

Computer Fundamentals and Applications

Unit 2: Computer Software

(3 Hours)

Course Code: CACS101 | Credit: 4 Cr. | Year/Semester: I / I

Class Load: 8 Hrs./Week (Theory: 4 Hrs., Practical: 4 Hrs.)

Contents

1. Introduction to Software

1.1 What Is Software?

1.2 The Software Development Life Cycle (SDLC)

1.3 Software Licensing Models

2. Types of Software

2.1 System Software

2.2 Application Software

2.3 Utility Software

2.4 Programming Software and Language Translators

3. Program vs. Software

3.1 What Is a Program?

3.2 What Is Software?

3.3 Key Differences

3.4 How Programs Combine into Software

4. Computer Virus and Antivirus

4.1 What Is a Computer Virus?

4.2 The Virus Life Cycle

4.3 Types of Computer Viruses

4.4 Other Common Malware

4.5 Symptoms of Infection

4.6 Antivirus Software and How It Works

4.7 Safe-Computing Practices

Key Takeaways

1.1 What Is Software?

Software is the set of instructions, or programs, that tells computer hardware what to do and how to do it. While hardware provides the physical components of a computer, such as the processor, memory, and peripherals, it is software that gives these components purpose, directing them to perform useful tasks such as writing a document, browsing the internet, or managing a database. Without software, even the most powerful hardware is simply an inert collection of electronic circuits.

Software and hardware exist in a dependent relationship: hardware supplies the processing power and physical resources, while software supplies the logic and instructions that put those resources to work. Software is typically stored on a storage device as a set of binary instructions and is loaded into memory to be executed by the processor. Because software can be changed, updated, or replaced far more easily than hardware, it is what gives a computer its flexibility, allowing the same physical machine to serve completely different purposes depending on which programs are installed and run.

Software vs. Hardware at a Glance

Aspect	Hardware	Software
Nature	Physical, tangible components	Logical instructions and data
Examples	CPU, RAM, keyboard, monitor	Operating system, applications, drivers
Change	Difficult and costly to modify	Can be updated or replaced easily
Failure	Wears out or breaks physically	Fails through bugs or corruption, not wear

Table 1.1 — How software and hardware differ, despite depending on each other.

Example 1.1 — The Same Hardware, Different Purposes

A single laptop can serve as a word processor, a gaming machine, a video-editing workstation, or a programming environment, without any change to its physical hardware.

The only thing that changes between these uses is the software installed and running at the time, showing clearly why software, not hardware, defines what a computer can actually do for its user.

bcalink.com

1.2 The Software Development Life Cycle (SDLC)

Software is not written in a single step; it is produced through a structured sequence of stages known as the Software Development Life Cycle. Following a defined life cycle helps ensure that the finished software actually meets the needs it was built for, works reliably, and can be maintained after release, rather than being assembled in an ad hoc way that is difficult to test or fix later.

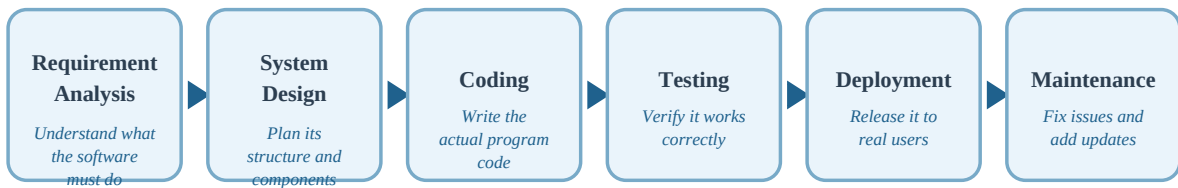


Figure 1.1 — The six stages of the Software Development Life Cycle (SDLC).

In the requirement analysis stage, developers work with stakeholders to understand exactly what the software must accomplish. The system design stage then plans the software's internal structure, including its data organization and the programs that will make it up, before implementation, where the design is translated into actual program code. Testing verifies the software behaves correctly and catches defects before they reach users, deployment releases the finished software for real-world use, and maintenance keeps it working correctly and up to date long after its initial release.

