Kriterien für die Auswahl von Software

Inhaltsverzeichnis

1. Motivation

Dieser Artikel soll dabei helfen sich über die Kriterien von guter Software klar zu werden. Dazu wurden diese in Kategorien eingeteilt, die im folgenden einzeln besprochen werden. Es wird aufgezeigt wie ein Programm in diesen Kategorien gut oder schlecht abschneidet. Am Ende erlangt der Leser hoffentlich mehr Klarheit, was für sie oder ihn ein gutes Programm ist und worauf man achten kann, wenn man sich ein neues sucht.

Bei diesem Artikel handelt es sich um die Verschriftlichung meiner persönlichen Erfahrung. Es wurde keine systematischen Tests oder Befragungen gemacht.

Ich würde mich freuen, wenn Benutzer sich **aktiv** für ein Programm entscheiden würden. Hat man sich für ein Office-Programm entschieden, oder wurde es einem in der Schule in die Hand gedrückt und seitdem benutzt man das? Ist das verwendete Programm davon abhängig, was auf dem Computer drauf war, als man ihn kaufte?

Voraussetzung ist die Reflexion der eigenen Wünsche und Ansprüche. **Zuerst** muss man wissen, was man möchte: Was ist das Problem, was ist der Zweck? Erst **danach** sucht man das Programm aus. Sonst passiert es schnell, dass man den Versprechungen der Werbung nachrennt, danach entscheidet was scheinbar „alle“ benutzen oder auf das Marketing hereinfällt. Dann gerät man an Programme, die in zu vielen Kriterien zu schwach sind oder verfängt sich im Lock-in-Effekt.

Das perfekte Computerprogramm existiert nicht. Man muss immer Kompromisse eingehen: Vielleicht ist einem 3D-Grafikprogramm die resultierende Bildqualität hervorragend, dafür braucht es ein Vielfaches an Zeit, verglichen mit einem Programm, das nicht so schöne Ergebnisse produziert. Vielleicht lässt sich in einem Textverarbeitungsprogramm besonders ablenkungsfrei und komfortabel schreiben, aber es kann nicht ins PDF Format exportieren. Oder es macht genau was man sich wünscht, stürzt aber regelmäßig ab und man muss sich angewöhnen alle 30 Sekunden auf Strg+S zu drücken. Evtl. ist das Programm überhaupt nicht für meine Hardware/System verfügbar.

Für jeden ist irgendwann das Ende der Kompromissbereitschaft erreicht. Ein Programm, das zuverlässig und schnell alle Aufgaben erfüllt, aber dafür jeden Schritt ausspioniert und den Inhalt der eigenen Festplatte über das Internet versendet sollte indiskutabel sein und von niemandem benutzt werden.

Die Begriffe „Programm“ und „Software“ werden synonym verwendet. Gemeint sind ausführbare Computerprogramme, die eine Aufgabe erfüllen oder ein Problem lösen. Das schließt Produktivsoftware, wie auch Unterhaltung mit ein. Das Programm kann dabei auf beliebigen Systemen laufen: Allzweck-Computer wie Desktops und Laptops, Telefone, Tablets auf der Firmware eines Kühlschranks …

2. Kriterien

Nach einer Reflexion über meine eigene Softwareauswahl habe ich für mich folgende Kriterien formuliert:

- Effektivität: Macht was ich möchte, was es versprach und stürzt nicht ab. Keine Sicherheitslücken.
- Effizienz: Nutzt Systemressourcen sparsam und arbeitet schnell.
- Robustheit: Kommt mit vielen Systemkonfigurationen und (Fehl)eingaben klar.
- Dateiformat: Offenes, öffentlich spezifiziertes Dateiformat, am besten textbasiertes Standardformat: Interoperabilität und Archivierbarkeit für Jahrzehnte.
- Benutzerführung: Schnell und fehlerfrei zu bedienen, keine unnötigen Arbeitsschritte vom Menschen verlangen, gute Lernkurve selbst für komplexe Anforderungen. Keine „Verdummung“. Am Anfang Ergebnisse, am Ende Kontrolle.
- Flexibilität: Kann für vieles eingesetzt werden, auch über den vom Entwickler vorgesehen Zweck hinaus. Kann gut mit anderen Programmen zusammenarbeiten. Die Lizenz schränkt die Benutzung nicht künstlich ein.
- Skalierbarkeit: Eignet sich für verschiedene Größen von Projekten, oder ist klar für eine definierte Größe ausgelegt.

- Verfügbarkeit: Läuft auf vielen Computern und Betriebssystemen. Jetzt und in Zukunft.
- Preis: So günstig wie möglich, ohne dass die Entwickler Verlust machen.
- Datenschutz: Gibt nichts ungefragt an Dritte weiter, liest nur seine eigenen Daten oder andere nach Erlaubnis.

