Ade, DTP!

Michael Marek

Ade, DTP!
Markdown, Git und LLM - Eine Alternative für technische Redaktionen in KMU.

Beschreibung

Der vorliegende Artikel beschreibt einen Ansatz, wie KMU technische Dokumentationen ohne DTP- oder CCMS-Systeme erstellen können und stattdessen mit Markdown, eigenem Git und lokalem LLM arbeiten. Er ermöglicht kostengünstige, kollaborative und offene Workflows für kleine Teams mit begrenztem Budget, die Datenhoheit und Flexibilität benötigen. Herausforderungen wie normorientierte Standardisierungen oder komplexe Formatierungen lassen sich durch technische Modifikationen bewältigen.

Vorbemerkung

Der vorliegende Artikel, erstmals veröffentlicht am 30.07.2026, ist noch nicht abgeschlossen; Änderungen oder Ergänzungen können folgen.

Hintergrund

Während meines Zertifikatslehrgangs *Technische Redaktion (TAE)* im ersten Halbjahr 2026 an der Technischen Akademie Esslingen habe ich circa 50 Redakteurinnen und Redakteure und deren Arbeitsprozesse kennengelernt. Ein Teil von ihnen, aus kleinen und mittelständischen Unternehmen (KMU), arbeitet mit Microsoft Word oder Adobe InDesign und ist mit den Abläufen zum Bearbeiten, Versionieren etc. oder dem Datenaustausch mit externen Dienstleistern (Agenturen, Übersetzungsbüros) alles andere als zufrieden. Ein anderer Teil, aus größeren Unternehmen, arbeitet mit sogenannten *Component-Content-Management-Systemen* (CCMS), wobei zum einen die Lizenzkosten und zum anderen die Komplexität dieser Systeme auch nicht in allen Fällen zufriedenstellend sind. Vor diesem Hintergrund wurde zur Abschlussprüfung eine Präsentation verfasst, die Ausgangspunkt für diesen Artikel war.

Ziel und Zielgruppe

Ziel ist es, einen Ansatz für technische Redaktionen aufzuzeigen, der ohne DTP-Applikationen oder CCM-Systeme auskommt. **Zielgruppe** sind somit KMU, mit einer geringen Anzahl von Produkten und Varianten, die Bedienungsanleitungen und Dokumentationen erstellen.

Nachteile von DTP

Durch eingeschränkte oder fehlende…

- Rollen- und Rechtekonzepte,
- Verwaltung von Status und Versionen,
- Verwaltung von Varianten,
- Verwaltung von Fremdsprachen,
- Wiederverwendung von Inhalten,
- Trennung von Struktur, Inhalt und Layout,
- Grundlagen für SingleSource-/CrossMedia-Publishing,
- Suchfunktion durch unzureichende Metadaten,
- Migration durch binäre, z. T. proprietäre Formate,
- Integration durch systemspezifische Schnittstellen.

Quelle: Vgl. Marco Hattemer, Informationsmanagement in der TK, Folie Nr. 6. Vortrag im Rahmen des Zertifikatslehrgangs.

Datensilo DTP

1. Transfer von Dateien zwischen Abteilungen oder zu externen Dienstleistern sind fehleranfällig und zeitraubend; Die Nachteile sind bekannt.
2. Steigende Lizenzkosten der *Quasimonopole* und damit einhergehende Abhängigkeiten sind für Unternehmen schwer vorhersehbar und riskant.

Datenraffinerie CCMS

1. CCM-Systeme tendieren zur Abschottung von Daten, auf die nur durch spezifische Schnittstellen (APIs) zugegriffen werden kann, um sie für verschiedene Medienformate und Anforderungen (Knowledge-Management, Online-Hilfe, Service/Support etc.) bereitzustellen.
2. Hohe Kosten für Lizenzen, Implementierung und Wartung stehen oftmals in keiner optimalen Relation zum Nutzen innerhalb kleiner Teams.
3. CCM-Systeme sind nach eigenen Angaben erst ab einer erhöhten Anzahl von Mitarbeiter:innen, Produkten, Varianten etc. angebracht.

