What Is A Computer Program?

A computer program is a collection of instructions that performs a specific task when executed by a computer. Most computer devices require programs to function properly. A computer program is usually written by a computer programmer in a programming language. Once it is written, the programmer uses a compiler to turn it into a language that the computer can understand.

A computer program is stored as a file on the computer’s hard drive. When the user runs the program, the file is read by the computer, and the processor reads the data in the file as a list of instructions. Then the computer does what the program tells it to do.

What You Need To Know About Program

1. Program is a set of instructions written in a programming language used to execute for a specific task or particular function.
2. A program does not have further categorization.
3. A program cannot be software.
4. A program consists of a set of instructions which are coded in a programming language like c, C++, PHP, Java etc.
5. Programs do not have a user interface.
6. A program is developed and used by either a single programmer or a group of programmers.
7. A program is compiled every time when we need to generate some output from it.
8. Program has limited functionality and less features.
9. Program functionality is dependent on compiler.
10. A program takes less time to build/make.
11. Program development approach is un-procedural, un-organized and unplanned.
12. The size of a program ranges from kilobytes (Kb) to megabytes (Mb).

What Is A Computer Software?

Computer software popularly referred to as software, is a set of instructions, data or programs used to operate computers and execute specific tasks. It includes all programs on a computer such as applications and the operating system. Software is often divided into three categories:

* **Application software**. This is intended to perform certain tasks. Examples of application software include office suites, gaming applications, database systems and educational software.
* **Programming Software**. Programming software is a set of tools to aid developers in writing programs. The various tools available are compilers, linkers, debuggers, interpreters and text editors.
* **System Software.**System software act as a base for application software. Examples include device drivers, operating systems, compilers, disk formatters, text editors and utilities helping the computer to operate more efficiently. System software is usually written in C programming language.

What You Need To Know About Software

1. Software is a collection of several programs and other procedures and documentation.
2. Software can be categorized into three types: application software, system software and utilities.
3. Software can be a program.
4. Software consists of bundles of programs and data files. Programs in specific software use these data files to perform a dedicated type of tasks.
5. Every software has a dedicated user interface. The user interface of software may be in the form of a command prompt or in a graphical format.
6. Software is developed by either a single programmer or a group of programmers.
7. Whole software is compiled, tested and debugged during the development process.
8. Software has lots of functionality and features such as GUI, input/output data, process etc.
9. Software functionality is dependent on the operating system.
10. Software takes relatively more time to build/make when compared to program.
11. Software development approach is systematic, organized and very well planned.
12. The size of a software ranges from megabytes (Mb) to Gigabytes (Gb).
13. Examples of software include:  Microsoft Word, Microsoft Excel, VLC media player, Firefox, Adobe Reader, Windows, Linux, Unix, Mac etc.

**Software**is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

**Engineering** on the other hand, is all about developing products, using well-defined, scientific principles and methods.



**Software engineering** is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures. The outcome of software engineering is an efficient and reliable software product.

## Definitions

IEEE defines software engineering as:

(1) The application of a systematic,disciplined,quantifiable approach to the development,operation and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in the above statement.

Fritz Bauer, a German computer scientist, defines software engineering as:

Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

## Need of Software Engineering

The need of software engineering arises because of higher rate of change in user requirements and environment on which the software is working.

* **Large software -**It is easier to build a wall than to a house or building, likewise, as the size of software become large engineering has to step to give it a scientific process.
* **Scalability-**If the software process were not based on scientific and engineering concepts, it would be easier to re-create new software than to scale an existing one.
* **Cost-**As hardware industry has shown its skills and huge manufacturing has lower down he price of computer and electronic hardware. But the cost of software remains high if proper process is not adapted.
* **Dynamic Nature-**The always growing and adapting nature of software hugely depends upon the environment in which user works. If the nature of software is always changing, new enhancements need to be done in the existing one. This is where software engineering plays a good role.
* **Quality Management-**Better process of software development provides better and quality software product.

