Engineer

Software

the programs and data a computer runs

What it is

Software is every part of a computer system that can be changed by writing instead of by soldering: the programs, the data they carry, and the settings that tell them how to behave. The word is a mass noun in English, so a single item is “a piece of software” or “a program”, and something a person opens to do a job is usually called an application. “Code” tends to mean the source a developer edits, and specifications sometimes abbreviate the whole thing to SW.

Its usual opposite is hardware, the physical machine, with firmware in the gap between them. The NIST glossary draws that line through change: software is “computer programs and associated data that may be dynamically written or modified during execution”, while firmware is the kind that cannot be modified while it runs. A laptop’s operating system is software; the code in its keyboard controller is firmware; the keyboard is hardware.

Inside software there are layers. System software, such as an operating system or a device driver, manages the machine. Application software does something for a person: a spreadsheet, a booking site, a messaging client. Between the two sit libraries and packages, ready-made pieces that both kinds are assembled from, and in practice most of the code in a modern product arrived that way rather than being written for it.

One more distinction decides a lot later on. Source code is what people write and read; a compiler or an interpreter turns it into instructions a processor executes. Both are software, but they are not interchangeable when you buy it. A delivery of compiled files alone is something only its supplier can change.

Use the concept in a decision

Put “What is software?” to work in a real decision. We can help relate the concept to the system you are building or buying. The shape of a system worked out on paper first, so the parts fit and the cost is known before anyone writes code.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. One infrastructure plan for 14 departments — 14 research departments on one infrastructure plan

One infrastructure plan for 14 departments Get in touch →

Why it matters

Anyone commissioning, buying or renting software is choosing who carries its upkeep. The thing that makes software attractive, that copying it costs nothing and changing it after delivery is cheap next to changing a machine, is also why it is never finished. The components it was built from publish fixes. Operating systems and browsers retire interfaces. A flaw is disclosed in a library nobody on the team remembers choosing. Each of those turns into work for whoever owns the code.

The common routes divide that work differently:

- **Rented as a service.** The vendor runs the updates and the servers. You inherit their release schedule, their pricing and the effort of leaving, and your data lives on their side.
- **Licensed and run yourself.** You choose when to upgrade, and you also do the upgrading, the backups and the security patches.
- **Built for you.** It fits the process instead of the other way round, and its maintenance budget is yours for as long as it runs. The cost that decides whether that was wise comes after launch.
- **Open source.** Under a licence that meets the Open Source Definition you may use, study, change and redistribute it. Somebody still has to apply the updates; free to obtain is a different property from free to operate.

Whichever route you take, two questions about the result are worth asking before signing. What do you receive at the end: the source, the instructions to build it, and a list of what is inside it? And how will the supplier keep it secure? NIST’s Secure Software Development Framework exists partly for that second conversation: it gives buyers and suppliers a shared vocabulary for development practices. For the first, an NTIA report defines a software bill of materials as “a

formal record containing the details and supply chain relationships of various components used in building software". Asking for one is a cheap way to see how much of what you are paying for was written by someone else.

What it looks like in practice

Knowing what shipped is the part teams skip. On one platform the security work was moved from opinion into the build: in a continuous security programme, a software bill of materials is generated on every build with Anchore and Syft, Dependabot watches 6 ecosystems, CodeQL and ZAP analyse the code and the running instance, and even the outside steps of the build are frozen: each third-party workflow Action is referenced by its commit hash, which nobody can shift the way a tag can be shifted. Next to the automation sit 21 written security audits, including a threat model. A vulnerability recognised by the configured checks can fail the build. Its price is the volume of findings, which needs triage that never ends.

Getting software from a change to production is the other half. Where releases had depended on someone remembering a sequence of manual steps, and doing it slightly differently each time, automating build, test and deployment saved an estimated 4,000 hours and made releases faster and more reliable. The gain came from the path being identical every time rather than from anyone being more careful.

And the line between software and hardware is a practical one on a repair bench. Laptop repairs covered screens, hinges, keyboards and motherboards, and software problems across Linux, Windows and Mac. A laptop that will not power on could be the charger, the board or the battery, so the work started with finding the real fault before touching anything.

Where it goes wrong

An advisory turns into a week of archaeology

A vulnerability is announced in a widely used component, and nobody can say which of the organisation's systems contain it. The inventory only ever existed in people's heads and in lock files nobody reads, so it has to be reconstructed under pressure, system by system. Generate a bill of materials as part of every build and keep it with the release it describes. The question "are we affected?" then becomes a search.

Releases depend on who does them

The deploy works when one person runs it and breaks when another does, and the difference turns out to be a step that lives in somebody's memory. Each release is a small improvisation. Put the whole path into an automated job that builds, tests and deploys the same way every time, and treat any manual step that remains as a defect to remove.

The budget stops at launch

Two or three years in, every update feels risky, the framework is several major versions behind, and the only proposal on the table is a rewrite. The software was paid for like a building and then left alone, while everything around it kept moving. Keep a standing allowance for maintenance, upgrade dependencies in small regular steps while each step is still small, and expect the platforms underneath to change on their own schedule.

A name stays fixed while the code behind it changes

The same build produces different results on two days with no commit in between. Somewhere a reference points to a moving target: a version range, a `latest` tag, a CI action referenced by a tag that its owner can re-point. Pin third-party code to an exact version or, where tags can move, to a commit hash, commit the lock file, and let a bot propose the updates so that every change arrives as a reviewable diff.

Further reading

- NIST CSRC Glossary: software — the definitions NIST publications use, including the one that separates software from firmware.
- NIST SP 800-218, Secure Software Development Framework — a core set of practices for building software with fewer vulnerabilities, written so that buyers can use it with suppliers.
- NTIA: The Minimum Elements for a Software Bill of Materials — what a list of the components inside a piece of software has to record to be useful.
- The Open Source Definition — the ten criteria a licence must meet before software released under it counts as open source.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is software?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/software/