Datenhoheit - Zugriff, Schutz und Sicherheit

1. Microsoft 365, Adobe CC und CCMS-Anbieter mit cloudbasierten Lösungen setzen auf die Infrastruktur von Amazon AWS, Microsoft Azure und Google Cloud.
2. Dadurch bestehen erhöhte Risiken für die *eigenen* Daten (CloudAct).
3. Offene, europäische oder internationale Alternativen sind essentiell und existenziell.

Datenworkflow - Quellenvielfalt und Findbarkeit

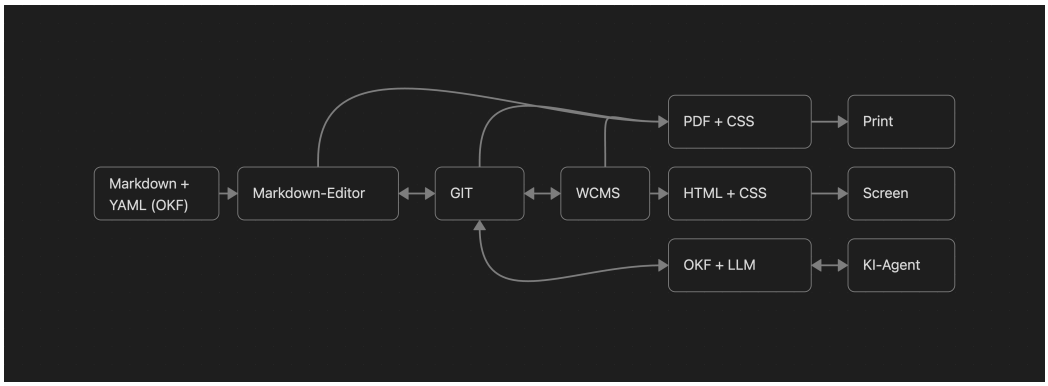
> *„Alle paar Wochen füge ich denselben Textblock in ein Chatfenster ein. Hier ist unsere API-Struktur. So sind unsere Fehlercodes organisiert. Das bedeutet Integration in diesem Zusammenhang. Dieses Modell hilft […] Der Chat endet und alles ist vergessen. [Solch] Ein „Speicherleck“ ist kein Arbeitsablauf.“*

Zitat: Steve Arrants, Technical Writer

Grundsätzliche Anforderungen an eine technische Lösung

1. Editieren - Menschen und Maschinen.
2. Klassifizieren - Metadaten und Findbarkeit.
3. Versionieren - Kollaboration und Qualitätssicherung.
4. Variieren - Modifikationen, Märkte und Übersetzungen.
5. Modularisieren - Effizienz und Wiederverwendbarkeit.
6. Exportieren - Medien, Apps, Datenbanken und KI.

Ansatz



Die technische Lösung ist nicht die einzige Herausforderung, sondern die Usability; Die komfortable, zufriedenstellende, effektive und effiziente Benutzung der „orchestrierten“ und zum Teil „ungewohnten“ Applikationen über den gesamten, dargestellten Prozess.

Warum Markdown, YAML und OKF?

Markdown, zusammen mit der Meta-Informationsebene YAML/Frontmatter respektive den Vorgaben aus der OKF-Konvention, entspricht dem Ansatz *Docs as Code*. Redakteure, Entwickler etc. und KI-Systeme können Markdown und YAML lesen und schreiben. Dieser in der Software-Branche etablierte Einsatz ist in zahlreichen anderen Branchen vorstellbar - oder zumindest eine Evaluation wert. *Docs as Code* bezeichnet einen Ansatz, nach der Dokumentationen mit denselben Applikationen wie (Programm-)Code erstellt werden können. Zum Beispiel zur:

- Problem-/Aufgabenverfolgung,
- Versionskontrolle,
- Bearbeitung von „Klartext“,
- Codeüberprüfung und
- Durchführung (semi-)automatischer Tests.