## Characteristics of good software

A software product can be judged by what it offers and how well it can be used. This software must satisfy on the following grounds:

* Operational
* Transitional
* Maintenance

Well-engineered and crafted software is expected to have the following characteristics:

### Operational

This tells us how well software works in operations. It can be measured on:

* Budget
* Usability
* Efficiency
* Correctness
* Functionality
* Dependability
* Security
* Safety

### Transitional

This aspect is important when the software is moved from one platform to another:

* Portability
* Interoperability
* Reusability
* Adaptability

### Maintenance

This aspect briefs about how well a software has the capabilities to maintain itself in the ever-changing environment:

* Modularity
* Maintainability
* Flexibility
* Scalability

In short, Software engineering is a branch of computer science, which uses well-defined engineering concepts required to produce efficient, durable, scalable, in-budget and on-time software products.

## The Importance of Software Engineers

Software engineers of all kinds, full-time staff, vendors, contracted workers, or part-time workers, are important members of the IT community.

What do software engineers do? Software engineers apply the principles of software engineering to the design, development, maintenance, testing, and evaluation of software. There is much discussion about the degree of education and or certification that should be required for software engineers.

According to StackOverflow Survey 2018, software engineers are lifelong learners; almost 90% of all developers say they have taught themselves a new language, framework, or tool outside of their formal education.  
<https://insights.stackoverflow.com/survey/2018/>

Software engineers are well versed in the software development process, though they typically need input from IT leader regarding software requirements and what the end result needs to be. Regardless of formal education, all software engineers should work within a specific set of best practices for software engineering so that others can do some of this work at the same time.  
Software engineering almost always includes a vast amount of teamwork. Designers, writers, coders, testers, various team members, and the entire IT team need to understand the code.

Software engineers should understand how to work with several common computer languages, including Visual Basic, Python, Java, C, and C++. According to Stackoverflow, for the sixth year in a row, JavaScript is the most commonly used programming language. Python has risen in the ranks, surpassing C# this year, much like it surpassed PHP last year. Python has a solid claim to being the fastest-growing major programming language.  
<https://insights.stackoverflow.com/survey/2018/>

Software engineering is important because specific software is needed in almost every industry, in every business, and for every function. It becomes more important as time goes on – if something breaks within your application portfolio, a quick, efficient, and effective fix needs to happen as soon as possible.

Whatever you need software engineering to do – it is something that is vitally important and that importance just keeps growing. When you work with software engineers, you need to have a check and balance system to see if they are living up to their requirements and meeting KPIs.

**Principles of Software Engineering**

**Separation of Concerns**

Separation of concerns is a recognition of the need for human beings to work within a limited context. As descibed by G. A. Miller [[Miller56](https://www.d.umn.edu/~gshute/softeng/bibliography.html#Miller56)], the human mind is limited to dealing with approximately seven units of data at a time. A unit is something that a person has learned to deal with as a whole - a single abstraction or concept. Although human capacity for forming abstractions appears to be unlimited, it takes time and repetitive use for an abstraction to become a useful tool; that is, to serve as a unit.

When specifying the behavior of a data structure component, there are often two concerns that need to be dealt with: basic functionality and support for data integrity. A data structure component is often easier to use if these two concerns are divided as much as posible into separate sets of client functions. It is certainly helful to clients if the client documentation treats the two concerns separately. Further, implementation documentation and algorithm descriptions can profit from separate treatment of basic algorithms and modifications for data integrity and exception handling.

There is another reason for the importance of separation of concerns. Software engineers must deal with complex values in attempting to optimize the quality of a product. From the study of algorithmic complexity, we can learn an important lesson. There are often efficient algorithms for optimizing a single measurable quantity, but problems requiring optimization of a combination of quantities are almost always NP-complete. Although it is not a proven fact, most experts in complexity theory believe that NP-complete problems cannot be solved by algorithms that run in polynomial time.

In view of this, it makes sense to separate handling of different values. This can be done either by dealing with different values at different times in the software development process, or by structuring the design so that responsibility for achieving different values is assigned to different components.

