Dokumentationswerkzeuge für Sicherheitszertifizierungen

Alexander Krumeich

Zusammenfassung

Dieser Beitrag beschreibt ein technisches Verfahren für die Erstellung systematisch strukturierter Dokumentation, entwickelt für eine Common Criteria (CC) Zertifizierung im Gesundheitswesen. Wir zeigen eine dicht an den best practices des Software-Engineerings angelehnte Kette von Werkzeugen und Workflows. Mit diesem Verfahren ermöglichen wir kollaboratives Arbeiten für Autorinnen und Autoren und produzieren Dokumente, die sich durch besonders gute Lesbarkeit und Benutzbarkeit auszeichnen. Die technische Verknüpfung von Strukturen des CC-Frameworks mit der Gliederung der Dokumentation sichert die Konsistenz in Inhalt und Form. Versionierungsverfahren sowohl für die Quelldateien als auch für die Ausgabedokumente stellen eine lückenlose Nachvollziehbarkeit des Dokumentationsprozesses sicher. Durch hoch automatisierte Arbeitsabläufe werden mehrmals täglich aktuelle Dokumentversionen erstellt und publiziert.

1 Einführung und Problemstellung

In diesem Beitrag stellen wir eine Dokumentationsplattform vor, die bei n-design GmbH im Rahmen des CC-Verfahrens des e-Health Konnektors *KoCoBox MED+* entwickelt wurde.

Schon im Rahmen des vorangegangenen Netzkonnektor-Verfahrens nach [BSI-PP47] haben wir festgestellt, dass die Dokumentation nicht nur fachlich eine Herausforderung darstellt, sondern auch technisch ihre Tücken hat. Diese Schwierigkeiten haben sich im laufenden Verfahren nach [BSI-PP98] weiter verschärft.

Die fachliche Herausforderung bestand darin, ein komplexes Softwareprodukt auf die Strukturen der CC und des Schutzprofils abzubilden. Die etwa 130 funktionalen Sicherheitsanforderungen (SFR) des Schutzprofils [BSI-PP98] werden von 162 Modulen in 23 Subsystemen umgesetzt. Es ist keine einfache Aufgabe, hier den Überblick zu bewahren und dafür zu sorgen, dass alle Zusammenhänge zwischen den CC-Elementen und der Implementierung konsistent dokumentiert werden.

Die technischen und organisatorischen Tücken liegen in den verwendeten Werkzeugen und den damit verbundenen Arbeitsabläufen. Im Vorgängerprojekt haben wir an einigen Stellen Schwierigkeiten gehabt, die in der Funktionsweise von Office-Produkten liegen:

- *Koordination der Zusammenarbeit an den Dokumenten*: Es entstand hoher Aufwand beim Zusammenführen der Arbeitsergebnisse der einzelnen Teams. Zwar erlauben Office-Produkte, dass mehrere Personen am selben Dokument arbeiten, doch dies setzt gemeinsames Arbeiten in der Cloud voraus. Das ist im vorliegenden Projekt nicht erlaubt.

- Die *Versionierung* der Dokumente war praktisch unmöglich. Das wurde besonders verschärft durch standortübergreifendes Arbeiten mit zwei separaten Unternehmen in getrennten Infrastrukturen und ohne Zugriff auf herstellerspezifische Cloud-Dienste.
- *Schwer einzuhaltende Konsistenz in Erscheinungsform und Inhalt*: Es ist schwer, restriktive Vorgaben in Bezug auf Layout und Terminologie umzusetzen. Formatvorlagen lassen sich leicht umgehen und feste Begriffe können von Autor zu Autor variiert werden, was hohen redaktionellen Aufwand verursacht.
- *Ungenügende Navigationsmöglichkeiten*: CC-Dokumente werden nicht linear vom Anfang zum Ende gelesen, sondern anhand von Verweisen und Sprüngen durchgearbeitet. Es ist nicht einfach, die dokumentinternen Verweise und Navigationsmöglichkeiten zu erstellen und zu erhalten, wenn sich dies nicht automatisieren lässt.
- *Mangelnde Akzeptanz bei den Autorinnen und Autoren*: Für Entwicklerinnen und Entwickler ist das Schreiben von CC-Dokumenten kein Vergnügen. Wenn dazu noch ungeliebte Werkzeuge und Arbeitsabläufe kommen, sinkt die Motivation dramatisch, was sich auf die Qualität der Ergebnisse auswirken kann.