Markdown :: Aufbau

ELEMENT	XML	HTML	MARKDOWN
Überschrift	`<title>Überschrift</title>`	`<h1>Überschrift</h1>`	`# Überschrift`
Absatz	`<paragraph>Text</paragraph>`	`<p>Text</p>`	`Text`
Hinweis	`<note>Hinweis</note>`	`<p class="note">Hinweis</p>`	`> Hinweis`
Beispieltext	`<text>Beispiel</text>`	`<p>Beispiel</p>`	`Beispiel`
Fett	`<bold>Fett</bold>`	`<strong>Fett</strong>`	`**Fett**`
Kursiv	`<italic>Kursiv</italic>`	`<em>Kursiv</em>`	`*Kursiv*`

Exemplarisch werden folgende Markdown-Editoren aufgeführt:

1. Obsidian: Kostenlos, im B2B-Umfeld kostenpflichtig. Schwerpunkt im Knowledge-Management von Notizen auf Markdown-Basis.
2. iA Writer: Kostenpflichtig. Dokumentenorientierter Markdown-Editor.
3. iA Presenter: Kostenpflichtig. Vortragorientierter Markdown-Editor.
4. PHPStorm: Kostenpflichtig. Entwicklungsgumgebung für Websites respektive Webapplikationen; Als Markdown-Editor einsetzbar.

YAML/Frontmatter :: Aufbau

```
---
title: "Beispiel-Dokument"
author: "Michael Marek"
date: 2026-07-15
license: CC-BY-4.0
---
```

Erläuterungen

YAML™ ist eine Datenserialisierungssprache, die auf Datenstrukturen agiler Programmiersprachen basiert. *Frontmatter* ist ein von Menschen und Maschinen lesbarer *Datei-Header*, der Metadaten für die Verarbeitung bereitstellt (Vgl. Titelei).

OKF :: Definition

Das **Open Knowledge Format**, vorgestellt am 12.06.2026 in Version 0.1, ist eine offene Spezifikation, die Daten und Informationen als Verzeichnis von Markdown-Dateien mit YAML/Frontmatter strukturiert. Jede Datei steht für ein „Konzept“ – eine Tabelle, einen API-Endpunkt, eine Metrik […] oder alles, was [einzeln] erfasst werden kann bzw. soll. OKF formalisiert verteiltes Wissen in ein *portables, interoperables Format*, das von Menschen und KI-Agenten gelesen, geschrieben und ausgetauscht werden kann - ohne Abhängigkeiten von herstellerspezifischen Applikationen oder Entwicklungsumgebungen.

OKF :: Header :: Aufbau

```
---
type: <Typ>                     # Pflicht!
title: <Titel>                  # Empfohlen
description: <Beschreibung>     # Empfohlen
resource: <URI>                 # Empfohlen
tags: [tag1, tag2]              # Empfohlen
timestamp: 2026-07-05T12:00:00Z # Empfohlen (ISO 8601)
weitere felder:                 # Erlaubt
---
```

Erläuterungen

- **type**, **title** und **description(!)** sind elementare Felder für die technische Redaktion.
- **resource** sind physische Ressourcen wie APIs, Datenbanken, Dateien, Repos etc.
- **tags** und **timestamp** können automatisiert eingefügt werden.

- Unterhalb eines solchen *Headers* befindet sich der Inhaltsbereich mit Markdown und bei Bedarf Abschnitte wie Schema, Examples, Citations.

„Irgendjemand in Ihrer Organisation wird diese Beschreibung(!) ausfüllen. Die Frage ist, ob es jemand sein wird, der zwischen dem was in einem Dokument steht, und dem was es bedeutet, unterscheiden kann.“

