Criteria for the selection of software

Inhaltsverzeichnis

1. Introduction

NOTE This is a translation from the original German text by the original author. Suggestions for translation-improvements are welcome.

This article is intended to help you identify critera of good software.

It is primarily targeted at individuals who want to reflect and review their own user behaviour. It is an essay based on my personal experience. Unlike ISO/IEC 9126 it is not a formal document. It is an essay based on my personal experience. No systematic tests or tests or surveys were made.

I would be happy if users would **actively** choose a program. Did you choose an office program or was it given to you in school and you have been using it ever since? Does the programs you use depend on what was installed on your computer when you bought it?

The prerequisite is to reflect on one's own wishes and demands. **First** you have to know what you want: What is the problem, what is the purpose? Only **then** you can choose the program. Otherwise it happens quickly that one chases after the promises of advertisement, decides based on what apparently "everyone" seems to be using. Then you end up with programs that are too weak in too many of our criteria or you get caught in the vendor lock-in.

For this purpose several categories will be discussed and analyzed individually. It will be shown how a program in these categories performs well or poorly. At the end, the reader hopefully gets more clarity, what is a good program for him or her and what to look for when choosing a new one.

1.1. The perfect computer program

The perfect computer program does not exist, there are always compromises: Maybe a 3D graphics program renders in perfect image quality, but it takes many times longer than a program that does produces only average results. Maybe writing in a word processor is distraction-free and comfortable, but it can't export to PDF. Or it does exactly what you want it to do, but crashes regularly and you have to get in the habit to press Ctrl+S every 30 seconds. Maybe the program is not available for your hardware/system.

For everyone, the end of the willingness to compromise is reached at some point. A program that performs all tasks reliably and quickly, but spies on every step and sends the contents of one's hard drive over the Internet should be out of the question and not be used by anyone.

The terms "program", "application" and "software" are used synonymously. They refer to executable computer programs that perform a task or solve a problem. This includes productive software, as well as entertainment. The program can run on any system: General-purpose computers such as desktops and laptops, telephones, tablets, or the firmware of a refrigerator …

2. Criteria

After reflecting on my own software selection, I found the following criteria:

- Effectiveness: The program does what I want, what it promised, and does not crash. No security issues.
- Efficiency: Uses system resources economically and works fast.
- Robustness: Handles many system configurations and inputs, even errors.
- File format: Open, publicly specified file format, preferably text-based standard format: interoperability and archivability for decades.
- User interface: Fast and convenient to use, no unnecessary steps required from humans, good learning curve even for complex requirements. No "dumbing down". Results in the beginning, control in the end.
- Flexibility: Can be used for many tasks, even beyond the developer's intended purpose. Can work well with other programs. License does not artificially restrict usage.
- Scalability: Suits different sizes of projects, or is clearly designed for a defined size.
- Availability: Runs on many computer architectures and operating systems. Now and in the

future.

- Price: As cheap as possible, without developers making a loss.
- Privacy: Does not send anything to third parties without being asked, only reads its own data. Asks for permission.

Consciously or not, these criteria can be examined and evaluated individually in order to finally decide whether to learn or use a program.

2.1. Effectiveness or Functionality

The program does what you want to use it for, and what it is supposed to do. It is error-free: It has no logical errors, for example it does not miscalculate math equations, does not play the note F# when I have typed Db or marks a word during spell check that is spelled correctly. But it also has no bugs: it doesn't crash, saves all data completely, and readable again, etc. Security vulnerabilities are also considered bugs in the program.

Effectiveness is evaluated in relation to what the program promises: either explicitly in a text, for example on a web page, or implicitly through the design of the graphical user interface (GUI), by following established conventions: If I see a white rectangle on the screen with a blinking line in the upper left corner, I assume that I can type in text.

A related concept is "features." In general, you want your software to do as much as possible; as your needs and abilities grow, you should still be able to use it. This is where a fundamental problem with software selection shows up for the first time in this article: Software that offers too much becomes too complex. The user experience suffers, from an overly steep learning curve to unusability; and behind the scenes, the program becomes poorly auditable and maintainable, creating sources of potential bugs, crashes, and data loss.

Whether, and at what point, the balance of these criteria becomes disproportionate and negative aspects are no longer tolerable, or positive aspects are so important that they make all others of secondary importance, is what the rest of this article is intended to discuss.