Die folgenden Abschnitte zeigen, wie aus einem kleinen Experiment eine Dokumentationsplattform entstanden ist, an der bis zu sieben Autorinnen und Autoren in mittlerweile 15 Dokumenten ca. 3.000 Seiten Text produziert haben, die in einem kontinuierlichen Prozess integriert und innerhalb des Projekts ausgeliefert werden.

2 Datenmodell des Evaluierungsgegenstandes

Ein datenbankbasiertes Modell sorgt für die Abbildung des Evaluierungsgegenstandes (EVG) auf die Elemente der Common Criteria. In dieser Datenbank sind die Subsysteme, Module und Interfaces, aber auch die Sicherheitsanforderungen (SFR), Sicherheitsfunktionalitäten (SF) und Schnittstellen der SF (TSFI) modelliert und zueinander in Relation gesetzt. Abbildung 1 zeigt einen Ausschnitt aus dem Datenmodell.

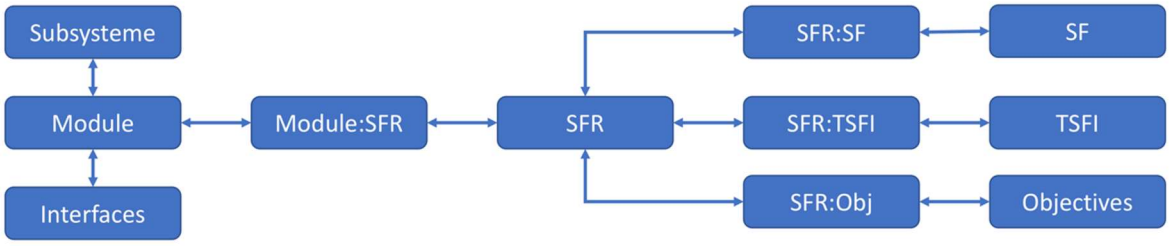

Abbildung 1: Datenmodell der CC-Dokumentation (Ausschnitt)

Das Modellieren des EVG in einer relationalen Datenbank hat uns geholfen, Sicherheit über diese Relationen und den Zuschnitt des EVG zu gewinnen. Zwar haben wir vorher schon Zuordnungen von SFR auf Subsysteme gemacht und dadurch einen Überblick gehabt, doch eine relationale Datenbank bietet den Vorteil, dass komplexere Relationen damit sichtbar werden. Constraints auf der Definition der Datenbanktabellen unterstützen, dass das Modell konsistent bleibt.

Darüber hinaus lassen sich mit der Datenbank auch Bezüge zwischen CC-Elementen herstellen, die sonst nur schwer berechenbar wären: Die Frage, welches Modul welche TSFI repräsentiert, ist mit einem einfachen SQL Join beantwortet. Diese Modellierung hilft nicht nur den Autorinnen und Autoren: Die Datenbank wird – gemeinsam mit den Dokumenten – an den Evaluator

ausgeliefert, der selbständig Abfragen definieren kann, um mehr über die Zusammenhänge im EVG zu erfahren. Natürlich setzt dies voraus, dass das Schutzprofil in Bezug auf die Sicherheitsanforderungen sorgfältig modelliert ist, da diese als zentrales Element des Datenmodells einen allgemeinen Referenzpunkt für die Relationen bilden.

Die Datenbank dient bei der Dokumentation verschiedenen Zwecken: (1) Sichern der terminologischen Konsistenz, (2) Formalisieren der Relationen und (3) Generieren von Tabellen und Querverweisen. Auf alle Punkte wird in späteren Abschnitten eingegangen.

Technisch wird die Datenbank über SQLite abgebildet. Das Datenmodell liegt in Form von CSV-Dateien vor. Diese Dateien lassen sich sehr gut von Hand pflegen und passen – da es sich um reine Textdateien handelt – hervorragend in unser Versionierungsverfahren.