Zitat: Steve Arrants, Technical Writer

OKF :: Verzeichnis :: Aufbau

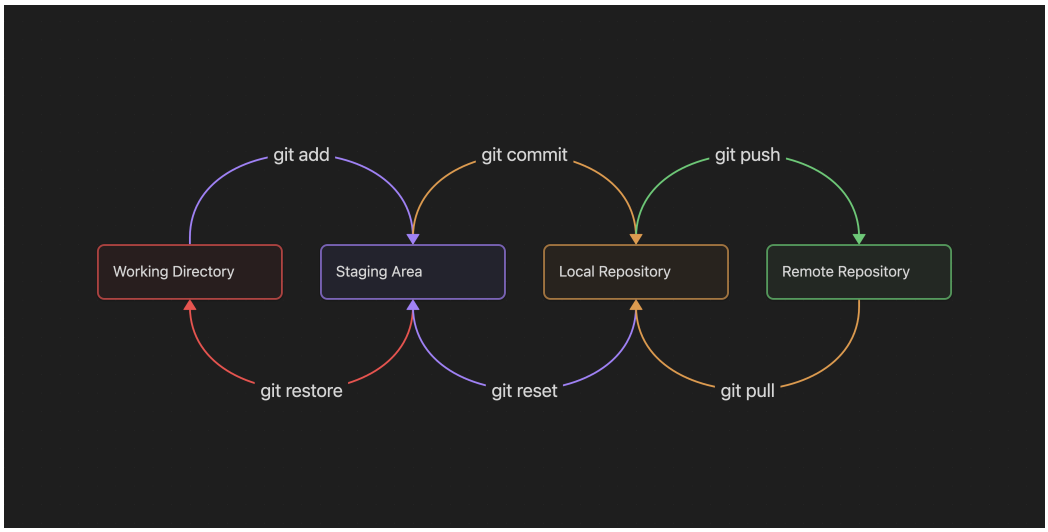
```
pfad/zum/bundle/
├── index.md     # Optional. Verzeichnisübersicht (Schrittweise).
├── log.md       # Optional. Verlauf (Chronologisch)
├── <Konzept>.md # Konzept im Stammverzeichnis.
└── <Unterverzeichnis>/ # Unterverzeichnisse (Konzepte in Gruppen)
    ├── index.md
    ├── <Konzept>.md
    └── <Unterverzeichnis>/
        └── …
```

OKF im Vergleich zu DITA

KATEGORIE	OKF (MARKDOWN/YAML)	DITA (XML/CCMS)
Wiederverwendbarkeit	SSG-Transklusion (Hugo-Shortcodes, MkDocs-Snippets), YAML-Variablen; kein natives Äquivalent zu „conref“.	Tiefe Inhaltsverweise (conref/conkeyref) und hierarchische Zuordnungen.
Governance	Dezentraler Git-Flow, Markdown-Linter, Pull-Request-Prüfungen.	Zentralisiertes CCMS, strenge XML-Schema-/DTD-Validierung, sperrbasierte Bearbeitung.
Toolchain	Entwickler-IDEs (VS Code), CI/CD-Pipelines, statische Website-Generatoren.	Spezialisierte XML-Editoren (Oxygen), DITA Open Toolkit, CCMS-Datenbanken.
Endnutzer	Hohe Lesbarkeit für LLM/Agenten, Web-First-Navigation, responsive Dokumentation.	Multi-Channel-Veröffentlichung (PDF, HTML5, EPUB, Eclipse Help); hohe Wiedergabetreue der strukturierten Ausgabe.
Modularität	Durch Manifest definierte Bundle-Struktur (bundle.yaml), für Menschen lesbare Verzeichnisverschachtelung.	Themenbasierte Architektur, metadatenreiche Prologe, Ditamaps.

