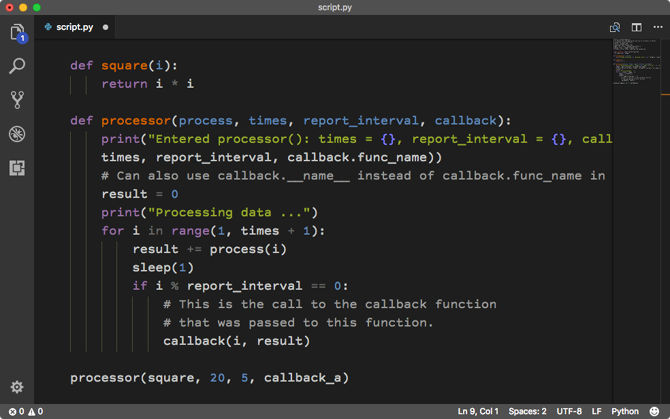
**Modular programming** is a [software design](https://en.wikipedia.org/wiki/Software_design) technique that emphasizes separating the functionality of a [program](https://en.wikipedia.org/wiki/Computer_program) into independent, interchangeable **modules**, such that each contains everything necessary to execute only one aspect of the desired functionality.



A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The [implementation](https://en.wikipedia.org/wiki/Implementation) contains the working code that corresponds to the elements declared in the interface. Modular programming is closely related to [structured programming](https://en.wikipedia.org/wiki/Structured_programming) and [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), all having the same goal of facilitating construction of large software programs and systems by [decomposition](https://en.wikipedia.org/wiki/Decomposition_(computer_science)) into smaller pieces, and all originating around the 1960s. While the historical usage of these terms has been inconsistent, "modular programming" now refers to high-level decomposition of the code of an entire program into pieces: structured programming to the low-level code use of structured [control flow](https://en.wikipedia.org/wiki/Control_flow), and object-oriented programming to the *data* use of [objects](https://en.wikipedia.org/wiki/Object_(computer_science)), a kind of [data structure](https://en.wikipedia.org/wiki/Data_structure).

**What is Object-oriented programming?**

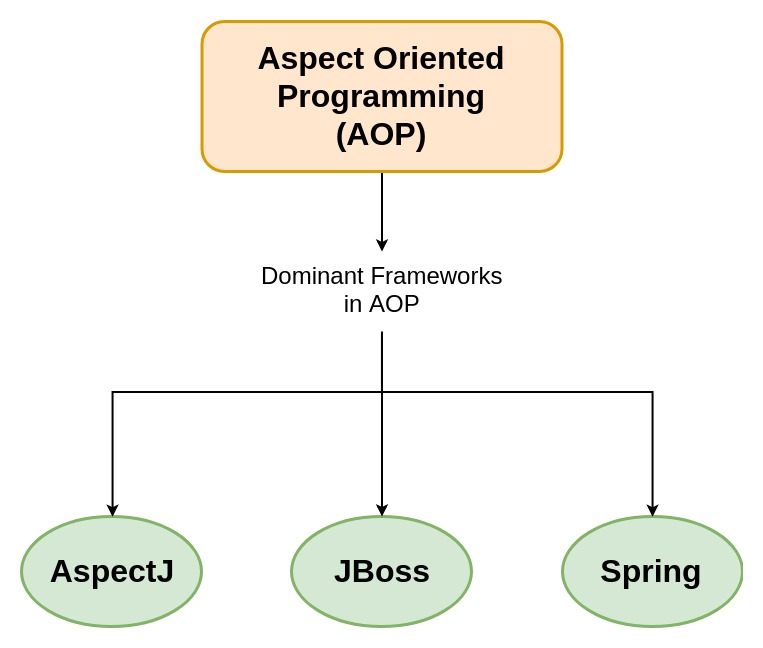
**Object-oriented programming** (**OOP**) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) based on the concept of "[objects](https://en.wikipedia.org/wiki/Object_(computer_science))", which can contain [data](https://en.wikipedia.org/wiki/Data), in the form of [fields](https://en.wikipedia.org/wiki/Field_(computer_science)) (often known as *attributes* or *properties*), and code, in the form of procedures (often known as [*methods*](https://en.wikipedia.org/wiki/Method_(computer_science))). A feature of objects is an object's procedures that can access and often modify the data fields of the object with which they are associated (objects have a notion of "[this](https://en.wikipedia.org/wiki/This_(computer_programming))" or "self"). In OOP, computer programs are designed by making them out of objects that interact with one another.[[1]](https://en.wikipedia.org/wiki/Object-oriented_programming#cite_note-1)[[2]](https://en.wikipedia.org/wiki/Object-oriented_programming#cite_note-2) OOP languages are diverse, but the most popular ones are [class-based](https://en.wikipedia.org/wiki/Class-based_programming), meaning that objects are [instances](https://en.wikipedia.org/wiki/Instance_(computer_science)) of [classes](https://en.wikipedia.org/wiki/Class_(computer_science)), which also determine their [types](https://en.wikipedia.org/wiki/Data_type).



