OSH-Entwicklungsrichtlinie

- OSH-Entwicklungsrichtlinie
  - Vorbemerkungen
  - 1. Versionsverwaltung und Versionierung
  - 2. Ordnerstruktur und Benennung
  - 3. Dokumentation
  - 4. Lizenzierung
  - 5. Prinzipien
  - Anhang
    - OSH Directory Standard - Unixish Version

Vorbemerkungen

Komponenten frei und offen zu entwickeln unterscheidet sich in vielen zu bewältigenden Aspekten stark von der traditionellen firmeninternen Entwicklungsweise. Viele Abläufe wie sie in der modernen Softwareentwicklung mittels Versionierungssystemen kollaborativ über das Internet stattfinden haben nicht selten auch ein unternehmensinternes (manuelles) Pendant. So müssen Dokumente einen Freigabeprozess durchlaufen, sowie nachträgliche Anpassungen in die Informationsbasis eingepflegt werden. Ebenso spielt das Thema Lizenzierung eine zentrale Rolle. Allerdings aus der gegenüberliegenden Perspektive. Während für die Entwicklung von Open Source Komponenten das Urheberrecht maßgeblich ist um anderen Personen die Verwendung, Veränderung und Verteilung erlaubt spielt im klassischen Maschinenbau das Patentrecht eine entscheidende Rolle, welches diese Freiheiten einschränkt. Beide Welten verbindet jedoch das gemeinsame Ziel ein fehlerfreies Produkt mit möglichst geringem Aufwand zu erzeugen.

1. Versionsverwaltung und Versionierung

Versionsverwaltung

Verwende ein Versionierungssystem! Das Versionierungssystem überwacht wer, wann, was am Repository ändert. Das ist neben einer Reihe weiterer Vorteile die Grundlage für eine gültige

Lizenzierung. Häufig verlangen freie Lizenzen, wie die `CERN-OHL-2`, dass Veränderungen dargelegt werden müssen. Es stellt eine Reihe von Werkzeugen bereit um bspw. konkrete Entwicklungsstände anhand einer Versionsnummer eindeutig zu identifizieren, Änderungen zurückzuverfolgen, verschiedene Entwicklungen parallel ablaufen zu lassen und zusammenzuführen, mit anderen Personen strukturiert zusammenzuarbeiten, …

Empfehlung: **Git**

- Open Source Software
- kompatibel zu GitHub, GitLab, …
- wird unabhängig lokal verwendet, es ist kein Server oder Service-Anbieter notwendig
- Verfügbar für Windows, MacOS, Linux, Android
- Link: https://git-scm.com/
- Grafische Oberfläche:
  - Git ist ein Kommandozeilenwerkzeug. Es existieren aber zahlreiche Oberflächen für verschiedene Betriebssysteme.
  - Fortgeschrittene arbeiten meist direkt mit den Befehlen im Terminal. Alle Werkzeuge/Oberflächen sind miteinander kompatibel.
  - Wähle daher das Werkzeug mit welchem Du am besten zurecht kommst.
  - Empfehlung:
    - `VSCode` mit `GitGraph` oder `GitLens` Extension
      - der Editor kann für viele weitere Entwicklungsprozesse konfiguriert und erweitert werden, z. B. zur Erstellung Markdown-basierter Dokumentationen, zur Veränderungsverfolgung (`diff-tool`), siehe Dokumentation
    - Link: https://code.visualstudio.com/Download
    - Anmerkung:
      - `VSCodium` :
        - Telemetriefreie Version auf Basis von `VSCode`
          - Repository: https://github.com/VSCodium/vscodium/releases
          - Windows: `VSCodium-x64-1.88.0.24096.msi` [1]
          - Linux: nutze das Paket deines Paketmanagers!
        - Anmerkung: `VSCodium` hat standardmäßig keinen Zugriff auf den `Visual Studio Code Marketplace` , kann aber konfiguriert werden

1: https://github.com/VSCodium/vscodium/releases/download/1.88.0.24096/VSCodium-x64-1.88.0.24096.msi

