# Programming Development Cycles

## ****Program development****

**Programming** is the process of creating a set of instructions that tell a computer how to perform a task. **Programming** can be done using a variety of computer "languages," such as SQL, Java, Python, and C++.

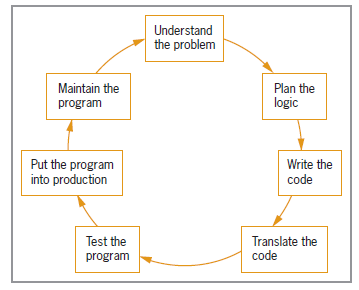
**Syntax** refers to the spelling and grammar of a **programming** language. Computers are inflexible machines that understand what you type only if you type it in the exact form that the computer expects. The expected form is called the **syntax**. Program with syntax errors cannot execute.

A **logic error** (or **logical error**) is a mistake in a program's source code that results in incorrect or unexpected behavior. It is a type of runtime **error** that may simply produce the wrong output or may cause a program to crash while running. Many different types of programming **mistakes** can cause **logic errors**

Program development is the process of creating application **programs**. **Program development** life **cycle** (PDLC) The process containing the five phases of **program development**: analyzing, designing, coding, debugging and testing, and implementing and maintaining application **software**.

The following are six steps in the Program Development Life Cycle:

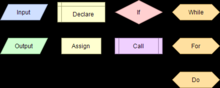
1. Analyze the problem. The computer user must figure out the problem, then decide how to resolve the problem - choose a program.
2. Design the program. A flow chart is important to use during this step of the PDLC. This is a visual diagram of the flow containing the program. This step will help you break down the problem.
3. Code the program. This is using the language of programming to write the lines of code. The code is called the listing or the source code. The computer user will run an object code for this step.
4. Debug the program. The computer user must debug. This is the process of finding the "bugs" on the computer. The bugs are important to find because this is known as errors in a program.
5. Formalize the solution. One must run the program to make sure there are no syntax and logic errors. Syntax are grammatical errors and logic errors are incorrect results.
6. Document and maintain the program. This step is the final step of gathering everything together. Internal documentation is involved in this step because it explains the reasoning one might of made a change in the program or how to write a program



### **Writing Code**

Computer code is a series of statements that have been assigned a function by a higher level language (typically referred to as source code). This language is similar to English and has been converted to machine language using a type of program known as a compiler. Because code is used to instruct computers to perform a wide array of tasks, there are many different kinds of languages and programs available. One of the most important aspects of coding is deciding which jobs (creating a web page, writing a game, etc.) a computer will do. Regardless of what is chosen, the majority of codes utilize plain-text because of its compatibility. Though the actual content is written this way, documents are each given a unique file extension that is indicative of their type. One can write a simple code with a basic word processor or text editor. However, using a software application (specifically designed for coding in a particular language) is significantly more effective and efficient. As with a document written in English, where word processing software is used to aid in detection of spelling errors and non-standard grammar, a coding editor provides comparable tools to ensure accuracy. A code editor is also known as an integrated development environment (IDE), which is a software application for formatting. Using a code editor decreases the chances of errors in codes and time spent reading a code. A large downfall of working with IDEs is a lack of flexibility. While some IDEs work with multiple programming languages, a sizable amount are very specific for only one language.

### **Flowcharts and Pseudocode**



During the design process of the Program Development Life Cycle, it is important that programmers (and non-programmers) are able to visualize the way in which the program will work. Certain tools such as flowcharts and pseudocode are used to simplify the design process and allow the developers to see the program before any actual coding is used. A common type of design tool is the flowchart. A flowchart can be either handwritten or created with software such as Visual Logic or Flowgorithm. Using software helps you save your work digitally which can be more reliable. Many of these software programs have similar symbols to represent certain actions such as input, output, assignments, and various types of loops. For example, a rhombus represents inputs and outputs and a rectangle represents a process. Flowcharts are also useful for education tools because they focus more on the concept of programming rather than focusing on the syntax of languages. Another type of design tool is pseudocode. Pseudocode is very similar to a programming language except that it uses non-syntactical words to summarize the processes of a program. Pseudocode cannot be compiled or executed but it does serve as a good starting point for programmers. Here is an example of pseudocode:

If user’s age is greater than or equal to 18:

Print “You can vote”

Else

Print”You cannot vote”

### **Compiler**

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. When executing (running), the compiler first parses (or analyzes) all of the language statements syntactically one after the other and then, in one or more successive stages or "passes", builds the output code, making sure that statements that refer to other statements are referred to correctly in the final code. A compiler works with what are sometimes called 3GLs (FORTRAN, BASIC, COBOL, C, etc.) and higher-level languages.[[11] (Links to an external site.)Links to an external site.](https://en.wikibooks.org/wiki/Introduction_to_Computer_Information_Systems/Program_Development#cite_note-11) There are one-pass and multi-pass compilers as well as just-in-time compiler, stage compiler, and source-to-source. The compiler front end analyzes the source code to build an internal representation of the program, called the intermediate representation. The compiler backend includes three main phases, such as analysis, optimization, and code generation.[[12] (Links to an external site.)Links to an external site.](https://en.wikibooks.org/wiki/Introduction_to_Computer_Information_Systems/Program_Development#cite_note-12)Because compilers translate source code into object code, which is unique for each type of computer, many compilers are available for the same language. For example, there is a FORTRAN compiler for PCs and another for Apple Macintosh computers. In addition, the compiler industry is quite competitive, so there are actually many compilers for each language on each type of computer. More than a dozen companies develop and sell compilers for the PC.[[13] (Links to an external site.)Links to an external site.](https://en.wikibooks.org/wiki/Introduction_to_Computer_Information_Systems/Program_Development#cite_note-13) There is also something called a decompiler - which translates from a low level language to a high level language.

### **Control Structures**

A control structure is a diagram used to show how functions, statements, and instructions are performed in a program or module. The diagram shows exactly when an instruction is performed, and how it’s performed. Most importantly, a control structure shows the order of the instructions. There are three basic types of control structures: sequence, selection, and repetition. Choosing a specific control structure depends on what you want the program or module to accomplish. A sequence control structure is the simplest and least complex control structure. Sequence control structures are instructions that are executed one after another. The structure could be compared to following a recipe. A more complex control structure might be a selection control structure, a structure that involves conditions or decisions. This means that the structure can allow different sets of instructions to be executed depending on whether a condition is true or false. The last basic control structure is a repetition control structure, which is sometimes called an iteration control structure. This control structure is used when repeating a group of code is necessary. The code will be repeated until a condition is reached. Repetition control structures are used when looping is needed to reach a specific outcome.

### **Testing Program Design**

Good program design needs to be specific. The program design is very important, especially because it involves the overall step-by-step directions regarding the program. A programmer must test the program design to ensure that it runs correctly and that there are no mistakes. The operation a programmer must do to complete this task is called desk checking. Desk checking allows the programmer to run through the program design step-by-step. Essentially, the programmer runs through lines of code to identify potential errors and to check the logic. The programmer uses tracing tables to keep track of any loop counters. The goal of checking the program design is to avoid running into mistakes further on in the program development cycle. The sooner the mistake is caught in the development cycle the better. If the error is not found until later in the developmental cycle, it may delay a project. Therefore, a programmer must make sure they pay strict attention while desk checking. Advantages to desk checking include the convenience of hands-on "proof-reading" of the programmer’s own code. The programmers wrote the code themselves, so it is an advantage that they can work immediately with familiar code. A disadvantage to the desk checking system includes potential human error. Since a computer is not checking the design code, it is prone to human error.