Object-oriented programming uses objects, but not all of the associated techniques and structures are supported directly in languages that claim to support OOP.

**What is Aspect-oriented programming?**

n [computing](https://en.wikipedia.org/wiki/Computing), **aspect-oriented programming** (**AOP**) is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) that aims to increase [modularity](https://en.wikipedia.org/wiki/Modularity_(programming)) by allowing the [separation of](https://en.wikipedia.org/wiki/Separation_of_concerns) [cross-cutting concerns](https://en.wikipedia.org/wiki/Cross-cutting_concern). It does so by adding additional behavior to existing code (an [advice](https://en.wikipedia.org/wiki/Advice_(programming))) *without* modifying the code itself, instead separately specifying which code is modified via a "[pointcut](https://en.wikipedia.org/wiki/Pointcut)" specification, such as "log all function calls when the function's name begins with 'set'". This allows behaviors that are not central to the [business logic](https://en.wikipedia.org/wiki/Business_logic) (such as logging) to be added to a program without cluttering the code, core to the functionality. AOP forms a basis for [aspect-oriented software development](https://en.wikipedia.org/wiki/Aspect-oriented_software_development).





**What is Activity-oriented programming?**

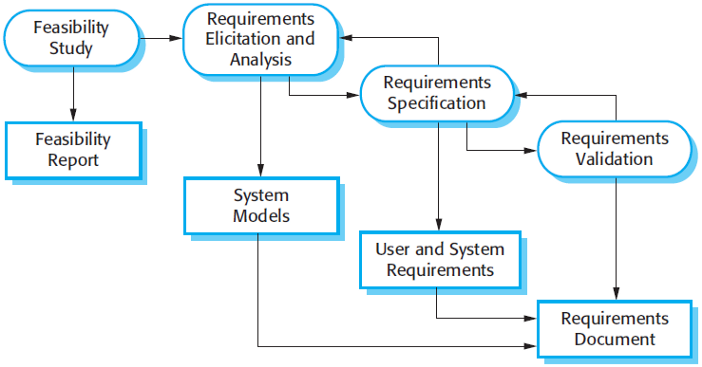
Software specification or requirements engineering is the process of understanding and defining what services are required and identifying the constraints on these services.

Requirements engineering processes ensures your software will meet the user expectations, and ending up with a high quality software.

It’s a critical stage of the software process as errors at this stage will reflect later on the next stages, which definitely will cause you a higher costs.

At the end of this stage, a requirements document that specifies the requirements will be produced and validated with the stockholders.

There are four main activities (or sub-activities) of requirements engineering:



The requirements engineering process.

1. **Feasibility study:**An estimate is made of whether the identified can be achieved using the current software and hardware technologies, under the current budget, etc. The feasibility study should be cheap and quick; it should inform the decision of whether or not to go ahead with the project.
2. **Requirements elicitation and analysis:** This is the process of deriving the system requirements through observation of existing systems, discussions with stakeholders, etc. This may involve the development of one or more system models and prototypes that can help us understanding the system to be specified.
3. **Requirements specification:**It’s the activity of writing down the information gathered during the elicitation and analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document; [user and system requirements](https://medium.com/omarelgabrys-blog/requirements-engineering-introduction-part-1-6d49001526d3).
4. **Requirements validation:**It’s the process of checking the requirements for realism, consistency and completeness. During this process, our goal is to discover errors in the requirements document. When errors are found, it must be modified to correct these problems.

Of course, the activities in the requirements process are not simply executed in a strict sequence, but, they are interleaved. For example, analysis activity continues during the specification as new requirements come to light.

*In agile methods, requirements are developed incrementally according to user priorities and the elicitation of requirements comes from users who are part of the development team.*

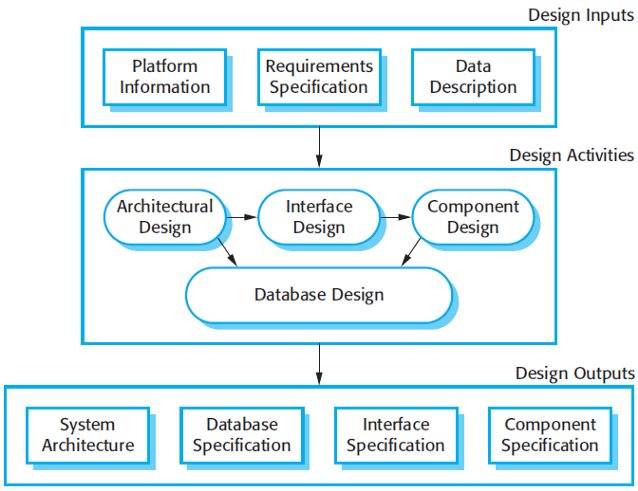
# Software Design And Implementation

The implementation phase is the process of converting a system specification into an executable system. If an incremental approach is used, it may also involve refinement of the software specification.