Versionierung

Es muss zwischen “interner” und “externer” Versionsnummer unterschieden werden. Eine externe Versionsnummer entspricht mehr einer Produktbezeichnung und ist vor allem aus marketingtechnischer Sicht von Bedeutung. Interne und externe Versionierung können daher voneinander abweichen, (siehe Windows NT Versionen 3.1 bis 10). Die Versionsnummer ist des Weiteren relevant zur eindeutigen Zuordnung der Entwicklungsdokumente zu einem konkreten Artikel, welcher durch eine Artikelnummer identifiziert wird. Die Versionsnummer kann ein Teil der Artikelnummer sein. Für die Entwicklung ist die interne Versionsnummer maßgeblich. Grundsätzlich ist man frei darin ob man nach einem bestimmten Schema Entwicklungsfortschritte markiert und wie dieses Schema aufgebaut ist. Versionsnummern sind häufig mehrteilig. Jede Teilnummer kann hierbei eine spezielle Bedeutung erhalten. Immer häufiger orientiert man sich am Konzept der **Semantischen Versionierung** (`Semantic Versioning 2.0.0`)[2] als Konvention für aussagekräftige Versionsnummern Die Versionsnummer nimmt hierbei folgende Struktur ein: **`MAJOR.MINOR.PATCH`** .

Eine Erhöhung an den einzelnen Stellen bedeutet:

- `MAJOR` -Version, wenn **inkompatible** Schnittstellenänderungen vorgenommen werden
  - dies ist vor allem dann der Fall, wenn bestimmte Funktionen oder Anschlüsse in der neuen Version entfernt oder ihre Signaturen (Bezeichnung, Parameter) verändert werden und somit inkompatibel zu anderen Komponenten werden
- `MINOR` -Version, wenn neue Funktionen mit Abwärtskompatibilität hinzugefügt werden
- `PATCH` -Version, wenn abwärtskompatible Fehlerbehebungen oder andere kleinere Änderungen vorgenommen werden

2: https://semver.org/lang/de/

2. Ordnerstruktur und Benennung

Dateinamen und Ordnerstrukturen dienen vorrangig den Menschen um sich möglichst intuitiv zurecht zu finden. Für eine automatisierte Verarbeitung von Daten (z. B.: Parser, Crawler, Toolchains/Pipelines) ist jedoch die strikte Einhaltung von Spezifikationen notwendig. Änderungen der Strukturen und Namen ziehen häufig Anpassungen an anderen Stellen nach sich. Abkürzungen sind hilfreich solange ihre Bedeutung im entsprechenden Kontext möglichst eindeutig sind. Um späteren Aufwand zu minimieren ist es daher wichtig die Dinge so ausdrucksstark wie möglich zu benennen. (“Wozu ist das? Das ist blaues Licht. Und was macht es? Es leuchtet blau.”) Diese Empfehlung bezieht sich sowohl auf die Benennung von Dateien als auch auf die Benennung von Baugruppen und Unterbaugruppen sowie Einzelteilen in einem CAD-Modell innerhalb einer CAD-Datei (Featurebaum, siehe [Konstruktionsvorgaben für CAD-Systeme]).