Bewusst oder unbewusst können diese Kriterien einzeln betrachtet und bewertet werden, um abschließend zu entscheiden, ob man ein Programm erlernen oder benutzen möchte.

2.1. Effektivität oder Funktionsumfang

Das Programm macht, wozu man es benutzen möchte und was es soll. Dabei ist es fehlerfrei: Es hat keine logischen Fehler, zum Beispiel verrechnet es sich nicht, spielt nicht die Note Fis, wenn ich Des eingegeben habe oder streicht mir ein Wort bei der Rechtschreibprüfung an, das richtig geschrieben ist. Es hat aber auch keine Programmfehler: Es stürzt nicht ab, speichert alle Daten vollständig, und wieder lesbar usw. Sicherheitslücken sind auch als Fehler im Programm anzusehen.

Bewertet wird die Effektivität im Verhältnis dazu, was das Programm verspricht: Entweder explizit in einem Text, etwa auf einer Webseite, oder implizit durch die Gestaltung der grafischen Oberfläche (GUI - Graphical User Interface), die etablierten Konventionen folgt: Sehe ich etwa ein weißes Rechteck auf dem Bildschirm, mit einem blinkenden kleinen Strich links oben gehe ich davon aus, dass ich Text eintippen kann.

Ein verwandtes Konzept ist der „Funktionsumfang“. Im Allgemeinen wünscht man sich, dass die ausgewählte Software so viel wie möglich kann, wenn die eigenen Ansprüche und Fähigkeiten wachsen sollte es weiterhin verwendet werden können. Hier zeigt sich ein grundlegendes Problem bei der Softwareauswahl das erste Mal in diesem Artikel: Software, die zu viel anbietet, wird zu komplex. Die Benutzerführung leidet, von einer zu starken Lernkurve bis zur Unbenutzbarkeit; und hinter den Kulissen wird das Programm schlecht überprüfbar und wartbar, was Fehlerquellen für Abstürze und Datenverlust auf den Plan ruft.

Ob, und ab wann die Balance der Kriterien unverhältnismäßig wird und negative Aspekte nicht mehr tolerierbar sind, bzw. positive Aspekte so wichtig sind, dass sie alle Anderen nachrangig machen, soll der Rest dieses Artikels erläutern.

2.2. Effizienz

Das Programm arbeitet schnell und verschwendet keine Computer-Ressourcen. Damit verbraucht es auch weniger Energie (Strom/Watt). Ein Programm mit den falschen Algorithmen kann deutlich mehr Zeit für dieselbe Aufgabe verbringen, und ist damit nicht effizient.

Auch Speicherverschwendung ist ein Problem. Einen Texteditor, der 1 Gigabyte RAM braucht um 20 Kilobyte Text anzuzeigen ist verschwenderisch. 8 MB Datenvolumen für eine Webseite mit einem Kuchenrezept sind eine Frechheit.

Gute Programmierarbeit führt zu effizienter Software. Diese entsteht oft durch Geduld und Leidenschaft der Programmierer, sich immer weiter mit einem schon funktionierenden Programm (im Sinne der Effektivität) zu beschäftigen. Wenn schon nach der ersten, naiven Version aufgehört

wird zu verbessern ist evtl. noch nicht der optimale Weg gefunden, das Programm seine Arbeit machen zu lassen. Auch Wissen und Erfahrung der Programmierer helfen hier, z.B. bei der Auswahl von Algorithmen, Programmbibliotheken wie GUI-Toolkits oder Spieleengines.

Letztendlich ist ineffiziente Software „egoistisch“. Selbst wenn ein Arbeitsschritt immer noch in für Menschen akzeptabler Zeit gelöst wird, werden dadurch andere Programme und Prozesse im System blockiert.

Eine andere Art von Effizienz ist gute Benutzerführung, die im Kapitel Benutzerführung unten besprochen wird.

2.3. Robustheit

Das Programm ist geschützt gegen Unfälle und Fehleingaben der Benutzer.

Robustheit entsteht im nicht-sichtbaren Teil des Programms. Hier muss streng darüber gewacht werden, dass exakt das richtige aufgerufen und gespeichert wird. Wird etwa die Funktion „Rechtsschreibkorrektur für ein Wort“ aufgerufen, aber es wurde eine Zahl 2301 eingegeben wird das Nachschlagen und der Vergleich mit dem Wörterbuch fehlschlagen. Ein robustes Programm wird in geordneter Weise einen Fehler melden (alles intern) und dafür sorgen, dass überhaupt nichts gemacht wird, anstatt etwas Falsches. In diesem Beispiel würde in einer GUI gar nichts passieren, die Zahl wird bei der Korrektur übersprungen. In anderen möglichen Beispielen könnte etwa eine unaufdringliche Warnung an den Benutzer zurückgegeben werden, bei nachweislich falschen Daten eine erneute Eingabe verlangt werden, etwa wenn jemand eine E-Mail-Adresse eingibt, in der das @ fehlt.