1.3 Software Licensing Models

Once software is built, it is distributed under a licence that determines how it may be used, copied, or modified. Proprietary software is owned by a company and sold or licensed to users with restrictions on copying or altering the code; freeware is distributed at no cost but keeps its source code private; shareware is offered free on a trial basis before requiring payment for continued use; and open-source software makes its underlying source code freely available for anyone to inspect, modify, and redistribute.

Licence Type	Cost to User	Source Code Access
Proprietary	Usually paid	Closed — owned and protected by the vendor
Freeware	Free	Closed — free to use, but not to modify

Licence Type	Cost to User	Source Code Access
Shareware	Free trial, then paid	Closed — full features often unlock after payment
Open-Source	Usually free	Open — code may be inspected, modified, and shared

Table 1.2 — Common software licensing models and what each grants the user.

Example 1.2 — Choosing a Licensing Model

A school with a limited budget needs office software for its computer lab. Rather than paying for a proprietary suite, it installs a free, open-source office suite whose source code is publicly available.

Because the licence permits redistribution, the school's IT staff can legally install it on every lab computer at no extra cost, and even customize it slightly for the school's needs, something a closed proprietary licence would not normally allow.

Software is broadly classified into four categories based on the role it plays: system software, application software, utility software, and programming software. These categories are also organized hierarchically in terms of how close they sit to the hardware. System software sits closest to the hardware and loads first when a computer starts; utility software typically runs alongside or within the operating system to keep the machine healthy; and application software sits at the top of the stack, relying on the operating system to grant it access to memory, storage, and peripherals whenever the user runs it.

Category	Role	Examples
System Software	Manages hardware and provides a platform for other software	Operating system, device drivers, firmware
Application Software	Helps the user perform a specific task	Word processor, spreadsheet, web browser
Utility Software	Maintains, configures, and optimizes the computer	Antivirus, disk cleanup, file compression tools
Programming Software	Provides tools to write and test other software	Compilers, code editors, debuggers

Table 2.1 — The four major categories of software classified by role.

2.1 System Software

System software manages and controls a computer's own hardware and resources so that other programs can run reliably on top of it. The operating system is the most important example, and it is responsible for several core functions: process management, which decides which program gets to use the processor and when; memory management, which allocates RAM to running programs and reclaims it when they close; file management, which organizes and controls access to files on storage devices; and device management, which coordinates communication with hardware such as printers, keyboards, and network adapters.

Alongside the operating system, device drivers act as translators that let the operating system communicate with a specific piece of hardware, while firmware is a permanent form of system software written directly onto a hardware device, such as the startup instructions stored in a computer's BIOS or UEFI chip.

OS Function	What It Does
Process Management	Decides which program uses the CPU, and for how long
Memory Management	Allocates and reclaims RAM for running programs
File Management	Organizes, names, and controls access to stored files
Device Management	Coordinates communication with printers, drives, and other hardware

Table 2.2 — Core functions performed by the operating system, the primary system software.

2.2 Application Software

Application software is designed to help the user accomplish a specific task rather than to manage the computer itself. It is commonly divided into general-purpose application software, which is flexible enough to be used across many different jobs, such as a word processor or spreadsheet, and specialized application software, which is built for one specific organizational purpose, such as a hospital patient-management system or a payroll processing package.

Application Type	Description	Example
General-Purpose	Flexible software usable for many different tasks	Word processor, spreadsheet, web browser
Specialized/Custom	Built for one specific organizational task	Hospital patient-record system, airline booking system

Table 2.3 — General-purpose versus specialized application software.

2.3 Utility Software

Utility software does not perform user-facing tasks in the way application software does; instead, it helps maintain, configure, and optimize the computer so that everything else continues to run smoothly. Common examples include antivirus programs that guard against malware, disk cleanup tools that remove unnecessary files, disk defragmenters that reorganize stored data for faster access, backup utilities that copy important files to a safe location, and file compression tools that reduce file size for storage or transfer.

Utility Category	Purpose
Antivirus/Security	Detects and removes malware, blocks unauthorized access
Disk Cleanup	Removes temporary and unnecessary files to free up space
Disk Defragmenter	Reorganizes fragmented data for faster access
Backup Utility	Copies important files to a safe secondary location
File Compression	Reduces file size for easier storage or transfer

Table 2.4 — Common categories of utility software and what each maintains.