Ordnerstruktur

Im Rahmen des Interfacer Projekts wurde für Open-Source-Hardware-Projekte ein Entwurf für eine Projektordnerstruktur entwickelt (“OSH project directory structure standards”).[3] Der Standard ist aktuell ein erster Entwurf und erhebt keinen Anspruch auf Vollständigkeit. Je nach Projektanforderungen kann es notwendig werden die vorgegebene Struktur sinnvoll zu ergänzen. Kostenkalkulationen können bspw. unter `/res/economic/` oder weitere spezifische Dokumente wie Wartungsanleitungen unter `/doc/maintenance/` gespeichert werden. Es wurden bereits eine Reihe Projekte nach diesem Standard strukturiert.[4] Einige Bezeichnungen und Abkürzungen erscheinen anfänglich ungewohnt, insb. in der Konstruktion, wie “mech” oder “manuf”. Bereits wenige ähnlich strukturierte Projekte zeigen ein erleichtertes Zurechtfinden. Auf Basis einer vereinheitlichten Struktur lassen sich ferner einfacher Automatisierungen (“Toolchains”) entwickeln. Nachfolgend ist die Hauptstruktur (Auszug) aufgeführt. Der vollständige Standard ist im Anhang zu finden.

```
.
├─┬ LICENSES/                    # Verzeichnis für Lizenztexte der verwendeten Lizenzen, wird
│ ├── CERN-OHL-S-2.txt           # Lizenztext der CERN-OHL-S-2
│ └── AGPL-3.0-or-later.txt
├─┬ doc/                         # Verzeichnis für die Dokumentation, Anleitungen
│ ├─┬ assembly/                  # Montageanleitung
│ │ └── main.{md|odt|svg}
│ ├── history/                   # Projektgeschichte
│ ├── manuf/                     # Herstellungsdokumente
│ ├── recycling/                 # Recyclingdokumente
│ ├── user/                      # Benutzeranleitungen
│ ├── maintenance/               # Wartungsanleitungen
│ ├── main.odt.license           # `.license`, ist eine einfache Textdatei um die gleichnamig
│ └── main.md                    # Allgemeine Dokumentation, Index, Überblick
├─┬ gen/                         # Verzeichnis für automatisch generierte Inhalte, welche nic
│ ├── doc/
│ ├── mech/
│ ├── software/                  # Kompilierte Software aus den Quellen `src/software`
│ ├── bom.{csv|ods}              # automatisch generierte Stückliste
│ ├── okh.toml                   # automatisch generierte Meta-Daten Datei, OKH-Manifest/Spez
│ └── ...
├── mod/                         # Git-Submodule, hier werden Abhängigkeiten zu anderen Kompo
├─┬ res/                         # projektbegleitende Dokumente
│ ├── concept/                   # Konzeptbeschreibung, lose Ideensammlung, Roadmap
│ ├── economic/                  # Kostenkalkulationen, Vergleichsrechnungen
│ └── var/                       # `various`-Verzeichnis für alle Dokumente welche keiner and
├── run/                         # Projektverwaltungs-Skripte, Installationsskripte, etc.
├─┬ src/                         # Quelldateien der Komponente
│ ├── anim/                      # Quelldateien für Animationen, als Basis für die Dokumentat
│ ├── calc/                      # Berechnungen
│ ├── elec/                      # Quelldateien für Elektronik, Schaltungen, PCB-Designs
│ ├── firmware/                  # Firmware-Quellen
│ ├─┬ mech/                      # Quelldateien für mechanische Designs, 3D-CAD-Modelle, Tech
│ │ └── case.fcstd               # CAD-Datei, bspw. für das Gehäuse
│ ├── software/                  # Alternative zu firmware oder andere Software-Quellen
│ └── test/                      # Quellcode für Softwaretests oder Designs für Testvorrichtu
├── bom.{csv|ods}                # Stückliste (Bill-of-Materials), manuell verwaltet
├── LICENSE.txt                  # Repository-Lizenz, die hier angegebene Lizenz gilt für all
├── okh.toml                     # Meta-Daten Datei, OKH-Manifest/Spezifikation, manuell verw
└── README.md                    # Grundlegende Beschreibung des Projekts sowie wichtige Info
```

3: https://gitlab.fabcity.hamburg/software/template-osh-repo-structure-minimal/ 4: https://www.interfacerproject.eu/news/buildworkshops/

- **Namenskonvention**:
  - `<Funktion>[_<Parameter>][_<Norm>|<Artikel-/Produktbezeichnung>].ext`
    - **Funktion**:
      - Eine Datei sollte so kurz aber so bezeichnend wie möglich anhand ihrer Funktion benannt werden (Funktionsaspekt [3])
    - in `[]` angegebene Bestandteile sind optional ergänzend
      - **Parameter**: Ist eine Unterscheidung nur anhand der Funktion nicht möglich müssen weitere Indikatoren hinzugezogen werden, z.B. der Ort (Ortsaspekt [3]) oder eine bestimmte Ausprägung (Geometrie, Leistung, Anschlüsse, …)
      - **Norm**: handelt es sich um ein normiertes Teil ist die Norm zu nennen.
      - **Artikel-/Produktbezeichnung**: Entspricht das Modell ausschließlich einem konkreten Produkt eines Herstellers ist die eindeutige Bezeichnung alternativ zur Norm zu nennen.
  - Wörter innerhalb eines Aspekts werden mit `-` voneinander getrennt, Aspekte mit einem `_`
  - Schreibe alle Wörter klein
  - Verwende englische Bezeichnungen

3: DIN EN 81346

3. Dokumentation

Erstelle eine **Readme**!