Quelle: Steve Arrants, Technical Writer

Grundlegendes Konzept von Git



1. Repository: Ein Verzeichnis, in dem Git alle Änderungen und Versionen einer Dateien speichert; *Working Directory*, *Staging Area* und *Local Repository* befinden sich auf dem lokalen Rechner; Das *Remote Repository* auf einem Server.
2. [Git] Add: *Add* erfasst gewünschte Dateien/Verzeichnisse zu einem Zeitpunkt.
3. [Git] Commit: *Commit* überträgt ausgewählte Daten, besitzt zur Kontrolle und Nachverfolgung eine einzigartige ID (Hash) und eine vom Entwickler, Editor etc. einzugebende Nachricht, die beschreiben sollte, was, warum geändert wurde.
4. Branch: Ein *Branch* ist ein unabhängiger *Entwicklungszweig* in einem Repository. Standardmäßig gibt es einen *Hauptzweig*, den main- oder master-Branch. Durch *Branches* kann parallel gearbeitet werden, ohne den Hauptzweig zu beeinflussen. Zum sicheren Wechsel zwischen Branches und zu deren jeweiligen Bearbeitung, inklusive Aktualisierung und Speicherung, dient der Befehl [Git] Checkout.
5. Merge: Ein Merge ist das Zusammenführen von Änderungen aus einem Branch in einen anderen z. B. von einem Entwicklungszweig in den Hauptzweig. Zudem gibt es den *Pull-/Merge Request*; Ein Vorschlag, Änderungen in ein anderes Repository zu integrieren.
6. [Git] Push: *Push* überträgt Commits aus dem lokalen Repo in ein remote Repo.
7. [Git] Pull: *Pull* holt Änderungen aus einem remote Repo in das lokale Repo.
8. [Git] Reset: *Reset* setzt den aktuellen Branch auf einen Commit zurück und ändert dabei optional die Staging Area und/oder das Working Directory.
9. [Git] Restore: *Restore* stellt Dateien im Working Directory oder in der Staging Area aus einem Commit wieder her, ohne den Branch-Verlauf zu ändern.

Diese 9 Punkte sind eine reduzierte Übersicht der Funktionalitäten und der Leistungsfähigkeit von *GIT*. Die Lernkurve ist beim erstmaligen Einsatz hoch, streng genommen aber nicht höher als die Lernkurven beim Einsatz von DTP-Applikationen inklusive zentraler Dateiablage oder von komplexen CCM-Systemen. Git kann via Terminal (CLI) bedient und gesteuert werden. Terminal-Eingaben zu Nutzen ist „tägliches Brot“ für professionelle Anwender:innen aus der Softwareentwicklung; Eine Alternative bzw. sinnvolle Ergänzung sind grafisch orientierte Desktopapplikationen - wie z. B. *Tower* oder *Fork* - welche die Arbeit mit Git nicht nur für Einsteiger:innen erleichtert. Weitreichende und umfänglichere Informationen zum Umgang mit Git finden sich u. a. beim Hersteller der Software *Tower*.

Vorteile von Git

1. Versionskontrolle und Historie

- Jede Änderung an der Dokumentation ist nachvollziehbar.
- Rollbacks sind einfach möglich, falls Fehler auftreten.

2. Zusammenarbeit und Parallelisierung

- Branches ermöglichen es, verschiedene neue Funktionen, Übersetzungen, Korrekturen etc. parallel zu bearbeiten.
- Pull Requests (PRs) ermöglichen Reviews vor der Integration in den Hauptzweig.

3. Dezentrale Arbeit

- Externe Dienstleister oder Remote-Teams können lokal arbeiten und ihre Änderungen später einpflegen.
- Grundsätzlich keine Abhängigkeit von einer zentralen Plattform (Saas vs. Selfhosting).

4. Integration mit Applikationen

- Git lässt sich mit Markdown, AsciiDoc, reStructuredText, DITA(!), Sphinx und anderen Dokumentationsformaten kombinieren.
- CI/CD-Pipelines z. B. für automatische PDF-Generierung oder Übersetzungsworkflows sind möglich.

5. Automatisierung und KI-Unterstützung

- KI-Agents können z. B. für automatische Übersetzungen, Rechtschreibprüfung, Inhaltsgenerierung etc. über Git-Hooks oder Bots eingebunden werden.
- Webhooks können z. B. Übersetzungsdienste (DeepL, Lokalise) oder Dokumentations-Builder (Sphinx, MkDocs) anstoßen.

Eine m. E. lohnes- und erwähnswerte Alternative zu der vorherrschenden Git-Plattform *GitHub* von Microsoft sind *Codeberg e.V.* aus Berlin, als SaaS-Angebot (Software As A Service) unter bestimmten Voraussetzungen, bzw. deren technologische Basis *Forgejo*, als FOSS (Free Open Source Software) erhältlich.