Da es ein Hintergrund-Kriterium ist, ist es schwierig für den Benutzer zu bewerten. Es gibt aber einige Hinweise, die man auch in der GUI beobachten kann:

Das Programm reagiert auf seinen eigenen Zustand, z.B. indem Menüpunkte oder Schaltflächen kurzfristig deaktiviert („ausgegraut“) werden, wenn sie momentan keinen Anwendungszweck hätten.

Des Weiteren ist jede Form von Live-Überprüfung auf mögliche Eingabefehler ein guter Hinweis. Webseiten können schon im Vorhinein überprüfen, ob man in ein Formular wahrscheinlich korrekte Sachen hineinschreibt („Das ist keine Postleitzahl“).

Robustheit bedeutet also nicht, dass man als Benutzer damit genervt und frustriert wird nur exakt richtige Eingaben und Arbeitsschrittreihenfolgen einzuhalten und ansonsten z.B. von vorne beginnen muss oder sich das Programm beendet. Es bedeutet, dass das Programm damit rechnet, dass irgendwann Fehler auftreten werden und diese korrigiert, umgeht oder innehält um beim Benutzer eine Bestätigung einzuholen.

Ein fehlerfreies Programm existiert schon aus dem Grund nicht, dass Fehler auch in Wechselwirkung mit dem Gesamtsystem entstehen. Andere Programme, das Betriebssystem selbst, Dateiformate können sich ändern und ein Programm muss bereits mit diesen Möglichkeiten von Grund auf programmiert werden. Kompatibilität ist also ein Zeichen für Robustheit.

2.4. Dateiformat

Ein besonders wichtiger Punkt ist das verwendete Dateiformat. Ist das Programm kein Selbstzweck (Computerspiel) geht es dem Benutzer hauptsächlich um die daraus resultierenden Daten: das fertige Bild (.png), das fertige Dokument (.pdf), das fertige Stück Musik (.wav).

Hinter der Frage, ob ein Programm in 20 Jahren noch laufen wird verbirgt sich eher die Frage, ob man die Daten bzw. Dateien noch lesen werden kann.

Einfache und weit verbreitete Formate mit offener Spezifikation sind prinzipiell anderen Formaten vorzuziehen.

Diese Dateiformate sind unabhängig von einem bestimmten Programm. Selbst bei zukünftigem Verlust des Programms, mit dem das Dokument ursprünglich erstellt wurde, wird es mit hoher Wahrscheinlichkeit so sein, dass die Daten in Zukunft noch les- und schreibbar sind. Ist das Dateiformat gut dokumentiert, lässt sich, mit etwas Mühe, jederzeit ein neues Programm schreiben, dass diese Daten wieder verarbeiten kann.

Ein „Verlust“ muss nicht zwingend dergestalt sein, dass man das Programm wortwörtlich verloren hat. Wenn, im Laufe von Jahren, die gesamte Welt einen Architektursprung macht werden viele Programme nicht an die neue Architektur angepasst. Dies geschah etwa Ende der 1980er Jahre als alle von sehr unterschiedlichen Heimcomputer-Architekturen zu IBM-Kompatiblen Personal Computern wechselten, die Art, die auch heute noch in Gebrauch ist.

Formate, die auf reinem Text aufbauen (.txt, .svg, .html, .json, .xml …) sind dafür prädestiniert Programm-, System- und Architekturwechsel bis in die ferne Zukunft zu überstehen. Damit ist nicht gemeint, dass man als Mensch den Text ließt (.svg ist ein Bildformat), sondern dass alle verwendeten Zeichen in der Datei einem menschen-lesbaren Buchstaben oder ein sonstigen Zeichen entsprechen.

Selbst ohne offene Spezifikation sind diese im Zweifelsfall durch Menschen lesbar und interpretierbar. Hierzu gehören auch alle Programm-Quelltexte und Konfigurationsdateien.

Es geht aber nicht nur um Zukunftstauglichkeit. Ein gutes Dateiformat ermöglicht auch die Zusammenarbeit von mehreren Personen und/oder Programmen. Ja, es ermöglicht erst Aufgabenteilung: Das eine Grafikprogramm kann gut mit geometrischen Formen umgehen, das nächste ist besonders gut im Anwenden von Filtern und Effekten, ein anderes wiederum ist dafür gedacht, mit Zeichen-Tablets und virtuellen Pinseln wie auf einer Leinwand zu zeichnen.