3 Dokumentation mit LaTeX

In Erwartung des komplexen Verfahrens nach [BSI-PP98] entschieden wir uns zu einer Abkehr von klassischen Office-Produkten für die Dokumentation. Wir haben ein alternatives Verfahren entwickelt, das die in der Einführung beschriebenen Problemfelder ausräumt. Grundlegender Baustein ist das Textsatzsystem LaTeX (gesprochen: „Latech“), das einige unserer Entwicklerinnen und Entwickler in der Vergangenheit sehr gerne und erfolgreich eingesetzt haben.

LaTeX (oder korrekt formuliert: TeX; allerdings spielt die Unterscheidung hier keine Rolle) ist ein fast vierzig Jahre altes Textsatzsystem, das sich besonders in den akademischen Bereichen der Mathematik, der Informatik, der Naturwissenschaften und der Linguistik großer Beliebtheit erfreut. Neben hoher typographischer Qualität der produzierten Dokumente und einer nahezu unerschöpflichen Vielfalt an fachspezifischen Zusatzpaketen wird LaTeX wegen seines Workflows geschätzt, der sich grundlegend von den ansonsten üblichen Textverarbeitungssystemen unterscheidet: Die Autorin oder der Autor schreibt den Text in einem beliebigen Editor und reichert ihn mit Steuerbefehlen, den Makros, an, um damit Struktur und Formatierungen umzusetzen. Der LaTeX Prozessor, eine Art Compiler, übersetzt diesen Quellcode in ein Zielformat – heute üblicherweise PDF. Dieser Ablauf ähnelt am ehesten dem, was in der Softwareentwicklung praktiziert wird; mit dem Unterschied, dass als Artefakt kein ausführbarer Code, sondern ein Dokument entsteht.

Trotz seines Alters existiert um LaTeX herum ein überaus vitales Ökosystem mit einer sehr aktiven Community. Ein Feld der Weiterentwicklung, das sich als sehr wertvoll für das hier beschriebene Projekt herausgestellt hat, ist LuaTeX. Dieser Prozessor hat einen Interpreter für die Programmiersprache Lua eingebaut. Über Makros kann eingebetteter Lua Code ausgeführt werden. Dieser Lua-Interpreter verfügt über einen Ausgabestrom, der direkt vom LaTeX Prozessor verarbeitet wird. Darüber lassen sich Dokumentteile programmatisch erstellen. Über den Zugriff auf Bibliotheken des Betriebssystems lassen sich auch noch andere Funktionen einbinden. Wir binden darüber die Datenbank ein, die das Modell unseres Evaluierungsgegenstandes enthält.

Das zentrale Element des Verfahrens ist der LuaTeX Prozess. Er erhält für jedes zu erstellende PDF-Dokument einen Satz an LaTeX-Quellen, die Texte, Bilder und Makros umfassen. Um zwischen den Dokumenten Einheitlichkeit zu wahren, werden für jedes Dokument weitgehend identische Makros verwendet: Dieselbe Datei wird von mehreren Dokumenten eingebunden. Auch bestimmte Textteile, die in mehreren Dokumenten verwendet werden, liegen nur einmal vor und werden mehrfach importiert.

Die LaTeX-Quellen umfassen ebenfalls einen Satz von Lua-Programmen, die ebenfalls vom LuaTeX-Prozessor eingelesen werden. Dieser Code startet die SQLite-Datenbank. Diese Datenbank wird flüchtig im Speicher gehalten. Beim Start eines LaTeX-Prozesses wird die Datenbank instanziiert und mit dem Datenmodell befüllt, das in CSV-Dateien vorliegt. Dieses Einlesen und Parsen der Dateien wird ebenfalls von Lua-Programmen umgesetzt.

Damit stehen alle Komponenten bereit, um ein Ausgabedokument zu erzeugen. Wenn LaTeX mit dem Textsatz fertig ist und das PDF-Dokument erstellt wurde, wird die In-Memory DB verworfen. Da sie ausschließlich zum Lesen verwendet wurde, entsteht kein Datenverlust.