A software design is a description of the structure of the software to be implemented, data models, interfaces between system components, and maybe the algorithms used.

The software designers develop the software design iteratively; they add formality and detail and correct the design as they develop their design.

Here’s an abstract model of the design process showing the inputs, activities, and the documents to be produced as output.



The software design process

The diagram suggests that the stages of the design process are sequential. In fact, they are interleaved. A feedback from one stage to another and rework can’t be avoided in any design process.

These activities can vary depending on the type of the system needs to be developed. We’ve showed four main activities that may be part of the design process for information systems, and they are:

1. **Architectural design:**It defines the overall structure of the system, the main components, their relationships.
2. **Interface design:**It defines the interfaces between these components. The interface specification must be clear. Therefore, a component can be used without having to know it’s implemented. Once the interface specification are agreed, the components can be designed and developed concurrently.
3. **Component design:**Take each component and design how it will operate, with the specific design left to the programmer, or a list of changes to be made to a reusable component.
4. **Database design:**The system data structures are designed and their representation in a database is defined. This depends on whether an existing database is to be reused or a new database to be created.

These activities lead to a set of design outputs. The detail and representation vary based on the system being developed.

For example, in critical systems, detailed design documents giving a precise and accurate description of the system must be produced.

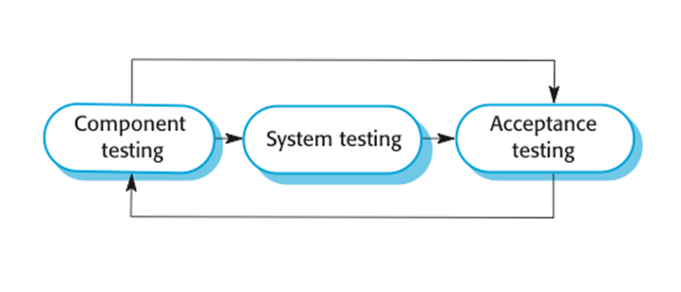
These outputs may be graphical models of the system, and in many cases, automatically generating code from these models.

# Software Verification And Validation

Software validation or, more generally, verification and validation (V&V) is intended to show that a system both conforms to its specification and that it meets the expectations of the customer.

Validation may also involve checking processes, such as inspections or reviews at each stage of the software process, from defining the requirements till the software development.

Testing, where the system is executed using simulated test data, is an important validation technique.



The stages of testing

Testing has three main stages:

1. **Development (or component) testing:**The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components.
2. **System testing:**System components are integrated to create a complete system. This process is concerned with finding errors that result from interactions between components. It is also concerned with showing that the system meets its functional and non-functional requirements.
3. **Acceptance testing:**This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than using simulated test data. It may reveal errors in the system requirements definition.

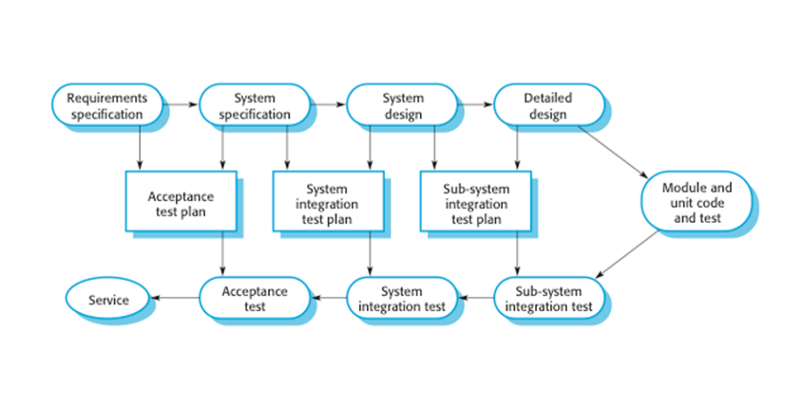
*Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities. Test automation tools, such as JUnit are commonly used to run component tests.*

Normally, component development and testing process are interleaved. Programmers tend o make up their own test data and incrementally test the code as it’s developed.

In some other cases, tests are developed along with the requirements before the development starts. This helps the testers and developers to understand the requirements and reveals requirements problems.

When a [plan-driven](https://medium.com/omarelgabrys-blog/software-engineering-software-process-and-software-process-models-part-2-4a9d06213fdc) software process is used, testing is driven by a set of test plans, which created from the system specification and design.