2.5. Benutzerführung

Gute Benutzerführung ermöglicht schnelles, robustes und sicheres Arbeiten, oder Unterhaltung. Sie ermöglicht die optimale Lernkurve: **So schnell wie möglich Ergebnisse, Hinführung zur vollständigen Kontrolle.**

Sie kommt meist in Form einer grafischen Oberfläche zur Geltung (GUI, Graphical User Interface), als textbasiertes Kommandozeilen-Programm, oder Terminalprogramm. Oft hört man auch das Stichwort UX - User Experience.

Man erkennt gute Benutzerführung daran, dass eine optimale Balance zwischen Freiheit und Anleitung existiert. Sind bestimmte Optionen im Moment nicht sinnvoll sollten diese ausgeblendet werden, damit sich der Benutzer auf die tatsächlich nützlich konzentrieren kann.

Häufig benutzte Funktionen sollten einfacher und schneller zu bedienen zu sein, als selten genutzte. Muss eine bestimmte Reihe von aufeinander aufbauenden Arbeitsschritten eingehalten werden, dann sollte das auch in dieser Form präsentiert werden (z.B. ein Wizard).

Hierbei ist zunächst keine Form der Eingabe einer anderen unter- oder überlegen. Eher ist eine gute Benutzerführung auf die Hardware des jeweiligen Systems zugeschnitten. Desktops/Laptops mit Tastatur und Maus bieten die meisten Möglichkeiten und die höchste Arbeitsgeschwindigkeit. Manche Programme sollten eher mit der Maus bedient werden, andere entfalten ihr Potenzial am besten bei reiner Tastaturbenutzung, teilweise sogar ist reine Texteingabe jeder grafischen Arbeitsumgebung überlegen.

Auf der anderen Seite ist ein Touchscreen für Anwendungen auf Tablets und Telefonen klar die bessere Wahl. Die Programme sind zwar nicht so mächtig in ihrem Funktionsumfang, aber immerhin sollten sie das, was sie können, komfortabel bedienbar machen.

Schließlich sei gesagt, dass bestimmte Programme so komplexe Aufgaben zu erledigen haben, dass die viel beschworene „intuitive Benutzerführung“ nicht existieren kann. z.B. Digital Audio Workstations oder Flugsimulatoren. Manche Autoren gehen sogar so weit, dass es so etwas wie eine intuitive GUI nicht gäbe. Alles sei gelernt. Das ist nicht ganz von der Hand zu weisen… Jedenfalls ist irgendwann die Komplexität erreicht, dass man ein Programm regelrecht erlernen muss.

Ein Programm, das Benutzer bevormundet ist ebenfalls ein Minuspunkt. Dies wird manchmal als „Verdummung“ oder „dumbed down“ beschrieben. z.B. ist ein Betriebssystem oder Dateimanager, der Dateiendungen (wie .jpg) versteckt beleidigt die Intelligenz des Benutzer. Oder z.B. ist es im iPad Videoplayer nicht möglich die Audio/Video Latenz von Hand einzustellen. Stattdessen wird es unmöglich gemacht, dass man ein digitales HDMI-Videosignal mit dem analogen Kopfhörerausgang kombiniert. Gerechtfertigt wird dies damit, dass die Benutzer zu dumm seien die (bei dieser Anwendung zwangsläufig entstehende) Verschiebung von Bild und Ton zu kompensieren. In der Realität ist dies eine simple Einstellung, die in Mediaplayern wie dem VLC seit Jahrzehnten zur Verfügung steht.

2.5.1. Dokumentation

Das Programm verfügt über Begleitmaterial wie Handbücher, Anleitungen in Schrift- und Videoform. Bei komplexen Programmen kann das auch ein gedrucktes Handbuch sein.

Die Dokumentation ist sprachlich einwandfrei und an die Zielgruppe angepasst. Auch sie folgt der optimalen Lernkurve: So schnell wie möglich Ergebnisse, Hinführung zur vollständigen Kontrolle.

Die Entwickler und Herausgeber des Programms unterstützen und fördern gegenseitige Hilfestellungen der Benutzer („Communitybildung“) und Herstellung von Hilfsmaterial (Bücher, Videos) durch Dritte.

Im Programm selbst kann es auch Hilfestellungen geben. Jede Form von Vorschlägen oder Beispielen erhöht die Klarheit, was das Programm von einem möchte.

2.6. Flexibilität

Das Programm soll für besonders viele Möglichkeiten und Einsatzzwecke nützlich sein. Damit ist nicht gemeint, dass das Programm selbst alles Mögliche können soll, und damit überfrachtet und unhandlich wird. Vielmehr ist gemeint, dass der Nutzen mit den Ansprüchen des Benutzers steigen kann. Gepaart mit einer guten Benutzerführung bzw. Lernkurve ist ein gutes Programm damit in der Lage sehr schnell Ergebnisse zu erzielen (für Anfänger oder kleine Aufgaben geeignet), andererseits kann es für sehr komplexe und große Ziele eingesetzt werden.