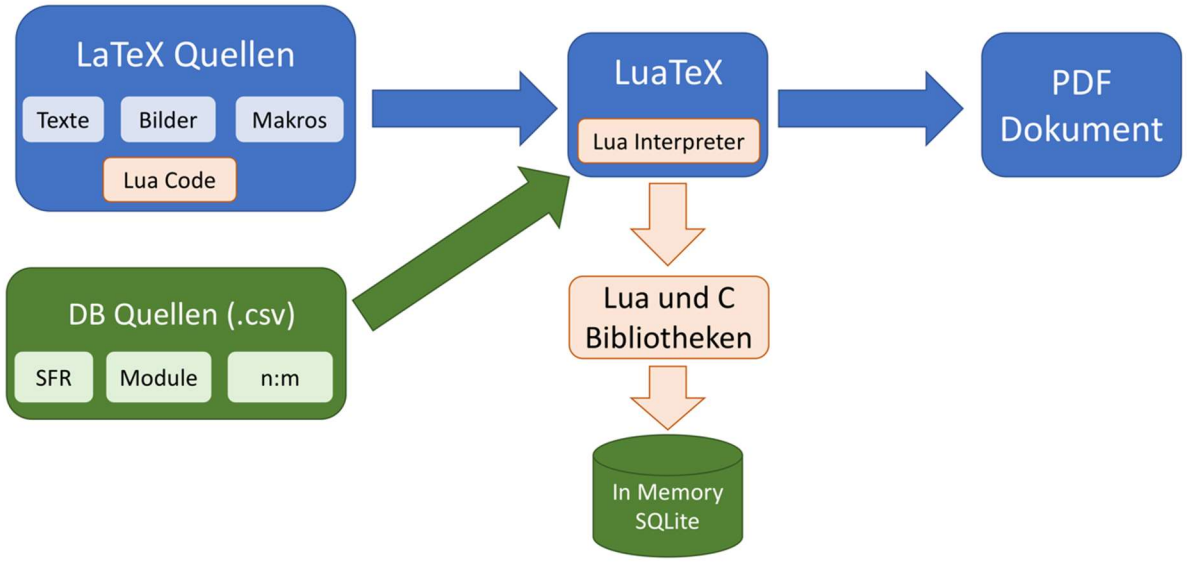

Abbildung 2: Prozess zur Dokumenterzeugung

4 Makros für Formatierung und Struktur

Die terminologische und strukturelle Konsistenz einer technischen Dokumentation sicherzustellen ist eine besondere Herausforderung; umso mehr, wenn verschiedene Personen an der Erstellung beteiligt sind. Unterschiedliche Sichtweisen oder einfach nur verschiedene Geschmäcker der beteiligten Autorinnen und Autoren können dazu führen, dass Terminologie unterschiedlich benannt wird oder dass uneinheitliche Formatierungen den Lesefluss stören. Uns ist es wichtig, dass die Leserinnen und Leser des Dokuments schon anhand der Typographie erkennen, ob ein bestimmter Begriff im Text gerade z.B. das Schlüsselwort aus der Spezifikation oder eine Instanzvariable im Code bezeichnet. Organisatorische Vorgaben helfen hier nur eingeschränkt weiter.

In LaTeX werden Formatierungen und Strukturelemente wie Überschriften in Form von Makros im Quelltext markiert. So setzt zum Beispiel das Makro

```
\textit{Dieser Text wird kursiv}
```

den in geschweiften Klammern genannten Text kursiv. Wir haben uns beispielsweise dafür entschieden, Schlüsselwörter aus der Spezifikation kursiv zu setzen. Doch statt jedes Schlüsselwort mit `\textit{}` zu umgeben, gibt es ein selbstdefiniertes Makro namens `\keyword{}`, das

intern wiederum `\textit{}` aufruft. Diese einfache Indirektion hilft durch eine zentrale Definition von `\keyword{}`, das Aussehen sämtlicher Schlüsselwörter mit einem Schlag ändern zu können. Aber nicht nur Formatierungen, sondern auch Begriffe oder immer wiederkehrende Standardtexte werden über Makros gesetzt: Das Makro `\kocobox{}` wird durch den Namen des Evaluierungsgegenstandes „KoCoBox MED+“ ersetzt.