How the test plans are the link between each phase of the development life cycle and its associated phase of testing can be demonstrated by a software process model called “V-model”.



Testing phases in a plan-driven software process

# Software Maintenance

Requirements are always changing, even after the system has been put into its operating environment. The flexibility of software systems is one of the main reasons why software is being used in large, complex systems.

Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance).

However, this distinction is increasingly irrelevant, and it makes much more sense to see development and maintenance as a continuum.

Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.

**What is Event -oriented programming?**

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), **event-driven programming** is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) in which the [flow of the program](https://en.wikipedia.org/wiki/Control_flow) is determined by [events](https://en.wikipedia.org/wiki/Event_(computing)) such as user actions ([mouse](https://en.wikipedia.org/wiki/Computer_mouse) clicks, key presses), [sensor](https://en.wikipedia.org/wiki/Sensor) outputs, or [messages](https://en.wikipedia.org/wiki/Message_passing) from other programs or [threads](https://en.wikipedia.org/wiki/Thread_(computer_science)). Event-driven programming is the dominant paradigm used in [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface) and other applications (e.g., JavaScript [web applications](https://en.wikipedia.org/wiki/Web_application)) that are centered on performing certain actions in response to [user input](https://en.wikipedia.org/wiki/Input/output). This is also true of programming for [device drivers](https://en.wikipedia.org/wiki/Device_driver) (e.g., [P](https://en.wikipedia.org/wiki/P_(programming_language)) in USB device driver stacks[[1]](https://en.wikipedia.org/wiki/Event-driven_programming#cite_note-1)).

In an event-driven application, there is generally a [main loop](https://en.wikipedia.org/wiki/Event_loop) that listens for events, and then triggers a [callback function](https://en.wikipedia.org/wiki/Callback_(computer_programming)) when one of those events is detected. In [embedded systems](https://en.wikipedia.org/wiki/Embedded_system), the same may be achieved using [hardware interrupts](https://en.wikipedia.org/wiki/Hardware_interrupt) instead of a constantly running main loop. Event-driven programs can be written in any [programming language](https://en.wikipedia.org/wiki/Programming_language), although the task is easier in languages that provide [high-level abstractions](https://en.wikipedia.org/wiki/Abstraction_(computer_science)), such as [await](https://en.wikipedia.org/wiki/Async/await)

## Event handlers

### A trivial event handler[

Because the code is for checking for events and the [main loop](https://en.wikipedia.org/wiki/Main_loop) are common amongst applications, many programming frameworks take care of their implementation and expect the user to provide only the code for the event handlers. In this simple example, there may be a call to an event handler called OnKeyEnter() that includes an argument with a string of characters, corresponding to what the user typed before hitting the ENTER key. To add two numbers, storage outside the event handler must be used. The implementation might look like below.

globally declare the counter K and the integer T.

OnKeyEnter(character C)

{

convert C to a number N

if K is zero store N in T and increment K

otherwise, add N to T, print the result and reset K to zero

While keeping track of history is normally trivial in a sequential program because event handlers execute in response to external events, correctly structuring the handlers to work correctly when called in any order can require special attention and planning in an event-driven program.

### Creating event handlers

The first step in developing an event-driven program is to write a series of [subroutines](https://en.wikipedia.org/wiki/Subroutines), or [methods](https://en.wikipedia.org/wiki/Method_(computer_science)), called event-handler routines. These routines handle the events to which the main program will respond. For example, a single left-button mouse-click on a command button in a [GUI](https://en.wikipedia.org/wiki/Graphical_user_interface) program may trigger a routine that will open another window, save data to a [database](https://en.wikipedia.org/wiki/Database) or exit the application. Many modern-day programming environments provide the programmer with event templates, allowing the programmer to focus on writing the event code.

The second step is to bind event handlers to events so that the correct function is called when the event takes place. Graphical editors combine the first two steps: double-click on a button, and the editor creates an (empty) event handler associated with the user clicking the button and opens a text window so you can edit the event handler.

The third step in developing an event-driven program is to write the [main loop](https://en.wikipedia.org/wiki/Main_loop). This is a function that checks for the occurrence of events, and then calls the matching event handler to process it. Most event-driven programming environments already provide this main loop, so it need not be specifically provided by the application programmer. [RPG](https://en.wikipedia.org/wiki/IBM_RPG), an early programming language from [IBM](https://en.wikipedia.org/wiki/IBM), whose 1960s design concept was similar to event-driven programming discussed above, provided a built-in main [I/O](https://en.wikipedia.org/wiki/I/O) loop (known as the "program cycle") where the calculations responded in accordance to 'indicators' ([flags](https://en.wikipedia.org/wiki/Flag_(computing))) that were set earlier in the cycle.