Diese stellt die erste grundlegende Projektdokumentation dar, welche eine Person sieht. Von ihr ausgehend Empfehlung: `Readme.md` in der obersten Dateiebene, (siehe Ordnerstruktur) Quasi-Standard in Git-Repositorys auf GitHub und co.

Grundlegende Struktur:

- Titel
- Beschreibung
- Aktueller Status (aktiv, inaktiv, …)
  - Projekte, welche nicht mehr aktiv betreut werden können, sollten in der Beschreibung darauf hinweisen

- Abhängigkeiten (inkl. konkrete Versionen und Lizenzen)
  - Software
  - Zukaufteile
- Ordnerstruktur
- Bauanleitung
- Arbeitsweise
  - Informationen für Beitragende
    - Developer-Guidelines, Namenskonventionen (wie diese Richtlinie), Branchingstruktur, etc.
  - Contributors License Agreement
    - Regeln wie mit Beiträgen Dritter rechtlich umgegangen werden soll
- Lizenzierung

Verwende `Markdown` !

- Link: https://www.markdownguide.org/basic-syntax/
- Extension für VSCode: `Markdown Preview Enhanced`

Markdown (`*.md`) hat sich in den letzten Jahren ebenfalls als Standard zur einfachen Formatierung von Texten etabliert. Mit voranschreitender Entwicklung des Projekts wird eine umfangreichere Dokumentation notwendig. Es existieren eine Reihe von Werkzeugen, welche auf Basis von Markdown automatisch ergonomische statische Webseiten generieren und direkt bspw. via Diensten, wie GitHub-Pages zur Verfügung stellen. Die Dokumentation findet somit synchron mit den Komponenten im Repository statt und wird über das Versionierungssystem überwacht.

4. Lizenzierung

Allgemeines

Alle Source-Codes/-Designs unterliegen dem Urheberrecht der urhebenden Person. Im dt. Recht ist das Urheberrecht ein Grundrecht und somit nicht übertragbar (Urheberpersönlichkeitsrecht), ein Lizenzvermerk (vgl. Copyright etc.) ist nicht notwendig. Nutzenden Personen werden daher Nutzungsrechte mittels einer Lizenz eingeräumt. Open Source Lizenzen sind “Standard-Nutzungslizenzen” im Urheberrecht und regeln die Verwendung, Veränderung und Verteilung (Veröffentlichung, Verwertung) der bereitgestellten Dateien. Keine Angabe bedeutet “Alle Rechte vorbehalten”: d.h. jegliche Nutzung ohne Erlaubnis der urhebenden Person stellt eine Urheberrechtsverletzung dar!

OS-Lizenzen werden grundlegend nach ihrer Copyleft-Wirkung unterschieden:

- freizügig (permissive)
- schwach (weak)
- stark (strict)

Dokumentiere und prüfe daher akribisch die Lizenzen der verwendeten Source-Codes und -Designs und deren Kompatibilität zueinander!

Empfehlung: Nutze das `REUSE` -Tool[4]) zur Lizenzverwaltung!

- Link: https://github.com/fsfe/reuse-tool

4: https://reuse.software

Empfohlene Lizenzen

Sofern die verwendeten externen Komponenten dies zulassen, sollten die eigenen Entwicklungen so "stark" wie möglich in ihrem Copyleft lizenziert werden, d.h.:

Strikte Lizenzierung:

- Software: AGPL-3.0-or-later
- Hardware und Elektronik: CERN-OHL-S-2.0-or-later
- Dokumentation: CERN-OHL-S-2.0-or-later

Die urhebende Person darf zu jedem Zeitpunkt ihre Dateien neu Lizenzieren. Sobald mehrere Personen an einer Datei gearbeitet haben wird diese urheberrechtlich zu einem Gemeinschaftswerk, daher ist es wichtig die Zusammenarbeit in einer *Contributors License Agreement* zu regeln. Ansonsten bedarf es der Zustimmung jeder beteiligten Person! Eine Lizenzänderung wirkt sich nicht auf die in der Vergangenheit in Umlauf gebrachte Dateien aus sondern immer nur auf die aktuell bereitgestellten. Alte Versionen sind daher schwer wieder "einzufangen" und können beliebig weiterlizenziert werden (daher von strikt zu freizügig).