### **Debugging**

Debugging is basically making sure that a program does not have any bugs (errors) so that it can run properly without any problems. Debugging is a large part of what a programmer does. The first step to debugging is done before you can actually debug the program; the program needs to be changed into machine language so that the computer can read it. It is converted using a language translator. The first goal of debugging is to get rid of syntax errors and any errors that prevent the program from running. Errors that prevent the program from running are compiler errors. These need to be removed right away because otherwise you cannot test any of the other parts of the program. Syntax errors occur when the programmer has not followed the correct rules of the programming language. Another kind of error is a runtime error, which occurs while the program is running and it is not noticed until after all syntax errors are corrected. Many run time errors are because of logic errors, which are errors in the logic of the program. It could occur when a formula is written incorrectly or when a wrong variable name is used.

There are different types of debugging techniques that can be used. One technique called print debugging, or also known as the printf method, finds errors by watching the print (or trace) statement live or recorded to see the execution flow of the process. This method originated in the early versions of the BASIC programming language. Remote debugging is the method of finding errors using a remote system or network, and using that different system to run the program and collect information to find the error in the code. If the program has already crashed, then post-mortem debugging can be used through various tracing techniques and by analyzing the memory dump of the program. Another technique is one created by Edward Gauss called wolf-fence debugging. Basically, this method find the error by zeroing in on the problem by continuous divisions or sectioning until the bug is found. Similar to this is the saff squeeze technique which uses progressive inlining of a failure test to isolate the problem.

Debugging a program can be done by using the tools provided in the debugging software. Typically, and especially with high-level programming languages, specific debugging tools are already included in the. Having language-specific debugging tools make it easier to detect the errors in a code, because they can look for known errors as opposed to tediously “walking through” the code manually. It also good to note that fixing one bug manually may lead to there being another bug; this is also why language-specific debugging tools are helpful. There are also debugging software for embedded system as well.

### **Testing/Implementation and Maintenance**

Relating to getting a program up and running, many things need to happen before it can be used. One step is to test the program. After the debugging process occurs, another programmer needs to test the program for any additional errors that could be involved in the background of the program. This person needs to perform all of the tasks that an actual user of the program would use and follow. To ensure privacy rights, test data is used in the testing process. However, this still has the same structure and feel to the actual data. The tester needs to check for possible input errors as well, as this would create many problems and issues in the future had it not been checked. Companies usually implement different types of tests. An Alpha test is first conducted, which is on-site at the company, and Beta tests are sent out to different states or countries to ensure the program is 100% ready for use. The Alpha test occurs before the Beta test. Once the debugging and testing are finished, the program is now in the system and the program implementation and maintenance phase are completed. Program maintenance still needs to be kept up, in case of future errors. This is the most costly to organizations because the programmers need to keep improving and fixing issues within the program.

As stated earlier, a program goes through extensive testing before it is released to the public for use. The two types of testing are called Alpha and Beta testing. First, it is important to know what each test does. Alpha testing is done “in house” so to speak. It is done within a company prior to sending it to Beta testing and its intention in this early stage is to improve the product as much as possible to get it Beta ready. Beta testing is done “out of house” and gives real customers a chance to try the program with the set intention of catching any bugs or errors prior to it being fully released. Alpha testing is the phase that takes the longest and can sometimes last three to five times longer than Beta. However, Beta testing can be completed in just a few weeks to a month, assuming no major bugs are detected. Alpha testing is typically performed by engineers or other employees of the company while Beta testing occurs in the “real world”, temporarily being released to the public to get the widest range of feedback possible. During Alpha testing, it is common for there to be a good amount of bugs detected as well missing features. During Beta testing, there should be a big decrease in the amount of these problems. When testing in the Alpha phase is over, companies have a good sense of how the product performs. After Beta testing is complete, the company has a good idea of what the customer thinks and what they experienced while testing. If all goes well in both phases, the product is ready to be released and enjoyed by the public. The length of time and effort that is put forth in order for the world to enjoy and utilize the many programs on computers today is often overlooked. Information such as this gives the user a new appreciation for computers and computer programs.