Flexibilität entsteht auch, wenn das Programm abseits davon eingesetzt werden kann, was die Entwickler sich ursprünglich ausgedacht haben. z.B. ein Programm, dass ursprünglich für die Transkription von als Ton aufgenommenen Labornotizen gedacht war nun eingesetzt werden kann um für ein Unterhaltungsvideo Untertitel zu schreiben.

Am besten ist es, wenn das Programm in einer Kette von Anderen eingesetzt werden kann oder relativ einfach erweitert werden kann (außer durch die Entwickler selbst).

Beim sogenannten modularen Ansatz erfüllt jedes Programm einen klar definierten Zweck und damit eine Rolle in einer beliebig langen Kette von Programmen, die sich die Daten untereinander automatisiert weiterschicken. Die berühmtesten Vertreter sind die sog. „UNIX Tools“ oder „GNU Tools“.

Ein Programm das „aus einem Stück“ ist, nennt man monolithisch. Es handelt sich meist um komplexe GUI Anwendungen, die möglichst viele Funktionen anbieten wollen. Möchte ein solches flexibel einsetzbar sein muss es auf offene Dateiformate aufbauen und Pluginschnittstellen und Skriptingmöglichkeiten bereitstellen. Manche Programme ermöglichen eine Art Steuerung von außen, man spricht von einer offenen API.

Schließlich sei gesagt, dass Programme, die alles können und machen wollen in der Vergangenheit stets gescheitert sind: Musik produzierende Videoeditoren, Tabellenkalkulation in Schreibprogrammen etc. Diese Art von Programmen haben zurecht den Ruf von Ramsch und zielen auf eine Käufergruppe ab, die es noch nicht besser weiß. Es ist nicht so, dass das All-In-One Prinzip theoretisch nicht funktionieren könnte, aber in der Praxis wird Software von Menschen hergestellt und benutzt, und diese verfügen nur über ein begrenztes Maß an Zeit, Ressourcen und Konzentration. Ein Programm, das doppelt so viel können soll ist meist nicht nur doppelt so schwierig zu bedienen und zu entwickeln, sondern um ein vielfaches mehr.

2.6.1. Softwarelizenz als künstliche Beschränkung

Ein Wort noch zur Softwarelizenz. Diese kann den Nutzungszweck einschränken und damit selbst technisch vorhandene Einsatzmöglichkeiten künstlich beschränken. Solche Lizenzen können verlangen ein Produkt ausschließlich für den „persönlichen, nicht-kommerziellen Gebrauch“ einzusetzen, und man ist entsetzt was dies im deutschen Recht wirklich bedeutet: keine Berufs-Bewerbungen mit Microsoft Office für Schüler, keine Veröffentlichung des selbst geschnittenen Videos auf Youtube etc. Privater Zweck hat in den allermeisten Fällen nichts mit Geld zu tun. Software für den „persönlichen Gebrauch“ ist so sehr beschnitten, dass sie fast nicht einsetzbar ist. Du möchtest deinem Nachbarn einen Zettel für die Suche nach seiner vermissten Katze erstellen? Das ist eine (unentgeltliche) Dienstleistung und damit in unserem momentanen Beispiel nicht erlaubt.

Typische Negativbeispiele sind Test-, Einsteiger und Schüler/Stundentenversionen. Diese bieten einen günstigen Preis (oder durch die Schule bezahlt) ein scheinbar mächtiges teures Programm. Diese Praxis ist ausschließlich in proprietärer und kommerzieller Software zu finden, die durch Lobbyarbeit ihre Produkte möglichst früh (Schule/Ausbildung/Universität) verbreiten möchte, um die jeweiligen Nutzer an die Software zu gewöhnen. Läuft die Nutzungsdauer aus, z.B. mit dem Schulabschluss, steht der Mensch nun da ohne Zugang zum Programm und ohne Zugang zu seinen Daten.

Im schlimmsten Fall liegen diese auch noch ausschließlich auf dem Firmenserver, oder euphemistisch: in der „Cloud“. Die scheinbar einzige Lösung ist der Kauf eines Produktes, das evtl. noch nicht einmal das beste ist (im Sinne der anderen Kriterien). Die Marketingabteilung und Ausflüchte anderer Nutzer, die bereits in diese Falle getappt sind („Industriestandard“) tun ihr übrigens, um ein sich selbsterhaltendes System zu bilden. Vorsicht.

2.7. Skalierbarkeit

Die Kriterien in diesem Artikel sind nicht ganz trennscharf; Skalierbarkeit hat Anteile der Benutzerführung, sowie der Flexibilität. Trotzdem ist es nützlich diesen Punkt auch separat zu betrachten. Hier geht es um Programme, die etwas herstellen: Texte, Grafiken, Ton etc. und nicht um Media Player oder Spiele.