### Exception handlers in PL/I

In [PL/I](https://en.wikipedia.org/wiki/PL/I), even though a program itself may not be predominantly event-driven, certain abnormal events such as a hardware error, [overflow](https://en.wikipedia.org/wiki/Overflow_(software)) or "program checks" may occur that possibly prevent further processing. [Exception handlers](https://en.wikipedia.org/wiki/Exception_handling) may be provided by "ON statements" in (unseen) callers to provide [cleaning](https://en.wikipedia.org/wiki/Housekeeping_(computing)) routines to clean up afterwards before termination, or to perform recovery operations and return to the interrupted procedure.

**II. PROGRAMMING LANGUAGES BASED ON DOMAIN REQUIREMENTS**

**Domain-specific language**

A **domain-specific language** (**DSL**) is a [computer language](https://en.wikipedia.org/wiki/Computer_language) specialized to a particular application [domain](https://en.wikipedia.org/wiki/Domain_(software_engineering)). This is in contrast to a [general-purpose language](https://en.wikipedia.org/wiki/General-purpose_language) (GPL), which is broadly applicable across domains. There are a wide variety of DSLs, ranging from widely used languages for common domains, such as [HTML](https://en.wikipedia.org/wiki/HTML) for web pages, down to languages used by only one or a few pieces of software, such as [MUSH](https://en.wikipedia.org/wiki/MUSH) soft code. DSLs can be further subdivided by the kind of language, and include domain-specific [*markup* languages](https://en.wikipedia.org/wiki/Markup_language), domain-specific [*modeling* languages](https://en.wikipedia.org/wiki/Modeling_language) (more generally, [specification languages](https://en.wikipedia.org/wiki/Specification_language)), and domain-specific [*programming* languages](https://en.wikipedia.org/wiki/Programming_language). Special-purpose computer languages have always existed in the computer age, but the term "domain-specific language" has become more popular due to the rise of [domain-specific modeling](https://en.wikipedia.org/wiki/Domain-specific_modeling). Simpler DSLs, particularly ones used by a single application, are sometimes informally called **mini-languages**.

The line between general-purpose languages and domain-specific languages is not always sharp, as a language may have specialized features for a particular domain but be applicable more broadly, or conversely may in principle be capable of broad application but in practice used primarily for a specific domain. For example, [Perl](https://en.wikipedia.org/wiki/Perl) was originally developed as a text-processing and glue language, for the same domain as [AWK](https://en.wikipedia.org/wiki/AWK) and [shell scripts](https://en.wikipedia.org/wiki/Shell_script), but was mostly used as a general-purpose programming language later on. By contrast, [PostScript](https://en.wikipedia.org/wiki/PostScript) is a [Turing complete](https://en.wikipedia.org/wiki/Turing_complete) language, and in principle can be used for any task, but in practice is narrowly used as a [page description language](https://en.wikipedia.org/wiki/Page_description_language).

## Use

The design and use of appropriate DSLs is a key part of [domain engineering](https://en.wikipedia.org/wiki/Domain_engineering), by using a language suitable to the domain at hand – this may consist of using an existing DSL or GPL, or developing a new DSL. [Language-oriented programming](https://en.wikipedia.org/wiki/Language-oriented_programming) considers the creation of special-purpose languages for expressing problems as standard part of the problem-solving process. Creating a domain-specific language (with software to support it), rather than reusing an existing language, can be worthwhile if the language allows a particular type of problem or solution to be expressed more clearly than an existing language would allow and the type of problem in question reappears sufficiently often. Pragmatically, a DSL may be specialized to a particular problem domain, a particular problem representation technique, a particular solution technique, or other aspects of a domain.

A domain-specific language is created specifically to solve problems in a particular domain and is not intended to be able to solve problems outside of it (although that may be technically possible). In contrast, general-purpose languages are created to solve problems in many domains. The domain can also be a business area. Some examples of business areas include:

* domain-specific language for life insurance policies developed internally in large insurance enterprise
* domain-specific language for combat simulation
* domain-specific language for salary calculation
* domain-specific language for billing

A domain-specific language is somewhere between a tiny programming language and a [scripting language](https://en.wikipedia.org/wiki/Scripting_language), and is often used in a way analogous to a [programming library](https://en.wikipedia.org/wiki/Programming_library). The boundaries between these concepts are quite blurry, much like the boundary between scripting languages and general-purpose languages.