As an example of this, run-time efficiency is one value that frequently conflicts with other software values. For this reason, most software engineers recommend dealing with efficiency as a separate concern. After the software is design to meet other criteria, it's run time can be checked and analysed to see where the time is being spent. If necessary, the portions of code that are using the greatest part of the runtime can be modified to improve the runtime. This idea is described in depth in Ken Auer and Kent Beck's article "Lazy optimization: patterns for efficient smalltalk programming" in [[VCK96](https://www.d.umn.edu/~gshute/softeng/bibliography.html#VCK96), pp 19-42].

**Modularity**

The principle of modularity is a specialization of the principle of separation of concerns. Following the principle of modularity implies separating software into components according to functionality and responsibility. Parnas [[Parnas72](https://www.d.umn.edu/~gshute/softeng/bibliography.html#Parnas72)] wrote one of the eariest papers discussing the considerations involved in modularization. A more recent work, [[WWW90](https://www.d.umn.edu/~gshute/softeng/bibliography.html#WWW90)], describes a responsibility-driven methodology for modularization in an object-oriented context.

**Abstraction**

The principle of abstraction is another specialization of the principle of separation of concerns. Following the principle of abstraction implies separating the behavior of software components from their implementation. It requires learning to look at software and software components from two points of view: what it does, and how it does it.

Failure to separate behavior from implementation is a common cause of unnecessary coupling. For example, it is common in recursive algorithms to introduce extra parameters to make the recursion work. When this is done, the recursion should be called through a non-recursive shell that provides the proper initial values for the extra parameters. Otherwise, the caller must deal with a more complex behavior that requires specifying the extra parameters. If the implementation is later converted to a non-recursive algorithm then the client code will also need to be changed.

Design by contract is an important methodology for dealing with abstraction. The basic ideas of design by contract are sketched by Fowler and Scott [[FS97](https://www.d.umn.edu/~gshute/softeng/bibliography.html#FS97)]. The most complete treatment of the methodology is given by Meyer [[Meyer92a](https://www.d.umn.edu/~gshute/softeng/bibliography.html#Meyer92a)].

**Anticipation of Change**

Computer software is an automated solution to a problem. The problem arises in some context, or *domain* that is familiar to the users of the software. The domain defines the types of data that the users need to work with and relationships between the types of data.

Software developers, on the other hand, are familiar with a technology that deals with data in an abstract way. They deal with structures and algorithms without regard for the meaning or importance of the data that is involved. A software developer can think in terms of graphs and graph algorithms without attaching concrete meaning to vertices and edges.

Working out an automated solution to a problem is thus a learning experience for both software developers and their clients. Software developers are learning the domain that the clients work in. They are also learning the values of the client: what form of data presentation is most useful to the client, what kinds of data are crucial and require special protective measures.

The clients are learning to see the range of possible solutions that software technology can provide. They are also learning to evaluate the possible solutions with regard to their effectiveness in meeting the clients needs.

If the problem to be solved is complex then it is not reasonable to assume that the best solution will be worked out in a short period of time. The clients do, however, want a timely solution. In most cases, they are not willing to wait until the perfect solution is worked out. They want a reasonable solution soon; perfection can come later. To develop a timely solution, software developers need to know the requirements: how the software should behave. The principle of acticipation of change recognizes the complexity of the learning process for both software developers and their clients. Preliminary requirements need to be worked out early, but it should be possible to make changes in the requirements as learning progresses.

Coupling is a major obstacle to change. If two components are strongly coupled then it is likely that changing one will not work without changing the other.

Cohesiveness has a positive effect on ease of change. Cohesive components are easier to reuse when requirements change. If a component has several tasks rolled up into one package, it is likely that it will need to be split up when changes are made.

**Generality**

The principle of generality is closely related to the principle of anticipation of change. It is important in designing software that is free from unnatural restrictions and limitations. One excellent example of an unnatural restriction or limitation is the use of two digit year numbers, which has led to the "year 2000" problem: software that will garble record keeping at the turn of the century. Although the two-digit limitation appeared reasonable at the time, good software frequently survives beyond its expected lifetime.