2.2. Efficiency

The program works fast and does not waste computer resources. Thus it also consumes less energy (power/Watt). A program with the wrong algorithms can spend significantly more time on the same task, and is thus not efficient.

Memory consumption is also a problem. A text editor that needs 1 gigabyte of RAM to display 20 kilobytes of text is wasteful. 8 MB of data volume for a web page with a cake recipe is just rude.

Good programming leads to efficient software. This often results from patience and passion of the programmers to continue working on an already functional program (in the sense of effectiveness). Also knowledge and experience of the programmers are helpful, e.g. in the selection of algorithms, program libraries such as GUI Toolkits or game engines.

Ultimately, inefficient software is "selfish." Even if a task is still solved in a time acceptable to humans, it will block other programs and processes in the system.

Another type of efficiency is a good user interface, which is discussed in a chapter below.

2.3. Robustness

The application is resistant to accidents and user input errors.

Robustness originates in the non-visible part of the application. Here, strict attention must be paid to ensure that exactly the right thing is called and saved. If, for example, the function "spell-correction for a word" is called, but the number 2301 is entered, the look-up and comparison with the dictionary will fail. A robust application will report an error in an orderly fashion (all internally) and ensure that nothing is done at all, rather than something wrong. In this example, nothing would happen at all in a GUI, the number would be skipped during spell-correction. In other examples an unobtrusive warning could be shown to the user, another input could be requested for demonstrably incorrect data, such as when someone enters an email address that is missing the @.

Since it is a background criterion, it is difficult for the user to evaluate. However, there are some indicators that can also be observed in the GUI:

The application reacts to its own state, e.g. by deactivating ("greying out") menu items or buttons for a short time if they would have no application purpose at the moment.

Furthermore, any form of live checking for possible input errors is a good indication. Websites can check in advance whether you are likely to type correct things into a form ("This is not a zip code").

Robustness does not mean annoying and frustrating the user by requiring him to make exactly the right inputs and follow the correct sequence of steps, that the user has to start all over again or that the application terminates. It means that the application expects errors to occur at some point and corrects them, bypasses them or pauses to ask the user for confirmation.

An error-free application does not exist for the simple reason that errors also occur in interaction with the system as a whole. Other applications, the operating system itself, file formats can change and a program must already be programmed from scratch with possible errors in mind. Compatibility is therefore a sign of robustness.

2.4. File format

A particularly important aspect is the file format. If the application is not an end in itself, like a computer game, the user is mainly concerned with the resulting data: the finished picture (.png), the finished document (.pdf), the finished piece of music (.wav).

The question of whether an application will still be running in 20 years is more a question of whether the data or files will still be readable.

Simple and widely used formats with open specifications are preferable to other formats.

These file formats are independent of a specific application. Even if the application with which the document was originally created is lost in the future, it is highly likely that the data will still be readable and writable in the future. If the file format is well documented, with a little effort, a new

application can be written at any time that can process this data again.

A "loss" does not necessarily mean that the application has literally been lost. When, in the course of years, the entire world makes an architectural switch, many applications are not adapted to the new architecture. This happened in the late 1980s when everyone switched from a range of different home computer architectures to IBM-compatible personal computers, the kind that is still in use today.

Formats built on pure text (.txt, .svg, .html, .json, .xml ...) are predestined to survive application, system and architecture changes into the distant future. This does not mean that humans must be able to read the text (.svg is an image format), but that all data used in the file correspond to a human-readable letter or other character.

Even without an open specification, text-based formats are readable and interpretable by humans. This includes all application source texts and configuration files.

But it is not just about future-proofing. A good file format also makes it possible for several people and/or applications to work together and to share tasks: one graphics program is good with geometric shapes, the next is particularly good at applying filters and effects, another is designed to draw with tablets and virtual brushes as if on a canvas.

2.5. User Interface

A good user interface enables fast, robust and safe work, or entertainment. It enables the optimal learning curve: **Results as quickly as possible, guidance towards complete control.**

It usually takes the form of a graphical user interface (GUI) or a text-based command line application. A related term is UX - user experience.

One recognises good user experience by the optimal balance between freedom and guidance. E.g. if certain options are not useful at the moment, they should be hidden so that the user can concentrate on what is actually useful. But nothing gets hidden which **could** be used, even if not deemed sensible by the developers themselves.