### In design and implementation

Domain-specific languages are languages (or often, declared syntaxes or grammars) with very specific goals in design and implementation. A domain-specific language can be one of a visual diagramming language, such as those created by the [Generic Eclipse Modeling System](https://en.wikipedia.org/wiki/Generic_Eclipse_Modeling_System), programmatic abstractions, such as the [Eclipse Modeling Framework](https://en.wikipedia.org/wiki/Eclipse_Modeling_Framework), or textual languages. For instance, the command line utility [grep](https://en.wikipedia.org/wiki/Grep) has a [regular expression](https://en.wikipedia.org/wiki/Regular_expression) syntax which matches patterns in lines of text. The [sed](https://en.wikipedia.org/wiki/Sed) utility defines a syntax for matching and replacing regular expressions. Often, these tiny languages can be used together inside a [shell](https://en.wikipedia.org/wiki/Operating_system_shell) to perform more complex programming tasks.

The line between domain-specific languages and [scripting languages](https://en.wikipedia.org/wiki/Scripting_language) is somewhat blurred, but domain-specific languages often lack low-level functions for filesystem access, interprocess control, and other functions that characterize full-featured programming languages, scripting or otherwise. Many domain-specific languages do not compile to [byte-code](https://en.wikipedia.org/wiki/Byte-code) or executable code, but to various kinds of media objects: GraphViz exports to [PostScript](https://en.wikipedia.org/wiki/PostScript), [GIF](https://en.wikipedia.org/wiki/GIF), [JPEG](https://en.wikipedia.org/wiki/JPEG), etc., where [Csound](https://en.wikipedia.org/wiki/Csound) compiles to audio files, and a ray-tracing domain-specific language like [POV](https://en.wikipedia.org/wiki/POV-Ray) compiles to graphics files. A computer language like [SQL](https://en.wikipedia.org/wiki/SQL) presents an interesting case: it can be deemed a domain-specific language because it is specific to a specific domain (in SQL's case, accessing and managing relational databases), and is often called from another application, but SQL has more keywords and functions than many scripting languages, and is often thought of as a language in its own right, perhaps because of the prevalence of database manipulation in programming and the amount of mastery required to be an expert in the language.

Further blurring this line, many domain-specific languages have exposed APIs, and can be accessed from other programming languages without breaking the flow of execution or calling a separate process, and can thus operate as programming libraries.

### Programming tools

Some domain-specific languages expand over time to include full-featured programming tools, which further complicates the question of whether a language is domain-specific or not. A good example is the [functional language](https://en.wikipedia.org/wiki/Functional_language) [XSLT](https://en.wikipedia.org/wiki/XSLT), specifically designed for transforming one XML graph into another, which has been extended since its inception to allow (particularly in its 2.0 version) for various forms of filesystem interaction, string and date manipulation, and data typing.

In [model-driven engineering](https://en.wikipedia.org/wiki/Model-driven_engineering), many examples of domain-specific languages may be found like [OCL](https://en.wikipedia.org/wiki/Object_Constraint_Language), a language for decorating models with assertions or [QVT](https://en.wikipedia.org/wiki/QVT), a domain-specific transformation language. However, languages like [UML](https://en.wikipedia.org/wiki/Unified_Modeling_Language) are typically general-purpose modeling languages.

To summarize, an analogy might be useful: a Very Little Language is like a knife, which can be used in thousands of different ways, from cutting food to cutting down trees. A domain-specific language is like an electric drill: it is a powerful tool with a wide variety of uses, but a specific context, namely, putting holes in things. A General Purpose Language is a complete workbench, with a variety of tools intended for performing a variety of tasks. Domain-specific languages should be used by programmers who, looking at their current workbench, realize they need a better drill and find that a particular domain-specific language provides exactly that.

### Design goals

Adopting a domain-specific language approach to software engineering involves both risks and opportunities. The well-designed domain-specific language manages to find the proper balance between these.

Domain-specific languages have important design goals that contrast with those of general-purpose languages:

* Domain-specific languages are less comprehensive.
* Domain-specific languages are much more expressive in their domain.
* Domain-specific languages should exhibit minimal [redundancy](https://en.wikipedia.org/wiki/Redundant_code).

### Idioms

In programming, idioms are methods imposed by programmers to handle common development tasks, e.g.:

* Ensure data is saved before the window is closed.
* Edit code whenever command-line parameters change because they affect program behavior.

General purpose programming languages rarely support such idioms, but domain-specific languages can describe them, e.g.:

* A script can automatically save data.
* A domain-specific language can parameterize command line input.

**2.** GIVE BRIEF PREVIEW OF THE EVOLUTION OF PROGRAMMING LANGUAGES IN A CHRONOLOGICAL ORDER

The **history of programming languages** spans from documentation of early mechanical computers to modern tools for software development. Early programming languages were highly specialized, relying on mathematical notation and similarly obscure syntax.[[1]](https://en.wikipedia.org/wiki/History_of_programming_languages#cite_note-1) Throughout the 20th century, research in [compiler](https://en.wikipedia.org/wiki/Compiler) theory led to the creation of high-level programming languages, which use a more accessible syntax to communicate instructions.

