Wissenskompendium Nachhaltigkeit in der IT

Arbeitsgruppe Digitales im BNW

15.03.23

Inhaltsverzeichnis

Einführung

1 Grundannahmen

- Right to Repair?
- Dimensionen von Nachhaltigkeit
  - … bezogen auf SW, HW, IT, Betrieb, …
  - **Wesentlichkeit** -> welche Faktoren in welchem Bereich wie wichtig?
- Ökologisch / Sozial
- "lohnt sich finanziell"

2 Software

In diesem Kapitel soll sich alles um Anwendungen drehen.

2.1 Software-Auswahl

Die Auswahl der "richtigen" Software ist schwierig: Der Markt ist schier endlos groß, regelmäßig kommt neue Software auf den Markt; und hin und wieder verschwindet auch Software wieder.

Jede Software löst ein gewisses **Problem** für eine **Zielgruppe** (hoffentlich :)).

Wir beschreiben hier Kriterien, die bei Softwareauswahl in Hinblick auf Nachhaltigkeit helfen können. Ziel ist hier die Reduktion des sogenannten **Vendor Lock-In**, um zu ermöglichen, auch zu einem späteren Zeitpunkt die Software zu wechseln.

2.1.1 Der besondere Wert von Daten

Viele Diskussionen drehen sich um die Auswahl bestimmter Software, kontrastiert Programme gegeneinander, usw. Wir empfehlen einen anderen Blick: Den Fokus auf die von der Software verarbeiteten Daten, wie bspw:

- Dateien (Textdateien, Office-Dateien, Bild-Dateien)
- Datenbanken (Open Source oder Proprietär)
- E-Mails (empfangen oder senden)
- Kommunikation mit fremden Systemen über Schnittstellen

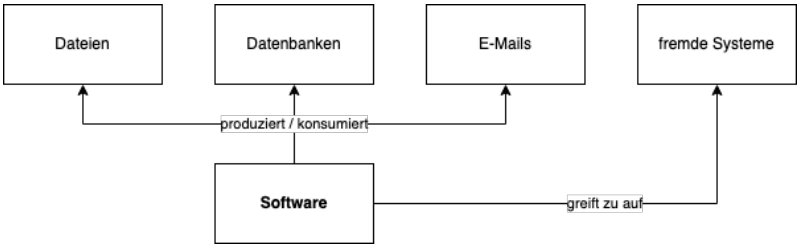

Abbildung 2.1: Zusammenhang zwischen Software und den Daten, die sie konsumiert/produziert

Als Erstes sollte für jede Software identifiziert werden, welche Daten diese verarbeitet oder ggf. erstellt. Für all diese Daten kann man sich folgende Fragen stellen:

2.1.2 Sind *Offene Standards* als Datenformate möglich?

Wenn es für das zu lösende Problem offene Standards gibt, welche das Problem umfänglich lösen, ist dies ein sehr guter Startpunkt. Dann kann man - ausgehend von dem offenen Standard - weiter nach Softwarelösungen, welche diesen Standard implementieren, suchen.

Offene Standards sind lizenzkostenfrei nutz- .und implementierbar, und "gehören" nicht einer einzelnen Firma, sondern in der Regel einem (mehr oder weniger formellen) Gremium aus mehreren Akteuren.

Proprietäre Standards werden in der Regel nur von einzelnen Firmen herausgegeben; häufig fallen für Nutzung auch Lizenzkosten an.

Problemstellung	Offener Standard	Proprietärer Standard
Office-Dokumente	ODF - OASIS Open Document Format for Office Applications	Word-Dateien (.doc, .docx) von Microsoft
E-Mail-Abruf, Kontakte, gemeinsame Kalender (PIM-Suite)	IMAP + Caldav + Carddav	Exchange / ActiveSync von Microsoft

Problemstellung	Offener Standard	Proprietärer Standard
gemeinsamer Chat	Matrix-Protokoll	bspw. Slack, Microsoft Teams

Wenn man einen offenen Standard identifiziert hat, kann man von diesem ausgehend nach Software suchen, die ihn implementiert. Hier sind die weiteren Kriterien relevant (Open Source etc).

Für viele Problemstellungen wird man noch keinen ubiquitären Standard finden. Daher sind weitere Kriterien relevant.

Mit **ubiquitär** meinen wir einen Standard, der sich duchgesetzt hat und breit, von vielen Marktteilnehmern, implementiert ist.