For another example where the principle of generality applies, consider a customer who is converting business practices into automated software. They are often trying to satisfy general needs, but they understand and present their needs in terms of their current practices. As they become more familiar with the possibilities of automated solutions, they begin seeing what they need, rather than what they are currently doing to satisfy those needs. This distinction is similar to the distinction in the principle of abstraction, but its effects are felt earlier in the software development process.

**Incremental Development**

Fowler and Scott [[FS97](https://www.d.umn.edu/~gshute/softeng/bibliography.html#FS97)] give a brief, but thoughtful, description of an incremental software development process. In this process, you build the software in small increments; for example, adding one use case at a time.

An incremental software development process simplifies verification. If you develop software by adding small increments of functionality then, for verification, you only need to deal with the added portion. If there are any errors detected then they are already partly isolated so they are much easier to correct.

A carefully planned incremental development process can also ease the handling of changes in requirements. To do this, the planning must identify use cases that are most likely to be changed and put them towards the end of the development process.

**Consistency**

The principle of consistency is a recognition of the fact that it is easier to do things in a familiar context. For example, coding style is a consistent manner of laying out code text. This serves two purposes. First, it makes reading the code easier. Second, it allows programmers to automate part of the skills required in code entry, freeing the programmer's mind to deal with more important issues.

At a higher level, consistency involves the development of idioms for dealing with common programming problems. Coplien [[Coplien92](https://www.d.umn.edu/~gshute/softeng/bibliography.html#Coplien92)] gives an excellent presentation of the use of idioms for coding in C++.

Consistency serves two purposes in designing graphical user interfaces. First, a consistent look and feel makes it easier for users to learn to use software. Once the basic elements of dealing with an inteface are learned, they do not have to be relearned for a different software application. Second, a consistent user interface promotes reuse of the interace components. Graphical user interface systems have a collection of frames, panes, and other view components that support the common look. They also have a collection of controllers for responding to user input, supporting the common feel. Often, both look and feel are combined, as in pop-up menus and buttons. These components can be used by any program.

Meyer [[Meyer94c](https://www.d.umn.edu/~gshute/softeng/bibliography.html#Meyer94c)] applies the principle of consistency to object-oriented class libraries. As the available libraries grow more and more complex it is essential that they be designed to present a consistent interface to the client. For example, most data collection structures support adding new data items. It is much easier to learn to use the collections if the name add is always used for this kind of operation.

* **Difference Between S/w Programming and S/w Engineering**



* **Software Engineering Team Members**

[1 PROJECT SPONSOR](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#PROJECT_SPONSOR)

[2 SUBJECT MATTER EXPERTS (SME)](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#SUBJECT_MATTER_EXPERTS_SME)

[3 PRODUCT OWNER](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#PRODUCT_OWNER)

[4 PROJECT MANAGER (PM)](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#PROJECT_MANAGER_PM)

[5 TECHNICAL LEAD](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#TECHNICAL_LEAD)

[6 SOFTWARE DEVELOPERS](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#SOFTWARE_DEVELOPERS)

[7 SOFTWARE TESTERS](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#SOFTWARE_TESTERS)

[8 USER ACCEPTANCE TESTERS](https://www.atlascode.com/blog/software-development-project-roles-and-responsibilities/#USER_ACCEPTANCE_TESTERS)

**Software projects are difficult and they all take careful planning, a talented development team and collaboration of a project’s team members, both internally within the company and externally with the software development company.**

Software projects can only move forward when the key stakeholders are all in place.

One of the keys to a successful software project is identifying and documenting the software project roles and responsibilities for your project. You’ll need to ensure that you define the key stakeholders within your business that will be involved in the delivery of the software solution.

 Get the right people. Then no matter what all else you might do wrong after that, the people will save you. That’s what management is all about.   
*-– Tom DeMarco*

Among the key stakeholders of a software project are the following eight key roles in software development and their corresponding responsibilities.

**PROJECT SPONSOR**

Project Sponsors play a critical role in all projects. Project sponsors have the bandwidth to take on the Project Sponsor role, their day job and no other project role, therefore Project Sponsors are not Project Managers, Scrum Masters or Product Owners.