The first high-level programming language was [Plankalkül](https://en.wikipedia.org/wiki/Plankalk%C3%BCl), created by [Konrad Zuse](https://en.wikipedia.org/wiki/Konrad_Zuse) between 1942 and 1945.[[2]](https://en.wikipedia.org/wiki/History_of_programming_languages#cite_note-2) The first high-level language to have an associated [compiler](https://en.wikipedia.org/wiki/Compiler) was created by [Corrado Böhm](https://en.wikipedia.org/wiki/Corrado_B%C3%B6hm) in 1951, for [his PhD thesis](http://e-collection.library.ethz.ch/eserv/eth:32719/eth-32719-02.pdf). The first commercially available language was [FORTRAN](https://en.wikipedia.org/wiki/FORTRAN) (FORmula TRANslation), developed in 1956 (first manual appeared in 1956, but first developed in 1954) by a team led by [John Backus](https://en.wikipedia.org/wiki/John_Backus) at [IBM](https://en.wikipedia.org/wiki/IBM).

The list of the programming languages as requested above are:

* 1951 – [Regional Assembly Language](https://en.wikipedia.org/wiki/Assembly_language)
* 1952 – [Autocode](https://en.wikipedia.org/wiki/Autocode)
* 1954 – [IPL](https://en.wikipedia.org/wiki/Information_Processing_Language) (forerunner to LISP)
* 1955 – [FLOW-MATIC](https://en.wikipedia.org/wiki/FLOW-MATIC) (led to COBOL)
* 1957 – [FORTRAN](https://en.wikipedia.org/wiki/Fortran) (first compiler)
* 1957 – [COMTRAN](https://en.wikipedia.org/wiki/COMTRAN) (precursor to COBOL)
* 1958 – [LISP](https://en.wikipedia.org/wiki/Lisp_(programming_language))
* 1958 – [ALGOL 58](https://en.wikipedia.org/wiki/ALGOL_58)
* 1959 – [FACT](https://en.wikipedia.org/wiki/FACT_computer_language) (forerunner to COBOL)
* 1959 – [COBOL](https://en.wikipedia.org/wiki/COBOL)
* 1959 – [RPG](https://en.wikipedia.org/wiki/IBM_RPG)
* 1962 – [APL](https://en.wikipedia.org/wiki/APL_(programming_language))
* 1962 – [Simula](https://en.wikipedia.org/wiki/Simula)
* 1962 – [SNOBOL](https://en.wikipedia.org/wiki/SNOBOL)
* 1963 – [CPL](https://en.wikipedia.org/wiki/Combined_Programming_Language) (forerunner to C)
* 1964 – [Speakeasy](https://en.wikipedia.org/wiki/Speakeasy_(computational_environment))
* 1964 – [BASIC](https://en.wikipedia.org/wiki/BASIC)
* 1964 – [PL/I](https://en.wikipedia.org/wiki/PL/I)
* 1966 – [JOSS](https://en.wikipedia.org/wiki/JOSS)
* 1966 - [MUMPS](https://en.wikipedia.org/wiki/MUMPS)
* 1967 – [BCPL](https://en.wikipedia.org/wiki/BCPL) (forerunner to C)