WCMS, PDF, HTML und CSS

1. Offene WCM-Systeme, die dateibasiert - ohne Datenbank - arbeiten und Markdown als HTML rendern können; Zum Beipiel:
2. Kirby CMS; Ein WCMS auf Basis von PHP/Laravel und dateibasierter (*.md, nativ) Erfassung (Keine Datenbank!) Lizenzkosten pro Version (mit großzügiger Interpretation, wann eine Version endet) und immer eine Lizenz pro Website/URL/Domain.
3. Offene PDF-Renderer, die an Markdown-Editoren, Repos via CI/CD-Pipelines oder WCM-Systeme angebunden werden können; Zum Beispiel:
4. Als Ausgangspunkt: PrintCSS und verschiedene Optionen zum direkten Erstellen von PDF-Dokumenten aus den Markdown-Editoren heraus.
5. Offene, branchenspezifische, standardisierte und normorientierte HTML- und CSS-Vorlagen; z. B. analog zu den Bestrebungen seitens tekom Deutschland e.V. in den Bereichen XML/XLST, DITA, Terminologie etc.

Potentiale beim Einsatz von LLM und KI

Aktive…

1. Überwachung/Kontrolle von Versionen, Varianten und *normgerechter Strukturierungen* (Makro- & Mikrosequenzierung) und *Formulierungen* (Sequenzmuster).
2. Neu-Generierung von Varianten und Dokumenten.
3. Unterstützung/Übernahme von Übersetzungen.
4. Bereitstellung für Wikis, Service und Support.

LLM und KI-Anbieter - Nicht die üblichen Verdächtigen

1. *Mistral Vibe* (Frankreich): Leistungsfähiges LLM - auch als offenes Modell (Mistral 3) bereitstellbar über eigene, interne Server - und auf *Augenhöhe* mit den aktuellen Modellen der US-Anbieter.
2. *Nenna AI* (Deutschland): Datenschutzkonforme Anbindung verschiedner KI-Anbieter mit integrierbaren und *anonymisierten* Platzhaltern.
3. *Hugging Face* / *Ollama* (Weltweit): Open-Source-Software/Plattform zur lokalen Implementierung großer LLM.

Gibt es Nachteile oder Schwächen?

Sicher! Anbei ein paar davon.

Markdown: Vor- und Nachteil

- Markdown ist im Gegensatz zu HTML (via W3C) nicht standardisiert, was zu einer unterschiedlichen Syntax oder Formatinterpretation (z. B. Images, Callouts etc.) in den verschiedenen Markdown-Editoren und WCM-Systemen führt.
- Gleichzeitig bieten diese Unterschiede die Option, eine unternehmens- oder branchenspezifische, also standardisierte und normorientierte Syntax und Formatinterpretation zu entwickeln.

Markdown: Schwächen

Referenzierungen
Keine spezifische Schwäche von Markdown, sondern eine grundsätzliche Herausforderung zwischen *Print und Screen*. Workaround: Redaktionelle Vorgaben plus Scripting z. B. mit Hilfe von *Python, Pandoc, LaTeX* oder kommerziellen Lösungen, z. B. *PrinceXML* bzw. die SaaS-Lösung *EuroPDF*.

Darstellungen
Spezifische Icons/Piktogramme und Absätze für verbindliche Signalwörter gemäß Richtlinien (ANSI Z535). **Lösung: Markdown-Syntax und CSS entwickeln.** (Vgl. nachfolgende Ausführungen.)

Bearbeitungen
Umfangreiche Tabellen sind für responsive Bildschirmdarstellungen leider grundsätzlich ungeeignet. Option: Geeignete Markdown-Editoren auswählen, die das Einfügen und komfortable Bearbeiten von Tabellen unterstützen z. B. *iA Presenter*.