2.1.3 Besitzt die Software eine standardisierte Schnittstelle?

Falls es keinen etablierten Standard für das gewünschte Datenformat gibt, ist es hilfreich zu schauen, ob die Software eine Schnittstelle besitzt, über die alle Funktionen der Software abrufbar sind. Dies kann bspw. sein:

- eine API (maschinenlesbare Schnittstelle) im OpenAPI oder GraphQL Format; oder aber ausführliche und vollständige Dokumentation der API
- Verwendung einer **Open Source Datenbank zur Datenspeicherung**: In diesem Falle kann man jederzeit auf die **Rohdaten** der Software zugreifen - gewissermaßen "an der Software vorbei". Wir empfehlen hier uneingeschränkt nur reine Open Source Datenbanken wie SQLite, Postgres, MariaDB oder MySQL; da für kommerzielle Datenbanken häufig die Lizenzierung einen direkten Datenzugriff verbietet, oder aber eine sehr teure Lizenz nachgekauft werden muss.
- Verwendung von Open Source mit Erweiterungsmöglichkeiten. In diesem Fall kann in den Quellcode der Software direkt geschaut werden, und dieser entsprechend um Import/Exportfunktionen erweitert werden.

Ziel ist es, zu vermeiden, dass ein “Datensilo” entsteht, aus dem die Daten nicht mehr gewinnbringend und nachhaltig in der Organisation nutzbar sind.

2.1.4 Nutzung von Open Source Software

Open Source Software ist Software unter einer OSI-kompatiblen Lizenz wie bspw. MIT oder GPL-Lizenz. Es gibt darüber hinaus auch Lizenzen, welche nach der strikten Definition nicht OSI-kompatibel sind, aber im Alltag für die meisten Szenarien trotzdem ähnlich verwendet werden können, wie bspw. die Business Source Lizenz.

Die BSL ist entstanden, um zu verhindern, dass die großen Cloud-Anbieter wie Amazon oder Google Open Source Software kostenfrei einkaufen und als kostenpflichtigen Cloud-Dienst verkaufen können, ohne sich an der Weiterentwicklung der Software zu beteiligen.

Wir empfehlen uneingeschränkt die Nutzung von quelloffener Software, sofern diese existiert und die notwendigen Funktionen umfasst.

Bei Open Source Software, welche viele der Funktionen umfasst, aber einige Funktionen fehlen, empfehlen wir zu prüfen, ob ein Anbieter im Ökosystem der Software gefunden werden kann, der diese Funktionen im Open Source Kern der Software erweitert.

> **Hinweis**
>
> Open Source und Künstliche Intelligenz
> TODO: write here about weights
>
> - “Weights” der KI != Open Source (meistens)

2.1.5 SaaS

- Vendor Lock In
- Fokus auf Daten: SAAS produkt, welches nur quelloffene Daten generiert, kein großes Problem.
- Open Source Anbieter (=Hersteller von Open Source) haben auch häufig SAAS im Angebot
- Open Source und SAAS - bspw. Sentry -> kann funktionieren.

2.2 “Hard Facts” / Messbare Kriterien

2.2.1 Datentransfer

2.2.2 Energieverbrauch

- Energieverbrauch beim Trainieren von KI-Modellen
- Energieverbrauch beim Nutzen von KI-Modellen (chatgpt)

2.2.3 Datenvolumen/-speicher

3 Soziale Dimension

3.1 “do not” - Dark Patterns

3.2 Barrierefreiheit

3.3 Biases (bspw. bei KI/Algorithmen)

4 Hardware

4.1 Beschaffung

4.2 Betrieb

4.3 “do not” - eingebaute Obsoleszenz

5 Services (Betrieb von IT-Infrastruktur)

- Energy aware computing

6 Sourcing

Beschaffung, Lieferketten IT

Ausschreibung

Community - “Zurückgeben” an die Community bei Dienstleistern

- also Dienstleister bevorzugen, die in der entsprechenden Open Source Community aktiv sind.

7 Prozesse (Agile) - auf Veränderungen reagieren...

8 Links / Weiterführende Ressourcen / Nachweise

Quellen

Knuth, Donald E. 1984. „Literate Programming“. *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Mitwirken

Wir freuen uns über alle, die hier mitwirken, kommentieren, oder Teile schreiben wollen :)

hilfreiche Quarto Funktionen

(aus der Dokumentation rauskopiert)

> **Hinweis**
>
> Note that there are five types of callouts, including: `note`, `tip`, `warning`, `caution`, and `important`.

See Knuth (1984) for additional discussion of literate programming.

Here is an inline note.[1]

This is a span that has the class aside which places it in the margin without a footnote number.

Sidebar text

Footnotes

Here is a footnote reference,[2] and another.[3]

This paragraph won't be part of the note, because it isn't indented.

[1] Inlines notes are easier to write, since you don't have to pick an identifier and move down to type the note.

[2] Here is the footnote.

[3] Here's one with multiple blocks.

Subsequent paragraphs are indented to show that they belong to the previous footnote.

```
{ some.code }
```

The whole paragraph can be indented, or just the first line. In this way, multi-paragraph footnotes work like multi-paragraph list items.