Example 2.1 — Types of Software in One Session

A student turns on a laptop, and the operating system, a piece of system software, loads first and manages the hardware. The student then opens a word processor, a general-purpose application software, to write an assignment.

Midway through, a disk cleanup utility runs automatically in the background to clear temporary files, and before shutting down, an antivirus utility performs a quick scan, illustrating how several categories of software cooperate within a single ordinary session.

2.4 Programming Software and Language Translators

Programming software provides the tools programmers use to write, test, and debug the system, application, and utility software described above. This includes code editors and Integrated Development Environments (IDEs) for writing code, and debuggers for finding and fixing errors. A particularly important category of programming software is the language translator, which converts human-readable source code into the machine code a processor can actually execute.

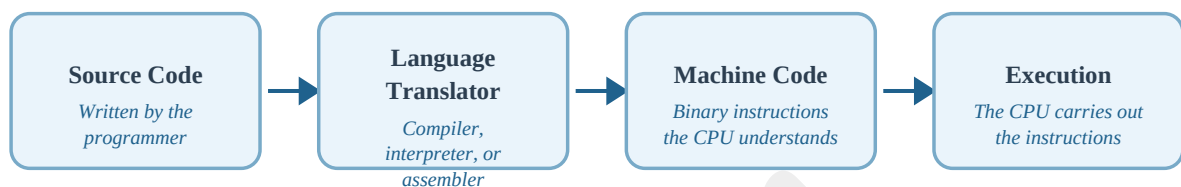


Figure 2.1 — How a language translator converts source code into an executable program.

A compiler translates the entire source code into machine code in one pass before the program is run, producing a standalone executable file that can be run repeatedly without re-translating the code each time. An interpreter instead translates and executes source code line by line as the program runs, which makes testing faster but generally results in slower overall execution than a compiled program. An assembler performs a similar translation, but specifically for assembly language, a low-level language that maps closely to a processor's own machine instructions.

Translator	How It Works	Trade-off
Compiler	Translates the whole program at once, before running it	Faster to run, but must fully re-translate after any change
Interpreter	Translates and runs code line by line	Easier to test, but generally slower to run
Assembler	Translates assembly language into machine code	Very close to hardware, but tedious to write

Table 2.5 — The three main types of language translator used in programming software.

3.1 What Is a Program?

A program is a single, self-contained set of instructions written to perform a specific task, such as a script that sorts a list of numbers or a small tool that converts one file format into another. A program is usually written by a programmer in a programming language, translated into machine code by a compiler or interpreter, and can often be understood, tested, and modified as one coherent unit of logic.

Because a program is a single unit, it can be quite small: a few lines of code that automate one repetitive task is still a complete program, provided it can be run and produces a defined result. What matters is not the size of the code but the fact that it forms one self-contained set of instructions aimed at a single, well-defined purpose.

3.2 What Is Software?

Software, by contrast, is a broader term that can refer to a single program or to a complete package of many programs, along with the data, configuration files, and documentation bundled with them, all working together to deliver a complete product. A useful way to see the distinction is that a program is one of the building blocks from which software is constructed.

A word-processing application, for example, is a piece of software made up of many individual programs and modules working together: one program may handle spell-checking, another may handle printing, and another may manage file saving, all coordinated under the single software product the user installs and opens. This distinction also matters for development and maintenance: a single program can often be written, tested, and modified by one person in isolation, whereas software, being composed of many interdependent programs, usually requires a coordinated development process involving version control, integration testing, and formal release cycles.

3.3 Key Differences

Aspect	Program	Software
Scope	A single set of instructions for one task	A complete product, often made of many programs
Independence	Can be a small, standalone script	Usually includes data, configuration, and documentation
Development	Often written and tested by one person	Usually requires coordinated, team-based development
Example	A script that renames a batch of files	A full operating system or office suite

Table 3.1 — Distinguishing a single program from a complete piece of software.

3.4 How Programs Combine into Software

In practice, most software a user installs is not one program but a coordinated collection of many smaller programs, each responsible for one piece of functionality, all packaged and presented to the user as a single application. The diagram below illustrates this using a typical office word processor as an example.