 Unengaged sponsor sinks the ship.   
*-– Angela Waner*

The Project Sponsor is **the person or group that provides direction and resources**, including financial resources for the software project. The Project Sponsor works with the project management team, aiding with wider project matters such as scope clarification, progress, monitoring, and influencing others in order to benefit the software project.

The Project Sponsor leads the project through the software supplier selection process until it is formally authorised. For issues that are beyond the control of the Product Owner, the Project Sponsor serves as an escalation path.

The Project Sponsor may also be involved in other important issues such as authorising changes in scope, phase-end reviews, and go/no-go decisions when the stakes of the project are particularly high.

Typically sponsors of projects tend to be senior management or director level executives.

**SUBJECT MATTER EXPERTS (SME)**

A Subject Matter Expert (SME) or Domain Expert is a person who is an authority in a particular area or topic. A Subject Matter Expert has superior (expert) knowledge of a discipline, technology, product, business process or entire business area.

The SME role and responsibilities in software development could be summarised as follows: **they are normally the people from whom technical requirements are captured**.

 If everyone is thinking alike, someone isn’t thinking.   
*-– General George Patton Jr.*

Subject Matter Experts are the accountants, finance controllers, salespeople, production managers and so on who roll up their sleeves each day. They know their roles inside and out and are rarely technical.

However, their lack of technical knowledge is their strength, as they are not bogged down in technicalities and instead are purely focused on business outcomes.

It’s imperative that discussions are held with Subject Matter Experts at the same time as the software product vision statement is being created. Feedback from this group of experts can save a lot of back and forth down the line.

However, given that Subject Matter Experts tend not to be technical the right amount and type of engagement are necessary so as not to overwhelm them. One of the ways to get them involved is to have them contribute to the creation of early-stage wireframes and prototypes.

**PRODUCT OWNER**

Product Owner is a software development role for **a person who represents the business or end-users and is responsible for working with the user group to determine what features will be in the product release**.

The Product Owner is also responsible for the prioritised backlog and maximising the return on investment (ROI) of the software project. Part of this role’s responsibility includes documenting user stories or requirements for the software project.

They act as the main point of contact for all decisions concerning the project and as such, need to be empowered to perform their responsibilities without the need to seek too much prior authorisation from the Project Sponsors.

Appointing the right person to this role, with the appropriate delegated authority to progress the project, is fundamental to the success of the project, especially if an agile methodology approach is undertaken.

In particular, the Product Owner is responsible for:

* ensuring that the software product vision statement is adhered to
* making the final decision on all scope related decisions
* maintaining and updating the product backlog on a continuous basis by
  + refining new requirements
  + removing requirements that fall out of scope
  + adding new requirements identified as being required to achieve the software product vision statement
  + reviewing and setting the priorities assigned to the product backlog and heading up all project planning meetings
* resolving any disputes either with the software development team or internally

Failure to have a Product Owner in place usually means that the software project will execute in fits and starts whilst the [**software developers**](https://www.atlascode.com/) are on hold waiting for crucial feedback.

A slowdown in the momentum of a software project can have long-term consequences, not least of missed milestones and deadlines. Don’t ever underestimate the importance of the Product Owner role in the success of your software development project.

**PROJECT MANAGER (PM)**

The Project Manager (PM) is **responsible for knowing the “who, what, where, when and why” of the software project**. This means knowing the stakeholders of the project and being able to effectively communicate with each of them.

The Project Manager is also responsible for creating and managing the project budget and schedule as well as processes including scope management, issues management and risk management.

Some of the Project Manager duties can include:

* Developing a software project plan
* Manage deliverables according to the software project plan
* Recruiting software project staff
* Leading and managing the software project team
* Determining the methodology used on the project
* Establishing a project schedule and determine each phase
* Assigning tasks to project team members
* Providing regular updates to senior management

It doesn’t matter if you are using an agile methodology or the waterfall method, once deliverables are defined, it is critical that the Project Manager starts to actively exercise change management. Change should not be perceived as negative or something to be avoided.

Change is inevitable and is acceptable in a software project as long as it is managed. The impact of any change needs to be assessed, measured and communicated. The major factors are typically timeline and budget. If the impact is deemed acceptable by the Project Sponsor, then the change can be incorporated.