Ein Programm eignet sich meist für eine bestimmte Aufgabengröße besonders gut. Skalierbarkeit beschreibt wie weit nach unten (weniger/simpler) und nach oben (mehr/komplexer) das Programm abweichen kann und dabei gut bedienbar und effizient bleibt. Wir gehen vom Optimalfall aus, dass es keine technischen Probleme gibt und das Programm gut programmiert wurde.

Am besten wird man bereits vor der Benutzung in der Beschreibung (Webseite, Werbematerial etc.) darauf hingewiesen wofür das Programm geeignet ist. Im Fall von Textverarbeitung: Ist es ein Notizbuch, oder eher für Essays? Schreibt man damit Kurzgeschichten oder den Herrn der Ringe?

Naiverweise könnte man denken, dass man am besten ein Programm nimmt, dass so viel wie möglich kann. In der Realität wird die Skalierbarkeit von Software nicht nur nach oben, sondern auch nach unten beschränkt!

Die Beschränkung nach oben äußert sich dadurch, dass die Aufgaben irgendwann zu groß werden. Man verbraucht zu viel Zeit damit seine Projekte zu verwalten, Änderungen am Gesamtdokument vorzunehmen oder neue Inhalte einzufügen. Man stelle sich vor ein Grafikprogramme ohne Ebenen zu verwenden, einen MIDI-Sequencer, bei dem man nur live aufnehmen, und dann einzelne Noten nicht mehr nachträglich abändern kann …

Nach unten sind Programme dadurch beschränkt, dass bereits das Einrichten des Projektes zu viel Zeit verbraucht. Der „Overhead“ ist zu groß. Wenn davon ausgegangen wird, dass man mehrbändige Romanreihen, oder eine Gesamtausgabe aller Beethoven-Sonaten produziert dann ist es kein Hindernis erstmal ein paar Stunden damit zu verwenden das Layout etc. einzustellen. Darauf folgt monatelange Arbeit, in der man sich nicht mehr um das Layout kümmern muss. Möchte man nur einen Einkaufszettel schreiben, sodass dieser genau auf eine DIN-A4-Seite passt wäre die Zeit für das Einrichten natürlich zu viel.

2.8. Verfügbarkeit

Das Programm läuft auf so vielen Computern, Geräten und Betriebssystemen wie möglich, jetzt und in Zukunft, und als Bonus auch auf älteren Modellen. Ein hochverfügbares Programm läuft auf dem eigenen Computer, und nicht als Service im Internet, selbst wenn es letztendlich über einen Browser gesteuert werden mag.

Dabei ist die Installation und Einrichtung so niedrigschwellig wie möglich. Gute existierende Lösungen sind Paketmanager, wie auf Linuxsystemen, oder Installationsprogramme, die alle Dateien mitliefern, wie in Windows üblich. Erstere sind schneller, komfortabler und einfacher für Programmupdates, letztere haben hohe Kompatibilität.

Kopierschutzsysteme, DRM („Digital Rights Management“) sind große Minuspunkte für dieses Kriterium. Seit es Computer gibt schlagen diese fehl und nerven die ehrlichen Benutzer, oder verhindern komplett, dass ein Programm benutzt werden kann. CD/DVD/Blu-Ray DRM in Spielen war immer schon eine Quelle des Leids, da selbst in Zeiten, in denen das Spiel gerade aktuell war, viele Laufwerke das Medium einfach nicht lesen konnte, bzw. der Kopierschutz das verhinderte.

Wie schaltet man ein Programm frei, das einen "USB-Dongle" benötigt, der auf speziellen Eigenheiten von USBv2 basiert, aber nun gibt es nur noch USBv3? Wie meldet man sich zur Freischaltung auf einem Onlineserver an, der schon vor Jahren abgeschaltet wurde?

Jede Form von Online-Zwang (Authentifizierung oder sogenannte „Software as a Service“) ist eine tickende Zeitbombe. Die Entwickler legen hier keinerlei Wert auf gute Lauffähigkeit des Programms, von Kundenfreundlichkeit ganz zu schweigen. Die Langzeit-Lauffähigkeit ist per Design schon abgeschrieben. Die Server werden abgeschaltet, wenn die Buchhaltung der Firma es sagt, und nicht wenn die Benutzer/Kunden kein Interesse mehr haben. Dieser Punkt ist momentan (2021) das Schlimmste was einer Software passieren kann. Jeder, der etwas anderes behauptet will dein Geld. Ist man bereits in solch einem System gefangen sollte man darauf achten so bald wie möglich alle damit hergestellten Daten auf seinem eigenen System zu speichern und daran zu arbeiten so schnell wie möglich von diesem Programm wegzukommen, egal wie effektiv es auch sein mag.