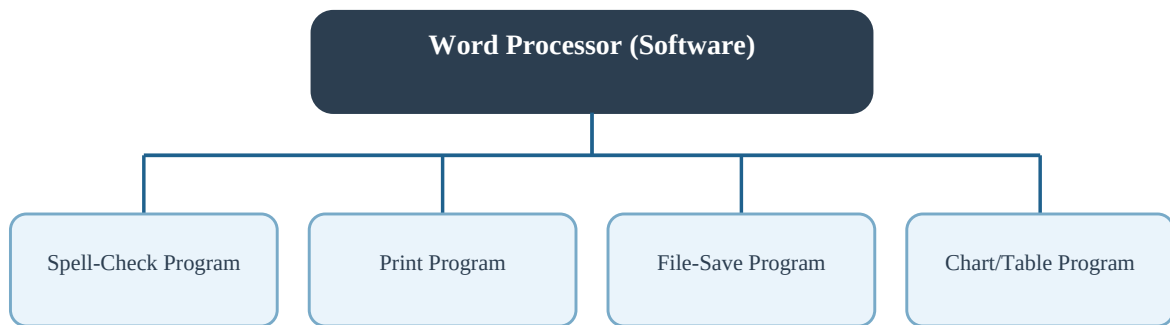


Figure 3.1 — A single piece of software built from several cooperating programs.

Each of the four programs shown handles one narrow responsibility, and none of them would be very useful to an ordinary user on its own. It is only when they are combined, packaged together, and given a shared interface that the result becomes the complete, recognizable piece of software the user opens and interacts with.

Example 3.1 — One Piece of Software, Many Programs

A spreadsheet application is a single piece of software, but internally it relies on many separate programs: one that recalculates formulas, one that renders charts, and one that handles printing.

The user experiences all of this as a single product, but from a technical standpoint, the software is an organized collection of individual programs working together toward a common purpose.

4.1 What Is a Computer Virus?

A computer virus is a type of malicious software, or malware, that attaches itself to a legitimate program or file and replicates itself when that host program is run, spreading to other files, programs, or computers in the process. The term is deliberately biological: like a biological virus, a computer virus cannot act on its own, it needs a host to carry it and an execution event, such as opening a file or running a program, to activate and begin spreading.

Viruses are written with deliberate intent, and their effects range from mildly disruptive, such as displaying unwanted pop-up messages, to severely destructive, such as deleting files, corrupting the operating system, or silently opening a backdoor that lets an attacker access the infected machine remotely. Because a virus depends on human action to spread, most infections originate from an unsafe user action: opening an unknown email attachment, running software from an untrusted source, or inserting an infected external drive.

4.2 The Virus Life Cycle

Regardless of the specific type of virus, nearly every virus passes through the same four-phase life cycle from the moment it reaches a system to the moment it causes visible harm. Understanding this cycle helps explain why some infections stay hidden for long periods before any damage becomes apparent.

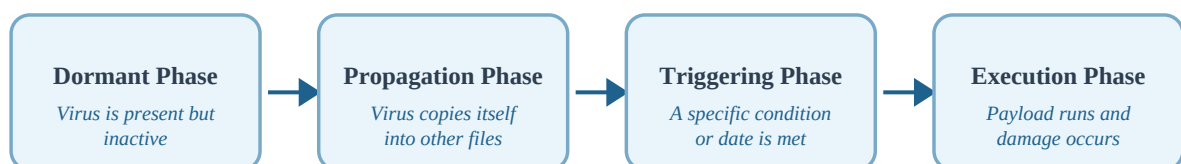


Figure 4.1 — The four-phase life cycle common to most computer viruses.

In the dormant phase, the virus sits inactive inside its host file, waiting for an event such as the program being opened. During the propagation phase, it places copies of itself into

other programs, files, or areas of disk, spreading its reach across the system. The triggering phase occurs when a predefined condition, such as a specific date, a number of executions, or a particular user action, is met, after which the execution phase begins and the virus's payload, its actual destructive or disruptive function, is carried out.