Nachteil dieses Vorgehens ist, dass sich strikt lizenzierte Designs schwerer mit anderen Komponenten verknüpfen lassen ohne diese in ihrer Lizenzierung zu beeinflussen. Was sich wiederum als Hürde hinsichtlich der Akzeptanz und Verbreitung darstellen könnte. Eine zu freizügige Lizenzierung lässt jedoch Verwertungsstrategien wie eine Dual-Lizenzierung unbedeutend werden. Als Kompromiss wird daher oft eine schwache Lizenzierung gewählt.

Die strikte Variante der CERN-OHL 2 (CERN-OHL-S-2.0) schließt Netzwerkdienste als Verbreitungsform nicht mit ein, weshalb für eine strikte Lizenzierung abweichend für Software die AGPL-3.0 gewählt werden sollte.

Für die Dokumentation wird häufig eine `CC-BY-SA-4.0` angewendet. Diese schreibt jedoch die Veröffentlichung der Quelldateien der Dokumentation nicht vor. Die einzige Lizenz, welche in ihrem Wortlaut unabhängig von Software genutzt werden kann und die Offenlegung der Quelldateien vorschreibt ist aktuell die CERN-OHL-2.0 in der schwachen und strikten Form.

Für die freizügige und schwache Lizenzierung sind daher für alle Aspekte jeweils die Varianten der CERN-OHL 2 anwendbar:

schwach: CERN-OHL-W-2.0-or-later freizügig: CERN-OHL-P-2.0-or-later

5. Prinzipien

In der Software-Entwicklung stößt man immer wieder auf sehr zentrale Grundprinzipien. Auch wenn sich diese auf Software beziehen so lassen sich durch aus Parallelen zur Entwicklung von Konstruktionen ziehen. Hier sollen einige Prinzipien genannt werden, die als Hilfestellung und Grundlage für (Design-)Entscheidungen dienen sollen. Nicht selten widersprechen sich Regeln. Spätestens dann muss von ihnen abgewichen werden. Entscheidend ist sich bewusst zu sein, welche Konsequenzen das Verletzen der Regeln haben wird. Sie dienen daher als Ausgangspunkt einer Entwicklung. “Breche niemals die Regeln, außer du weißt du was du tust.”

Bekannt als Unix-Philosophie fasst Raymond [7] diese in 17 Regeln zusammen:

1. Regel der Modularität: Schreibe einfache Bestandteile, die durch saubere Schnittstellen verbunden werden.
2. Regel der Klarheit: Klarheit ist besser als Gerissenheit.
3. Regel des Zusammenbaus: Entwirf Programme so, dass sie mit anderen Programmen verknüpft werden können.
4. Regel der Trennung: Trenne den Grundgedanken von der Umsetzung, trenne die Schnittstellen von der Verarbeitungslogik.
5. Regel der Einfachheit: Entwirf mit dem Ziel der Einfachheit; füge Komplexität nur hinzu, wenn es unbedingt sein muss.
6. Regel der Sparsamkeit: Schreibe nur dann ein großes Programm, wenn sich klar zeigen lässt, dass es anders nicht geht.
7. Regel der Transparenz: Entwirf mit dem Ziel der Durchschaubarkeit, um die Fehlersuche zu vereinfachen.
8. Regel der Robustheit: Robustheit ist das Kind von Transparenz und Einfachheit.
9. Regel der Darstellung: Stecke das Wissen in die Datenstrukturen, so dass die Programmlogik dumm und robust sein kann.

10. Regel der geringsten Überraschung: Mache beim Entwurf der Schnittstellen immer das Nächstliegende, welches für die wenigsten Überraschungen beim Benutzer sorgt.
11. Regel der Stille: Wenn ein Programm nichts Überraschendes zu sagen hat, soll es schweigen.
12. Regel des Reparierens: Wenn das Programm scheitert, soll es das lautstark und so früh wie möglich tun.
13. Regel der Wirtschaftlichkeit: Die Arbeitszeit von Programmierern ist teuer; spare sie auf Kosten der Rechenzeit.
14. Regel der Code-Generierung: Vermeide Handarbeit; schreibe Programme, die Programme schreiben, falls möglich.
15. Regel der Optimierung: Erstelle Prototypen, bevor du dich an den Feinschliff machst. Mache es lauffähig, bevor du es optimierst.
16. Regel der Vielseitigkeit: Misstraue allen Ansprüchen auf „den einzig wahren Weg“.
17. Regel der Erweiterbarkeit: Entwirf für die Zukunft, denn sie wird schneller kommen als du denkst.