Nun haben wir nicht nur Makros für Schlüsselwörter definiert, sondern für sehr viele Elemente, die man im Rahmen einer CC-Evaluierung braucht: beispielsweise für Sicherheitsanforderungen, für Codeschnipsel und für Links zu Subsystemen, Modulen und Schnittstellen. Durch diese Makros müssen die Autorinnen und Autoren kaum etwas über die allgemeinen LaTeX Formatierungsbefehle wissen, sondern sich lediglich im Kanon der *domänenspezifischen* Makros auskennen. Das verringert einerseits die Hürden für die Nutzung der Plattform, sichert andererseits die Konsistenz in Form und Inhalt. Daher sprechen wir von *semantischem Markup*, also einer Textauszeichnung, die sich an der Bedeutung eines Elements im Kontext des Dokuments und nicht an den visuellen Aspekten orientiert.

Für die Entwicklung der domänenspezifischen Makros ist wiederum das EVG-Modell in der Datenbank höchst wertvoll. Beim Schreiben muss beispielsweise niemals der „echte“ Name eines Subsystems, Moduls oder Interfaces angegeben werden. Man verwendet stattdessen ein Kürzel, das mit Hilfe eines Makros und eines Datenbankzugriffs in den echten Namen expandiert wird. Diese Kürzel sind meist einprägsamer als die ausgeschriebenen Namen, außerdem sind sie häufig direkt an die Benennungen im Code des EVG angelehnt, was wiederum den Entwicklerinnen und Entwicklern entgegenkommt.

Im Beispiel in Abbildung 3 wird das Makro `\tds{}` mit dem Parameter `mod.aas.core` aufgerufen. Ein Lua-Programm löst den Parameter mit zwei Zugriffen auf die Datenbank auf und ersetzt ihn durch „AccessAuthorizationService::Core“, was in der Nomenklatur des vorliegenden Verfahrens das Modul „Core“ des Subsystems „AccessAuthorizationService“ bezeichnet.

Der Vorteil dieses Vorgehens ist, dass die Namen immer fehlerfrei geschrieben sind und dass Autorinnen und Autoren nicht beliebig neue Modulnamen erfinden können, weil sie möglicherweise ein CC-Modul und ein Softwareartefakt verwechseln. Sollte es zu einer Abweichung kommen, wird ein in der Datenbank nicht definiertes Kürzel im PDF-Dokument auffällig als Fehler gekennzeichnet.

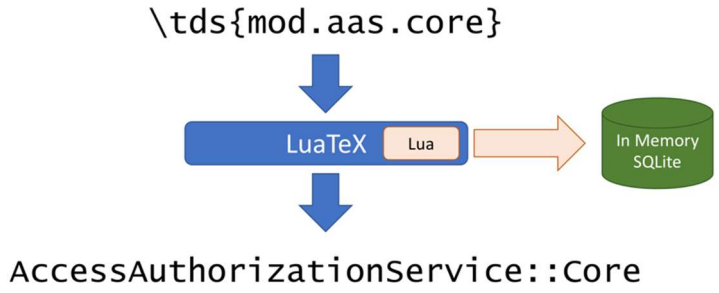

Abbildung 3: Auflösen eines TDS Modulnamens mit Makro und Datenbank

5 Generierte Textelemente: Tabellen und Links

Über die reine Textkonsistenz hinaus lassen sich mit diesen Mechanismen auch Tabellen generieren, die ohne Tool-Unterstützung nur schwer konsistent zu halten wären: Mit einer selbstentwickelten Lua-Funktion wird für jedes Modul in der Design Spezifikation eine Tabelle produziert, in der die *enforcing* und *supporting* SFR für dieses Modul gelistet sind. Der Ausschnitt in Abbildung 4 zeigt diese Tabelle beispielhaft für das Modul „AccessAuthorizationService::Core“. Der gesamte Text in diesem Ausschnitt ist programmatisch generiert worden. Im Anhang des TDS gibt es eine entsprechende Tabelle, in der für jedes SFR die zugehörigen Module aufgeführt sind. Solche Tabellen manuell zu pflegen und konsistent zu halten, ist aufwändig und fehlerträchtig.