4.3 Types of Computer Viruses

Viruses are classified according to where they hide and how they infect a system. A boot sector virus infects the boot sector of a storage device, the small area read first when a computer starts, allowing the virus to load before the operating system itself and making it especially difficult to remove. A file infector virus attaches itself to executable program files and activates whenever the infected program is run, while a macro virus is written in the macro language of an application such as a word processor or spreadsheet and spreads through infected documents rather than executable programs.

A polymorphic virus changes its own code slightly every time it replicates, making it harder for signature-based antivirus tools to recognize. A resident virus installs itself in a computer's memory and can infect files continuously in the background even after the original host program has closed, whereas a non-resident virus only acts while its host program is actively running. A multipartite virus combines more than one of these techniques, for example infecting both the boot sector and executable files, making it spread faster and making full removal more difficult.

Virus Type	Where It Hides	Key Trait
Boot Sector Virus	Boot sector of a disk or USB drive	Loads before the operating system itself
File Infector Virus	Executable program files (.exe)	Activates whenever the infected program runs
Macro Virus	Macro-enabled documents/spreadsheets	Spreads through documents, not just programs
Polymorphic Virus	Any host, code changes each copy	Evades simple signature-based detection

Virus Type	Where It Hides	Key Trait
Resident Virus	System memory (RAM)	Keeps infecting files after the host program closes
Multipartite Virus	Multiple locations at once	Combines several infection techniques

Table 4.1 — Common types of computer virus, classified by infection technique.

bcalink.com

4.4 Other Common Malware

Viruses are only one member of the broader malware family. A worm is similar to a virus but does not need a host file or human action to spread; it exploits network vulnerabilities to copy itself from machine to machine automatically, often consuming bandwidth and system resources rapidly as it spreads. A trojan horse disguises itself as legitimate, useful software to trick a user into installing it, then secretly performs a malicious function such as opening remote access for an attacker, without the self-replication typical of a virus or worm.

Ransomware encrypts a victim's files and demands payment in exchange for the decryption key, effectively holding the victim's own data hostage. Spyware secretly monitors user activity, such as keystrokes or browsing history, and transmits it to an attacker, while adware floods the user with unwanted advertisements, and a rootkit hides deep within the operating system to conceal the presence of other malware and maintain long-term, privileged access for an attacker.

Malware Type	How It Spreads	Typical Harm
Worm	Spreads automatically across networks	Consumes bandwidth, may deliver further payloads
Trojan	Disguised as legitimate software	Opens hidden access for an attacker
Ransomware	Infected files, links, or exploited vulnerabilities	Encrypts data and demands payment for its release
Spyware	Bundled with software or hidden downloads	Secretly monitors and steals user activity/data
Adware	Bundled with free software	Floods the system with unwanted advertisements
Rootkit	Exploits deep system vulnerabilities	Hides other malware and grants persistent access

Table 4.2 — Malware types beyond the traditional virus, and how each behaves.

4.5 Symptoms of Infection

Because malware is designed to operate quietly for as long as possible, infections are often noticed only through indirect symptoms rather than an explicit warning. Common warning signs include a sudden and unexplained drop in system performance, programs crashing or opening on their own, files that go missing or become inaccessible, unexpected pop-up advertisements, a noticeably increased volume of network activity, and security software that has been silently disabled without the user's action.

Warning Signs Worth Investigating

A laptop that suddenly runs much slower than usual, shows unfamiliar programs in the startup list, or has its antivirus mysteriously turned off should be treated as a likely infection rather than a routine performance issue, and scanned immediately with an up-to-date antivirus tool.

4.6 Antivirus Software and How It Works

Antivirus software is designed to detect, block, and remove viruses and other malware before they can cause harm. Modern antivirus products rarely rely on a single technique, instead layering several detection methods so that threats missed by one layer are more likely to be caught by another.

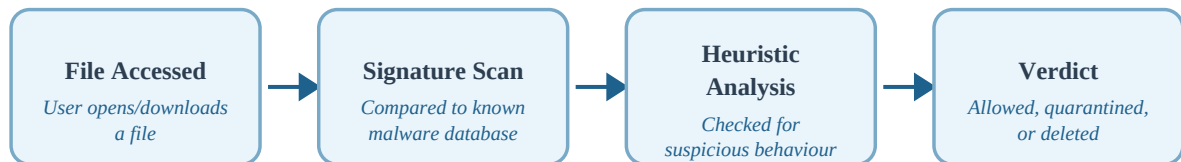


Figure 4.2 — A simplified antivirus detection workflow from file access to verdict.