Frequently used functions should be easier and faster to use than rarely used ones. If a certain series of sequential steps must be followed, then it should be presented in this form (e.g. a wizard).

Initially, no form of input is inferior or superior to another. Rather, good user navigation is tailored to the hardware of the respective system. Desktops/laptops with keyboard and mouse offer the most possibilities and the highest working speed. Some applications should rather be operated with the mouse, others unfold their potential best with pure keyboard use, sometimes even pure text input is superior to any graphical working environment.

On the other hand, a touchscreen is clearly the better choice for applications on tablets and phones. The applications may not be as powerful in their functional range (compared to a desktop computer version), but at least they should can be comfortable to use.

Finally, it should be said that certain applications have such complex tasks to perform that the often cited "intuitive user experience" cannot exist. Digital audio workstations or flight simulators are just too complex. Some authors even go so far as to say that there is no such thing as an intuitive

GUI. Everything was learned. This cannot be completely dismissed... In any case, at some point a level of complexity is reached that you just have to learn an application.

An application that patronises users is also a minus point. This is sometimes described as "dumbing down". For example, an operating system or file manager that hides file extensions (such as .jpg) insults the intelligence of the user. Or e.g. in the iPad video player it is not possible to set the audio/video latency manually. Instead, it is made impossible to combine a digital HDMI video signal with the analogue headphone output. The justification for this is that users are too stupid to compensate for the (technically inevitable) delay between picture and sound. In truth a simple setting that is available in players like VLC for decades.

2.5.1. Documentation

The application has educational and supporting material such as manuals, instructions in written and video form. For complex applications, this can also be a printed manual.

The documentation is well written and adapted to the target group. It also follows the optimal learning curve: results as quickly as possible, leading to complete control.

The developers and publishers of the application support and encourage community building among users and the production of support material (books, videos) by third parties. There may also be assistance in the application itself. Any form of suggestion or example increases the clarity of what the application wants you to do.

2.6. Flexibility

The application should be useful for a wide range of purposes. This does not mean that the application itself should be able to do everything and thus become overloaded and unwieldy. Rather, it means that its usefulness can increase with the user's requirements. Coupled with a good user interface or learning curve, a good application is thus able to achieve results very quickly (suitable for beginners or small tasks), on the other hand it can be used for very complex and large tasks.

Flexibility also means that the application can be used outside of what the developers originally intended, e.g. an application that was originally intended for transcribing laboratory notes, recorded as audio , can also be used to write subtitles for a movie.

Ideally the application can be used in a chain of applications, or can be extended relatively easily (without requiring the developers).

In the so-called modular approach, each application fulfils a clearly defined purpose and thus a role in an arbitrarily long chain of applications that automatically pass data to the next program. The most well-known representatives are the so-called "UNIX Tools" or "GNU Tools".

An application that is "made of one piece" is called monolithic. These are usually complex GUI applications that want to offer as many functions as possible. If such a software wants to be flexible, it must be based on open file formats and provide plug-in interfaces and scripting possibilities. Some applications allow some kind of external control, which is called an open API.

Finally, it should be said that applications that can and want to do everything have consistently failed in the past: Music producing video editors, spreadsheets integrated in writing programms etc. These kinds of applications rightly have a reputation for being junk and are aimed at a group of buyers who don't yet know any better. It's not that the all-in-one principle couldn't work in theory, but in practice software is made and used by people, and they only have a limited amount of time, resources and concentration. An application that is supposed to do twice as much is usually not only twice as difficult to use and develop, but many times more.

2.6.1. Software licence as an artificial restriction

A word about software licences. They can restrict the allowed usecases and artificially limit even technically available possibilities. Such licences can require a product to be used exclusively for "personal, non-commercial use", and one is startled by what this really means (at least under German law): no job applications with Microsoft Office for students, no publication of the self-edited video on Youtube, etc. "Private use" has nothing to do with money in the vast majority of cases, as often believed. Software for "personal use" is so restricted that it is almost impossible to use in reality. You want to create a note for your neighbour to search for his missing cat? That is a (gratuitous) service and thus not allowed in our current example.

Common negative examples are test-, beginner- and pupil/student versions. These offer a cheap price (or paid by the school) for a seemingly powerful, expensive application. This practice is found exclusively in proprietary and commercial software that wants to circulate its products as early as possible (school/training/university) through lobbying, in order to get the respective users used to the software. When the period of use expires, e.g. after graduation, the person is left without access to the application and maybe even without access to their data.