2.9. Preis

Ein Programm darf Geld kosten, die Entwickler sollen keinen Verlust machen. Falls die anderen Kriterien dies erlauben sollte „kostenlos“ die Wahl sein. In diesem Fall handelt es sich meist um Open Source Software, bei der die Entwicklungskosten (sprich: Lebenserhaltungskosten) durch etwas anderes als reine Verkäufe abgedeckt wurden. Sollte allerdings mit einem günstigen, oder kostenlosen, Preis eine Einschränkung der Nutzung (s.o.) verbunden sein sollte man darauf verzichten und entweder ein anderes Programm suchen oder bezahlen.

Programme, die auf dem eigenen Computer laufen, verursachen dem Hersteller keine weiteren Kosten mehr. Software ist keine Dienstleistung, sondern Ware. Wird verlangt, dass man regelmäßig Geld für etwas bezahlt, dass nicht klar und grundsätzlich einen Online-Service benötigt (wie etwa ein MMORPG Computerspiel) handelt es sich vermutlich um Abzocke, eine gezielte Verletzung des Datenschutzes, oder den neusten Trend zum Vendor-Lockin.

2.10. Datenschutz

Das Programm stellt am besten keine Verbindung zu Netzwerken oder dem Internet her. Wenn das in der Natur der Sache liegt und man etwa einen Austausch mit dem Internet wünscht soll das Programm sich genau mit der angeforderten Gegenstelle verbinden, und mit keinem Dritten.

Telemetrie und Statistiken zum Nutzerverhalten sind von zweifelhaftem Wert und ein Minuspunkt, gehören aber mindestens -im Programm selbst!- angekündigt und dürfen nur nach expliziter Erlaubnis durch den Benutzer versendet werden. Die Grundeinstellung muss sein nichts zu senden.

Das Programm liest nur seine eigenen Daten auf der Festplatte (und damit sind die eigentlichen Programmdaten gemeint, nicht die Dokumente des Benutzers) und alles andere erst nach expliziter Erlaubnis, etwa durch den Menüpunkt „Datei Öffnen“, oder implizit durch den Zweck des Programms selbst, z.B. ein Dateimanager oder ein Programm zur Suche nach Dateien.

Schließlich ist das Schreiben bzw. Speichern von Daten auch nur nach Anweisung des Benutzers erlaubt. Man mag jetzt denken, dass das selbstverständlich sei, aber es gibt Programme, die permanent temporäre Daten anlegen, etwa die Kamera eines Smartphones, damit ein Vorschaubild angezeigt werden kann. Diese Daten dürfen nicht gespeichert werden. Mittlerweile sollte klar sein, dass ein ungefragtes Versenden dieser Kameradaten indiskutabel ist. Kein noch so großer Pluspunkt in den anderen Kriterien kann ein solches Verhalten aufwiegen.

3. Exkurs: Bei welchen Kriterien führt „Open Source“ zur Verbesserung

Softwarelizenzen werden häufig benutzt um Programme zu gruppieren. Hier gibt es zwei Hauptgruppen: Open Source Lizenzen und „alles andere“ (proprietär, selten auch "closed source").

Es gibt viele Open Source Lizenzen (z.B. GPL, MIT, BSD...) mit teilweise deutlichen rechtlichen Unterschieden. Für uns Benutzer ist die Situation einfacher und die Lizenzen können zusammengefasst werden: Installation, Weitergabe und Verwendungszweck sind nicht eingeschränkt. Alles ist explizit durch die Lizenz erlaubt. Für Softwareentwickler spielt noch eine große Rolle, dass der Quellcode der Programme (also das, was man braucht um das Programm zu verändern) auch öffentlich ist und alle die Erlaubnis haben den Quellcode zu verändern. Für Benutzer hat dies nur indirekte Vorteile.

Um es noch einmal zu verdeutlichen, da viele wahrscheinlich über den obigen Satz hinweggelesen haben: `Installation, Weitergabe und Verwendungszweck sind nicht eingeschränkt`. Jede Rückfrage, die mit „Und was ist mit ... ?“ oder „Darf ich mit dieser Version ... ?“ beginnt, kann mit „ja, du darfst“ beantwortet werden.

Der Autor ist aber der Meinung, dass eine Open Source Lizenz erst dadurch ihren Wert erhält, indem sie sich positiv auf die hier besprochenen Kriterien auswirkt. Wertvoll sind also die Konsequenzen, die sich aus der Lizenz ergeben.

Welche Konsequenzen sind das?

Hier, so kurz wie möglich, eine Aufstellung, ob sich eine Open Source Lizenz (kurz: OS) positiv auf

ein Kriterium auswirkt. Das heißt, ob man erwarten kann, dass ein OS Programm tendenziell in einer Kategorie überdurchschnittlich gut abschneidet.