Ergänzend seien folgende Regeln zu nennen:

- “Konvention vor Konfiguration” (siehe diese Richtlinie)
    - Eine Konvention/Richtlinie reduziert späteren Dokumentationsaufwand

7: Deutsche Übersetzung von: https://cdn.nakamotoinstitute.org/docs/taoup.pdf, entnommen aus: https://de.wikipedia.org/wiki/Unix-Philosophie

Anhang

OSH Directory Standard - Unixish Version

```
.
|-- LICENSES
| `-- AGPL-3.0-or-later.txt
|-- doc
| |-- assembly
| | |-- chap-1
| | | |-- sec-1.md
| | | `-- sec-2.md
| | |-- main.md
| | `-- vf_recipe.ttl
| |-- history
| | `-- main.md
| |-- manuf
| | |-- chap-1
| | | |-- sec-1.md
| | | `-- sec-2.md
| | `-- main.md
| |-- recycling
| | |-- chap-1
| | | |-- sec-1.md
| | | `-- sec-2.md
| | `-- main.md
| |-- usr
| | |-- chap-1
| | | |-- sec-1.md
| | | `-- sec-2.md
| | `-- main.md
| `-- workshops
|     `-- build
|         `-- main.md
|-- gen
| |-- anim
| |-- calc
| |-- doc
| | |-- assembly
| | |-- manuf
| | |-- recycling
| | `-- usr
```

```
| |-- elec
| |-- firmware
| |-- mech
| |-- sim
| |-- site
| |-- software
| |-- assembly.vf_recipe.ttl
| |-- bom.csv
| |-- manuf.vf_recipe.ttl
| |-- okh.toml
| `-- recycle.vf_recipe.ttl
|-- mod
| |-- module-a
| `-- module-b
|-- res
| |-- assets
| | |-- media
| | | |-- img
| | | | |-- animation-1.gif
| | | | |-- drawing-1.png
| | | | |-- partially-auto-generated-drawing.pdf
| | | | |-- photo-1.jpg
| | | | `-- photo-2.webp
| | | `-- vid
| | |     `-- tutorial.webm
| | `-- var
| |     `-- datasheet-x.pdf
| |-- conf
| | |-- VisiCut-config.plf
| | `-- some-config.xml
| `-- media
|     `-- img
|         `-- diagram-1.svg
|-- run
| |-- build
| |-- ci
| |-- clean
| |-- release
| `-- test
|-- src
| |-- anim
| | `-- assembly.blend
| |-- calc
```

```
| | |-- feedback-evaluation.ods
| | `-- required-material-thickness.m
| |-- elec
| | |-- main-subs
| | | |-- parts.lib
| | | |-- sub-a.kicad_pcb
| | | |-- sub-a.pro
| | | |-- sub-a.sch
| | | |-- sub-b.kicad_pcb
| | | |-- sub-b.pro
| | | `-- sub-b.sch
| | |-- main.kicad_pcb
| | |-- main.pro
| | |-- main.sch
| | |-- plate.kicad_pcb
| | |-- plate.pro
| | `-- plate.sch
| |-- firmware
| | |-- Makefile
| | |-- main.c
| | |-- other.c
| | `-- other.h
| |-- mech
| | |-- case-alternative.fcstd
| | |-- case.fcstd
| | |-- parts.lib
| | `-- pieceX.scad
| |-- sim
| | `-- stress-test.OpenFOAM
| |-- software
| | `-- control_lib
| |     |-- src
| |     | `-- main.rs
| |     `-- Cargo.toml
| `-- test
|-- LICENSE.txt
|-- README.md
|-- assembly.vf_recipe.ttl
|-- bom.csv
|-- bom.csv.license
|-- bom.ods
|-- manuf.vf_recipe.ttl
```

```
|-- okh.toml
`-- recycle.vf_recipe.ttl
```