Markdown-Syntax entwickeln
Einfach zur merkende Einfügungen für die technische Redaktion.

```
> [!DANGER] Gefahr
> Hier steht ein dringender Hinweis mit ANSI Z535.6-konformer Darstellung.
> **Jeden Kontakt mit der Substanz vermeiden!**

> [!WARNING] Warnung
> Hohe Temperaturen können zu Verbrennungen führen.
> **Schutzhandschuhe tragen!**

> [!CAUTION] Achtung
> Dies ist eine Vorsichtsmaßnahme.
> **Komplette Umgebung vor dem Start prüfen.**

> [!NOTE] Hinweis
> Dies ist ein allgemeiner Hinweis.
> *Das vorliegende Benutzungshandbuch ist für Fachpersonal bestimmt.*
```

CSS entwickeln
Entsprechende Formatanweisungen für das Rendering als PDF und HTML.

```
/* ANSI Z535.6 Farben */

.callout--danger {
  --callout-color: #BD2024;
}
.callout--warning {
  --callout-color: #FF7900;
}
.callout--caution {
  --callout-color: #FFE100;
}
.callout--note {
  --callout-color: #004488;
}
```

Erstes Ergebnis: PDF

ANSI Z535.6

Gefahr

Hier steht ein dringender Hinweis mit ANSI Z535.6-konformer Darstellung.
Jeden Kontakt mit der Substanz vermeiden!

Warnung

Hohe Temperaturen können zu Verbrennungen führen.
Schutzhandschuhe tragen!

Achtung

Dies ist eine Vorsichtsmaßnahme.
Komplette Umgebung vor dem Start prüfen.

Hinweis

Dies ist ein allgemeiner Hinweis.
Das vorliegende Benutzungshandbuch ist für Fachpersonal bestimmt.

Erstes Ergebnis: HTML

ANSI Z535.6

GEFAHR

Hier steht ein dringender Hinweis mit ANSI Z535.6-konformer Darstellung.
Jeden Kontakt mit der Substanz vermeiden!

WARNUNG

Hohe Temperaturen können zu Verbrennungen führen.
Schutzhandschuhe tragen!

ACHTUNG

Dies ist eine Vorsichtsmaßnahme.
Komplette Umgebung vor dem Start prüfen.

HINWEIS

Dies ist ein allgemeiner Hinweis.
Das vorliegende Benutzungshandbuch ist für Fachpersonal bestimmt.

Nachteile von DTP werden eliminiert durch…

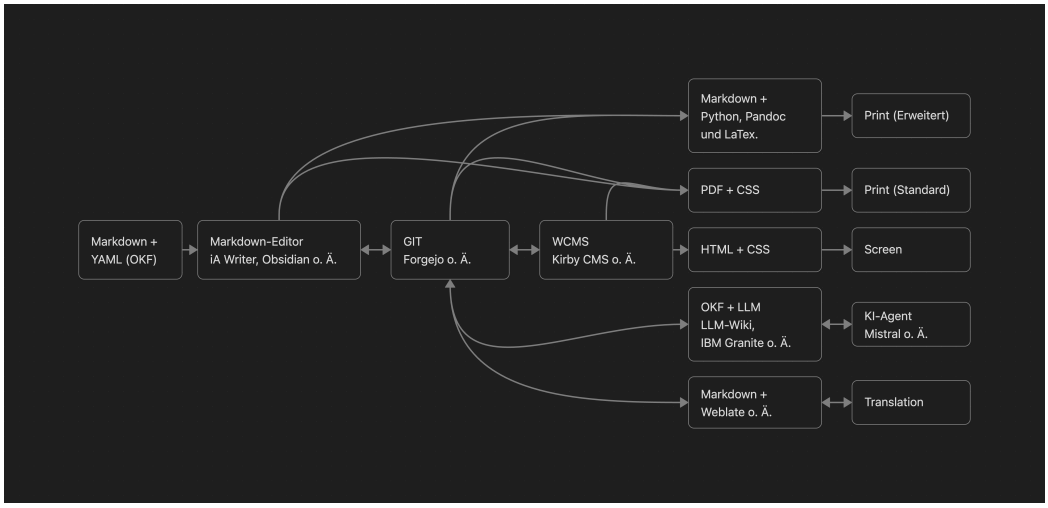
Git:

- Rollen- und Rechtekonzepte.
- Verwaltung von Status und Versionen.
- Verwaltung von Varianten.
- Verwaltung von Fremdsprachen.
- Wiederverwendung von Inhalten.