In the worst case, this data is stored exclusively on a company server, or euphemistically: in the "cloud". The seemingly only solution for the user is to buy a product that may not even be the best, in terms of our other criteria. The marketing department and excuses of other users who have already fallen into this trap ("well, it is the industry standard") do their part to form a self-perpetuating system. You have been warned.

2.7. Scalability

The criteria in this article are not entirely separable; scalability has components of user experience, as well as flexibility. Nevertheless, it is useful to consider this point separately. Here we are talking about applications that produce something: Text, graphics, sound etc. and not media players or games.

An application is usually particularly well suited to a certain size of task. Scalability describes how far down (less/simpler) and up (more/complex) the application can deviate while remaining easy to use and efficient. We assume the optimal case, that there are no technical problems and the application is well programmed.

Ideally, the description (website, promotional material, etc.) should tell you what the application is intended for before you use it. In the example of word processing: Is it a notebook or for scientific papers? Do you use it to write short stories or the Lord of the Rings?

Naively, one might think that it is best to take an application that can do as much as possible. In

reality, the scalability of software is not only limited upwards, but also downwards!

The upwards limitation means that tasks eventually become too big. You spend too much time managing your projects, making changes to the overall document or adding new content becomes time consuming. Imagine using a graphics programme without layers, a MIDI sequencer where you only record live and then can’t change individual notes afterwards …

Downwards, applications are limited by the fact that even setting up the project takes too much time. The "overhead" is too great. If it is assumed that one will produce a series of novels in several volumes or a complete edition of all Beethoven sonatas, then it is no obstacle to spend a few hours setting up the layout, etc. This is followed by months of work in which one no longer has to worry about the layout. If you only want to write a shopping list so that it fits exactly on a DIN A4 page, the time for setting up the basic structure of book first would of course be too much.

2.8. Availability

The application runs on as many computers, devices and operating systems as possible, now and in the future, and as a bonus on older machines as well. A highly available application runs on your own computer, not as a service on the internet, even if it may ultimately be controlled via a browser.

At the same time, installation and set-up should be as straightforward as possible. Good existing approaches are package managers, as on Linux systems, or installation programmes that include all files, as is common in Windows. The former are faster, more convenient and easier for software updates, the latter have high compatibility.

Copy protection systems and DRM ("Digital Rights Management") are major negative points for this category. Since computers have existed, these have failed and annoyed honest users, or completely prevented an application from being used. CD/DVD/Blu-Ray DRM in games has always been a source of misery, as even when the game was just released, many drives simply could not read the media because the "copy-protection" falsely prevented it.

And how do you unlock software that requires a "USB-Dongle" if the system fails because it was intended for the quirks of USBv2 but everybody moved on to USBv3? How do you authenticate online through a server if the server was switched off years ago?

Any form of forced online access (authentication or so-called "software as a service") is a ticking time bomb. The developers place no value whatsoever on good runnability of the application, not to mention customer friendliness. Long-term operability is already written off by design. The servers are switched off when the company’s accounting department says so, and not when the users/customers are no longer interested. This point is currently (2021) the worst thing that can happen to software. Anyone who says otherwise wants your money. If you are already trapped in such a system, you should take care to save all data produced with it on your own system as soon as possible and get rid of the application as soon as possible, no matter how effective it may be.

2.9. Price

An application has the right to cost money, the developers should not make a loss. However, if the

other criteria allow it, "free" should be the choice. In this case, it is usually open source software where the development costs and salaries have been covered by something other than pure sales. If, however, a low or free price is tied to restriction on use (see above), you should refrain from using it and either look for another application or pay fully for it.

Applications that run on one's own computer do not cause any further costs for the developer. Software is not a service, but goods. If you are asked to pay money on a regular basis for something that does not clearly and fundamentally requires an online service (such as an MMORPG computer game), it is probably a rip-off, a deliberate violation of data protection, or the latest trend towards vendor lock-in.

2.10. Privacy and Data Protection

Ideally, the application should not connect to any networks or the Internet. If internet access is the purpose of the application, it must connect to exactly the requested remote server and not to any third party.

Telemetry and statistics on user behaviour are of dubious value and a minus point, but should at least be announced in the application itself and may only be sent after explicit permission by the user. The default setting must be to send nothing.