The Project Manager also oversees [**software testing**](https://www.atlascode.com/services/software-testing/), delivery and formal acceptance by the customer. Then the Project Manager performs a project review with the software development team to document any lessons learned from the [**software development processes**](https://www.atlascode.com/process/).

**TECHNICAL LEAD**

This person **translates the business requirements into a technical solution**. Because of this responsibility, it is beneficial to have the Technical Lead involved in the planning phase to hear the business requirements from the customer’s point of view and ask questions.

The Technical Lead is the development team leader and works with the developers to provide technical details and estimates for the proposed solution. This information is used by the Project Manager to create the Statement of Work and the Work Breakdown Structure documents for the software project.

It is critical that the Technical Lead can effectively communicate the status of the software project to the Project Manager so that issues or variances can be effectively addressed as soon as possible.

The Technical Lead is also responsible for establishing and enforcing standards and practices with the software development team.

**SOFTWARE DEVELOPERS**

The Software Developers (front-end and back-end) are **responsible for using the technical requirements from the Technical Lead to create cost and timeline estimates**.

The Software Developers are also responsible for building the deliverables and communicating the status of the software project to the Technical Lead or Project Manager.

It is critical that the other team members effectively communicate the technical requirements to the Software Developers to reduce project risk and provide the software project with the greatest chance of success.

**SOFTWARE TESTERS**

The Software Testers ensure that the software solution meets the business requirements and that it is free of bugs, errors and defects.

In the test planning and preparation phases of the [**software testing**](https://www.atlascode.com/services/software-testing/), Software Testers should review and contribute to test plans, as well as be analysing, reviewing and assessing technical requirements and design specifications.

Software Testers are involved in identifying test conditions and creating test designs, test cases, test procedure specifications and test data, and may automate or help to automate the tests.

Some of the Software Testers duties can include:

* They often set up the test environments or assist system administration and network management staff in doing so
* As test execution begins, the number of testers often increases, starting with the work required to implement tests in the test environment
* Testers execute and log the tests, evaluate the results and document problems found
* They monitor the testing and the test environment, often using tools for this task, and often gather performance metrics
* Throughout the software testing life cycle, they review each other’s work, including test specifications, defect reports and test results

**USER ACCEPTANCE TESTERS**

You should expect your software solution provider to carry out a wide array of software testing to **ensure that your new software solution meets various quality assurance (QA) criteria**.

On from that, representatives of your company will need to perform the final checks to ensure that the software works for the business across a number of real-world scenarios.

User Acceptance Testing (UAT) is the final step prior to a new software solution being released to production (live). It’s absolutely essential that you have the resources to tackle user acceptance testing in a timely and organised fashion, as it is often UAT that creates the bottleneck between the software solution being completed and released to the business.

It’s often the case that the aforementioned Subject Matter Experts defined how the new software solution should work and, given their close proximity to the actual work, they can make excellent User Acceptance Testers.

 When end users get involved in the final stages of testing, light bulbs go on, and they often have an “aha” moment. Unfortunately, that is often too late.   
*-– Frank R. Parth*

It’s an excellent idea to ensure that those employees participating in UAT are brought in from the start, and understand, or perhaps better still contribute to, the design of the new software solution.

This emotional buy-in and understanding of the software solution’s objectives reduces the friction that might otherwise exist in attempting to move end-users from the existing software systems they know, love and use every day.

SOFTWARE DEVELOPMENT ROLES – CONCLUSION

Whilst it’s important that your software solution provider has the necessary resources in place to operate your project, it is equally as important that you as the customer understand the roles and responsibilities required within your team to bring your project to successful completion.

**The key to project success is clear and effective communication**. A critical portion of this communication is identifying the stakeholders and their roles.

Whatever labels you apply to the software project roles above, clear communication of expectations and status to the stakeholders throughout the life of the software project will increase the chances of your project’s success.

ERP

ishan

Jyoti Diwanshi Hemant specialization

Notices

Marks

Att

Report

uploading

Report

Att marking

Amit Sumit Raj Vikas