Markdown & YAML/OKF:

- Trennung von Struktur, Inhalt und Layout.
- Grundlagen für SingleSource-/CrossMedia-Publishing.
- Suchfunktion durch ~~unzureichende~~ Metadaten.
- Migration durch ~~binäre, z. T. proprietäre~~ offene Formate.
- Integration durch ~~systemspezifische~~ offene (und reduzierte) Schnittstellen.

Erweiterter Ansatz



Hinweise

Inwieweit Markdown-Editoren eventuell durch entsprechende Editoren innerhalb eines WCMS ergänzt oder ersetzt werden können, ist Aufgabe der laufenden Evaluation.

Sofern weiterhin oder zukünftig der „dringende“ Bedarf besteht, Adobe InDesign o. ä. Satzprogramme einzusetzen, könnten folgende zwei Aspekte hilfreich sein:

1. Adobe bietet mit *InCopy* respektive *RoboHelp* Optionen zum Importieren und Editieren von Markdown-Dateien; Bei anderen Anbietern könnten ähnliche Lösungen existieren.
2. Zur Gestaltung von Print-Publikationen sollten DTP-Applikationen für ihre vorrangige Aufgabe eingesetzt werden - Nicht als Schreibprogramm, sondern als Satzprogramm. Exemplarische Musterseiten für Print-Publikationen könnten „wie gewohnt“ erstellt werden, ihre Genehmigungsprozesse durchlaufen etc. und dann als Vorlage(!) für die gewünschten PDF- und HTML-Renderer bzw. die Erstellung der CSS herangezogen werden.

Ade, DTP? - Kein Abschied fällt leicht…

1. Markdown und Git - und LLM - sind eine Alternative für technische Redaktionen in KMU.
2. Es existieren zahlreiche, kommerzielle oder offene Editoren und Applikationen; Einsetzbar nach Bedarf, Know-how und vorhandenem Budget.
3. Es bedarf keinen komplexen APIs für Übersetzungen oder KI-Anwendungen.
4. Es verhindert interne *Gatekeeper* und externe *Wallet Garden* - oder vermindert zumindest deren Anzahl.

Nächste Schritte

1. Definition einer Markdown-Syntax und CSS-Vorlage insbesondere für Warnhinweise und Normvorgaben.
2. Auswahl geeigneter Markdown-Editoren und grafisch orientierter Git-Clients, um das *Editieren* der Dokumentationen zu erleichtern und den *Blick auf Git* für Redakteur:innen *benutzbarer* zu machen.
3. Evaluation von *OKF* oder möglicher Alternativen und deren Einsatzfähigkeiten für Markdown-Editoren/WCMS/PDF/HTML/KI.
4. Festlegung eines optimalen WCMS zur Darstellung von Dokumentationen als Website bzw. Webapplikation und einer den Anforderungen der Druckindustrie genügenden PDF-Renderlösung.
5. Überprüfung der technischen Machbarkeit und anschließende Umsetzung für RAG, LLM und KI-Chatbots bzw. -Agents.
6. Weitergehende Marktrecherchen und eventuell die Erstellung eines konventionellen Business-Plans, um KMU den vorgestellten Ansatz als *Komplettlösung* respektive Dienstleistung anbieten zu können.

Interesse an einem Pilotprojekt?

Falls Sie den Beitrag bis hier gelesen haben, gehe ich davon aus, dass Sie das Thema interessiert ;-) Lassen Sie uns reden und feststellen, ob dieser Ansatz für Ihr Unternehmen in Frage kommt.

- E-Mail-Adresse: mm -at- ergonomicon -punkt- de
- Via (link: https://signal.org/ text: Signal): micmar.10
- Via (link: https://matrix.org/ text: Matrix): @micmar:matrix.org