Effektivität: Die Lizenz hat keinen Einfluss auf den Funktionsumfang oder die Fehlerfreiheit des Programms. Allerdings ist ein offener Quellcode die Grundvoraussetzung für Sicherheit (keine Anfälligkeit für „Hacker“). Nur so kann der Code von unabhängigen Personen und Gruppen überprüft werden. OS hat so gut wie nie absichtliche Datenschutzbrüche.

Effizienz, Robustheit, Benutzerführung, Skalierbarkeit: Unbeeinflusst durch die Lizenz. OS hat keine Konsequenz in die eine, oder andere Richtung. Diese Kriterien verbessern sich durch systematisches Testen und Optimieren; alles keine Konsequenz der Lizenz, sondern eine Frage von Zeit und Geld sowie Fähigkeiten, Berufsehre und Leidenschaft der Softwareentwickler.

Dateiformat: OS ist ein Vorteil. Zwar kann ein proprietäres Programm auch offene, öffentlich spezifizierte Dateiformate nutzen, allerdings ist dies bei OS Programmen der Regelfall. Selbst im schlimmsten Fall ist OS im Vorteil: Wenn das Programm in Zukunft nicht mehr startet und das Dateiformat binär ist (also nicht textbasiert, und damit nicht menschenlesbar) könnten Menschen durch Analyse des Quellcodes das Dateiformat rekonstruieren und ein neues Programm schreiben, um die Dateien zu bearbeiten.

Flexibilität: Im Kapitel „Flexibilität“ wurde besprochen, dass Einschränkungen der Nutzung durch eine proprietäre Lizenz erwirkt werden kann. Dies ist bei OS niemals ein Problem: hier hat man (Rechts-) Sicherheit, dass alles im Rahmen der geltenden Gesetze erlaubt ist. Ob ein Programm so flexibel ist, dass es abseits des vom Entwickler vorgesehen Zweck hinaus eingesetzt werden kann ist allerdings unabhängig von OS, ebenso wie die Zusammenarbeit mit anderen Programmen. Hier ist lediglich die traditionelle Arbeitsweise von OS-Programmieren zu erwähnen auf bestehenden Programmbibliotheken aufzubauen, die sich positiv auswirken könnte. Könnte aber auch nicht. Für einen Benutzer ohne Programmierfähigkeiten hat ein Programm nur die Möglichkeiten, die es von Haus aus mitbringt, egal ob OS oder nicht. Hat man allerdings Programmierfähigkeiten (oder Geld einen Programmierer zu bezahlen) dann ist die Veränderbarkeit des Programms selbst ein enormer Pluspunkt für OS-Software.

Verfügbarkeit: Einer der stärksten und offensichtlichsten Vorteile von OS-Software. Diese werden selbst für die exotischsten, oder bereits abgeschriebenen, Betriebssysteme oder Hardwarekonfigurationen angeboten. Auch in Zukunft werden die Programme noch portierbar und reaktivierbar sein, und zwar auf heute noch nicht absehbaren Systemen. Hier entscheidet nicht die Buchhaltungs-Abteilung einer Firma. Das Einrichten eines Programms für neue, alte oder spezielle Hardware oder Betriebssysteme ist ein vergleichsweise geringer Aufwand und die Arbeit einer einzelnen Person wirkt sich augenblicklich positiv für hunderte oder tausende andere Benutzer aus. DRM, Sollbruchstellen und Verfallsdaten sind ebenfalls in OS nicht zu finden.

Preis: OS-Software ist mehrheitlich auch kostenlos und damit im Vorteil.

Datenschutz: Wie schon erwähnt ist Datenschutz nur mit OS umsetzbar. Hier ist OS so deutlich im Vorteil, dass jeder weitere Satz überflüssig ist.

3.1. Fazit: Open Source Lizenzen

Open Source Lizenzen wirken sich niemals direkt als Nachteil aus, aber manchmal als Vorteil.

Für die meisten Menschen ist der Funktionsumfang, bzw. „Effektivität“ das alleinige Kriterium für gute Software. Das dies allein keine gute Entscheidungsgrundlage ist, hat dieser Artikel hoffentlich verdeutlicht. Hat man daher die Auswahl zwischen zwei Programmen, die ähnliches zu leisten vermögen sollte man aus oben genannten Gründen stets auf Open Source Software zurückgreifen. Der Autor ist der Meinung, dass auch bei leichten funktionalen Nachteilen die praktischen Gesichtspunkte mehr als überwiegen und z.B. eine leicht umständlichere Benutzerführung durch Datenschutz, Sicherheit, Hochverfügbarkeit, Unkompliziertheit der Rechtslage („Was darf ich? Alles!“), Zukunftssicherheit/Archivierbarkeit und Flexibilität mehr als aufgewogen werden.