Signature-based detection compares a file against a continuously updated database of known malware fingerprints, and is fast and highly accurate for threats that have already been identified and catalogued elsewhere, but it cannot recognize a brand-new threat whose signature is not yet in the database. Heuristic and behaviour-based detection instead watches how a file or program actually behaves, flagging actions such as attempting to modify system files or replicate itself, which allows antivirus software to catch previously unseen malware at the cost of occasionally flagging legitimate software by mistake.

Many modern antivirus products add a third layer, sandboxing, in which a suspicious file is executed inside an isolated, contained environment separate from the real system so that its behaviour can be observed safely before it is ever allowed to run on the user's actual machine. Real-time protection keeps all of these detection methods running continuously in the background, scanning files as they are opened, downloaded, or modified, rather than waiting for the user to run a manual scan.

Detection Method	How It Works	Strength / Limitation
Signature-Based	Matches files against a database of known malware	Fast and accurate, but blind to brand-new threats
Heuristic/Behaviour-Based	Flags suspicious actions and code patterns	Catches unknown malware; can raise false alarms

Detection Method	How It Works	Strength / Limitation
Sandboxing	Runs suspicious files in an isolated test environment	Safely reveals hidden behaviour before real execution
Real-Time Protection	Continuously scans files as they are accessed	Blocks threats immediately, uses more system resources

Table 4.3 — The main detection methods combined inside modern antivirus software.

Example 4.1 — Antivirus Detection in Practice

An employee receives an email attachment that secretly contains a virus disguised as an invoice document. When the attachment is opened, real-time protection immediately scans the file: its signature does not match any known virus, since it is a newly modified variant.

The heuristic engine notices that the file is attempting to modify system startup settings, a behaviour typical of malware, and automatically quarantines it before it can cause any damage, illustrating why layered detection catches threats that signature-matching alone would miss.

4.7 Safe-Computing Practices

Antivirus software is essential, but it is only one part of a complete defence against malware. Good user habits reduce the chance of infection in the first place and limit the damage even if a threat does slip through automated detection.

- **Keep software updated:** Install operating system and application updates promptly, since many viruses exploit known, already-patched vulnerabilities.
- **Update virus definitions regularly:** An antivirus program with outdated definitions cannot recognize the newest threats.
- **Avoid untrusted attachments and links:** Do not open email attachments or click links from unknown or unexpected senders.
- **Download software from official sources:** Pirated or third-party software is a common vector for hidden malware.
- **Back up important data regularly:** Regular backups, kept disconnected from the main system, limit the damage ransomware can cause.
- **Use a firewall alongside antivirus:** A firewall blocks unauthorized network access that some malware relies on to spread or communicate.

Unit 2 — Key Takeaways

- Software is the set of instructions that gives otherwise inert hardware its purpose, and is built through the SDLC stages of requirement analysis, design, implementation, testing, deployment, and maintenance.
- Software is distributed under licensing models — proprietary, freeware, shareware, and open-source — that determine how it may be used, copied, or modified.
- Software is classified into system, application, utility, and programming software, layered from closest to the hardware to closest to the user.
- System software, led by the operating system, handles process, memory, file, and device management so that other software can run reliably.
- Application software splits into general-purpose and specialized tools, utility software maintains and protects the system, and programming software includes the compilers, interpreters, and assemblers that turn source code into machine code.
- A program is a single set of instructions for one task, while software is usually a complete product built from many cooperating programs, as shown by the composition of a typical office application.
- A computer virus moves through a dormant, propagation, triggering, and execution life cycle, and appears in forms such as boot sector, file infector, macro, polymorphic, resident, and multipartite viruses.
- Worms, trojans, ransomware, spyware, adware, and rootkits are related malware types that spread and cause harm differently from a classic virus.
- Antivirus software layers signature-based, heuristic/behaviour-based, sandboxing, and real-time protection to detect both known and previously unseen threats.
- Safe-computing habits — updates, cautious downloading, and regular backups — remain essential alongside antivirus software itself.

— End of Unit 2: Computer Software —