The software reads only its own data on the hard disk (and this means the actual software data, not the user's documents) and everything else only after explicit permission, e.g. through the menu item "Open file", or implicitly through the purpose of the software itself, e.g. a file manager or a program to search for files.

Finally, writing or saving data is also only allowed after instruction by the user. One might think that this is self-evident, but there are applications that permanently create temporary data, such as the camera of a smartphone so that a preview image can be displayed. This data must not be stored. By now it should be clear that sending this camera data without being asked is out of the question. No amount of plus points in the other categories can outweigh such behaviour.

3. For which criteria does "open source" lead to improvement?

Software licences are often used to group applications. There are two main groups: open source licences and "everything else" ("proprietary", sometimes "closed source").

There are many open source licences (e.g. GPL, MIT, BSD...) with sometimes significant legal differences. For us users, the situation is simpler and the licences can be grouped together: Installation, redistribution and intended use are not restricted. Everything is explicitly allowed by the licence. For software developers, it also matters that the source code of the applications (i.e. what you need to change the application) is public and everyone has permission to change the source code. For users, this has only indirect advantages.

To clarify, as many have probably glanced over the sentence above: 'Installation, redistribution and use are not restricted'. Any question that starts with "And what about ... ?" or "May I use this version

… ?" can be answered with "yes, you are allowed to do that".

The author is of the opinion that an open source licence only gains its value by having a a positive effect on the criteria discussed here. The consequences that result from the licence are the real value.

What are the consequences?

Here, as briefly as possible, is a list of whether an open source licence (OS for short) has a positive effect. That is, whether one can expect that an OS application tends to perform better than average in one of our categories.

Effectiveness: The licence has no effect on the functionality or bug-free nature of the application. However, open source code is the basic requirement for security (no vulnerability to "hackers"). Only in this way can the code be checked by independent people and groups. OSS almost never has intentional privacy breaches.

Efficiency, robustness, usability, scalability: Unaffected by licence. OSS has no consequence one way or the other. These criteria improve through systematic testing and optimisation; all not a consequence of the licence, but a matter of time and money as well as skills, professional pride and passion of the software developers.

File format: OSS is an advantage. It is true that a proprietary application can also use open, publicly specified file formats, but this is the rule for OSS applications. Even in the worst case scenario OSS has an advantage: if the application fails to start in the future and the file format is binary (i.e. not text-based, and thus not human-readable), people could reconstruct the file format by analysing the source code and write a new application to edit the files.

Flexibility it was discussed that restrictions on use can be created by a proprietary licence. This is never a problem with OSS: you have legal certainty that everything is allowed within the framework of the applicable laws. Whether an application is flexible enough to be used beyond the developer’s intended purpose is, however, independent of OSS, as is cooperation with other programmes. The only thing to mention here is the traditionally strong willingness of OSS programmers to build on existing software, which could have a positive effect. But it might not. For a user without programming skills, an application only has the possibilities it ships with, regardless of whether it is OSS or not. However, if one has programming skills (or money to pay a programmer) then the changeability of the application itself is a huge plus for OSS.

Availability: One of the strongest and most obvious advantages of OSS. Applications are offered for even the most exotic, or already depreciated, operating systems or hardware configurations. Even in the future, the applications will still be portable and can be reactivated on systems that cannot be foreseen today. This is not the decision of a company’s accounting department. Setting up an application for new, old or special hardware or operating systems is a comparatively small effort and the work of one person has an instant positive effect on hundreds or thousands of other users. DRM, planned obsolescence and expiry dates are also not found in OSS.

Price: The majority of OSS is also free of charge and thus has an advantage.

Privacy: As already mentioned, data protection can only be implemented with OSS. OSS has such a clear advantage that any further comment is unnecessary.

3.1. Conclusion: Open Source Licences

Open source licences are never directly a disadvantage, but sometimes an advantage.

For most people, the list of features or "effectiveness" is the sole criteria for good software. Hopefully, this article has made it clear that this alone is not a good basis for decision-making. Therefore, if you have to choose between two applications that can do similar things, you should always go for open source software for the reasons mentioned above. The author is of the opinion that even with slight functional disadvantages, the practical aspects more than outweigh them and, for example, a slightly more complicated user interface is more than outweighed by data protection, security, high availability, uncomplicated legal situation ("What am I allowed to do? Everything!"), future security/archivability and flexibility.