|  |  |
| --- | --- |
| * 1967 – [BCPL](https://en.wikipedia.org/wiki/BCPL) (forerunner to B) * 1968 – [Logo](https://en.wikipedia.org/wiki/Logo_(programming_language)) * 1969 – [B](https://en.wikipedia.org/wiki/B_(programming_language)) (forerunner to C) * 1970 – [Pascal](https://en.wikipedia.org/wiki/Pascal_(programming_language)) * 1970 – [Forth](https://en.wikipedia.org/wiki/Forth_(programming_language)) * 1972 – [C](https://en.wikipedia.org/wiki/C_(programming_language)) | * 1972 – [Smalltalk](https://en.wikipedia.org/wiki/Smalltalk) * 1972 – [Prolog](https://en.wikipedia.org/wiki/Prolog) * 1973 – [ML](https://en.wikipedia.org/wiki/ML_(programming_language)) * 1975 – [Scheme](https://en.wikipedia.org/wiki/Scheme_(programming_language)) * 1978 – [SQL](https://en.wikipedia.org/wiki/SQL) (a query language, later extended) |
| * 1980 – [C++](https://en.wikipedia.org/wiki/C%2B%2B) (as [C with classes](https://en.wikipedia.org/wiki/C_with_classes), renamed in 1983) * 1983 – [Ada](https://en.wikipedia.org/wiki/Ada_(programming_language)) * 1984 – [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp) * 1984 – [MATLAB](https://en.wikipedia.org/wiki/MATLAB) * 1984 – dBase III, dBase III Plus (Clipper and [FoxPro](https://en.wikipedia.org/wiki/FoxPro) as [FoxBASE](https://en.wikipedia.org/wiki/FoxBASE), later developing into [Visual FoxPro](https://en.wikipedia.org/wiki/Visual_FoxPro)) * 1985 – [Eiffel](https://en.wikipedia.org/wiki/Eiffel_(programming_language)) * 1986 – [Objective-C](https://en.wikipedia.org/wiki/Objective-C) | * 1986 – [LabVIEW](https://en.wikipedia.org/wiki/LabVIEW) (Visual Programming Language) * 1986 – [Erlang](https://en.wikipedia.org/wiki/Erlang_(programming_language)) * 1987 – [Perl](https://en.wikipedia.org/wiki/Perl) * 1988 – [Tcl](https://en.wikipedia.org/wiki/Tcl) * 1988 – [Wolfram Language](https://en.wikipedia.org/wiki/Wolfram_Language) (as part of [Mathematica](https://en.wikipedia.org/wiki/Mathematica), only got a separate name in June 2013) * 1989 – [FL](https://en.wikipedia.org/wiki/FL_(programming_language)) (Backus) |
| * 1990 – [Haskell](https://en.wikipedia.org/wiki/Haskell_(programming_language)) * 1990 – [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) * 1991 – [Visual Basic](https://en.wikipedia.org/wiki/Visual_Basic) * 1993 – [Lua](https://en.wikipedia.org/wiki/Lua_(programming_language)) * 1993 – [R](https://en.wikipedia.org/wiki/R_(programming_language)) * 1994 – [CLOS](https://en.wikipedia.org/wiki/CLOS) (part of ANSI [Common Lisp](https://en.wikipedia.org/wiki/Common_Lisp)) | * 1995 – [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)) * 1995 – [Ada 95](https://en.wikipedia.org/wiki/Ada_95) * 1995 – [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) * 1995 – [Delphi (Object Pascal)](https://en.wikipedia.org/wiki/Embarcadero_Delphi) * 1995 – [JavaScript](https://en.wikipedia.org/wiki/JavaScript) * 1995 – [PHP](https://en.wikipedia.org/wiki/PHP) * 1997 – [Rebol](https://en.wikipedia.org/wiki/REBOL) |

* 2000 – [ActionScript](https://en.wikipedia.org/wiki/ActionScript)
* 2001 – [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language))
* 2001 – [D](https://en.wikipedia.org/wiki/D_(programming_language))
* 2002 – [Scratch](https://en.wikipedia.org/wiki/Scratch_(programming_language))
* 2003 – [Groovy](https://en.wikipedia.org/wiki/Groovy_(programming_language))
* 2003 – [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language))
* 2005 – [F#](https://en.wikipedia.org/wiki/F_Sharp_(programming_language))
* 2006 – [PowerShell](https://en.wikipedia.org/wiki/Windows_PowerShell)
* 2007 – [Clojure](https://en.wikipedia.org/wiki/Clojure)
* 2009 – [Go](https://en.wikipedia.org/wiki/Go_(programming_language))
* 2010 – [Rust](https://en.wikipedia.org/wiki/Rust_(programming_language))
* 2011 – [Dart](https://en.wikipedia.org/wiki/Dart_(programming_language))
* 2011 – [Kotlin](https://en.wikipedia.org/wiki/Kotlin_(programming_language))
* 2011 – [Elixir](https://en.wikipedia.org/wiki/Elixir_(programming_language))
* 2012 – [Julia](https://en.wikipedia.org/wiki/Julia_(programming_language))
* 2012 - [TypeScript](https://en.wikipedia.org/wiki/TypeScript)
* 2014 – [Swift](https://en.wikipedia.org/wiki/Swift_(programming_language))

3. **VIVIDLY DISTINGUISH BETWEEN MODULAR PROGRAMMING PARADIGM AND OBJECT ORIENTED PROGRAMMING PARADIGM.**

An object-oriented program usually contains different types of objects, each corresponding to a particular kind of complex data to manage, or perhaps to a real-world object or concept such as a bank account, a hockey player, or a bulldozer.

Modular programming (also called "top-down design" and "stepwise refinement") is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains everything necessary to execute only one aspect of the desired functionality.

Here is an example (the way I understand it)

Consider you have a program. A few input fields and a button. Then some calculations are made and the program outputs something.

This program can have 2 modules: The input/output one and the calculation one.

However I don't see why the program can't have a layout (a class containing all the objects that will be shown on the screen) and a logic part (which can be a class or a function depending on the depth of the calculations).