3.14.1. Modul AccessAuthorizationService::Core

Für den Gesamtkonnektor: SFR-enforcing
Für den Netzkonnektor: non-TSF

Das Modul erfüllt die Anforderungen, die durch die SFR in Tabelle 3.348 an den EVG gestellt werden.

Enforcing SFR		
FDP_ACC.1/AK.Infomod	FDP_ACF.1/AK.KD	FMT_MTD.1/AK.eHKT_Abf
FDP_ACC.1/AK.KD	FMT_MSA.1/AK.Infomod	FMT_SMF.1/AK
FDP_ACF.1/AK.Infomod	FMT_MSA.3/AK.Infomod	FPT_FLS.1/AK
Supporting SFR		
FDP_ACC.1/AK.Sgen	FDP_ACF.1/AK.Sgen	
FDP_ACC.1/AK.SigPr	FDP_ACF.1/AK.SigPr	

Tabelle 3.348.: SFR des Moduls AccessAuthorizationService::Core

Abbildung 4: Generierte Tabellen im TDS

Ein weiterer wichtiger Punkt ist die Navigation innerhalb der Dokumente. Damit ein Dokument nicht nur mit Hilfe des Inhaltsverzeichnisses navigiert werden kann, sind Querverweise wichtig. Querverweise werden in PDF-Dokumenten als Hyperlinks umgesetzt. Auf der Basis des Datenmodells und der Makros, die auf die Datenbank zugreifen, können automatisch Links generiert werden. Mit deren Hilfe kann der Evaluator *nicht-linear* durch das Dokument navigieren und von Modul zu Modul, von Interface zu Interface springen und damit dem Ablauf einer Sicherheitsfunktionalität folgen. Die Konsistenz des Datenmodells ist dabei von zentraler Bedeutung. Sie stellt sicher, dass es in jedem Dokument geeignete Anker für die Hyperlinks gibt. Die Querverweise repräsentieren das Datenmodell in den Dokumenten.

6 Arbeit im Team: Koordination, Versionierung und Release-Management

Versionsverwaltung ist ein unabdingbares Element der Softwareentwicklung. Nur mit einer etablierten und akzeptierten Versionsverwaltung im Hintergrund sind Teams in der Lage, gemeinsam an Softwareprojekten zu arbeiten. Im vorliegenden Projekt haben wir uns für Git als

System zur Versionsverwaltung entschieden. Die Wahl fiel nicht schwer, denn Git ist seit Jahren der de facto Standard mit umfangreicher Tool-Unterstützung und breitem Know-How in der Community.

Git arbeitet, wie viele Versionsierungssysteme, auf Basis von Differenzen von Textdateien („ASCII-Dateien“). Dateiformate wie die von Word oder anderen Office-Produkten lassen sich schwer mit Git versionieren, da die inhärente Struktur komplexer ist und Binärelemente enthalten kann. Hierauf lassen sich schwerer Differenzen bilden, sodass sich gemeinsames Arbeiten am selben Dokument nicht einfach abbilden lässt. LaTeX-Quelldateien hingegen sind reine Textdateien, die sich hervorragend mit Git versionieren lassen.

Weiterhin kann ein LaTeX Dokument aus vielen kleinen Einzeldateien bestehen, die über Input/Include-Anweisungen in Rahmendateien eingebunden werden. Wir konnten die Dokumente so herunterbrechen, dass üblicherweise pro Kapitel eine neue Datei entsteht. Im Fall der Design Spezifikation wurden die Dateigrenzen an Modulgrenzen gezogen: Jedes Modul hat seine eigene Datei. Diese Modularisierung macht die Arbeit im Team deutlich effizienter und reibungsärmer. Je mehr einzelne Dateien beteiligt sind, desto weniger Konflikte muss das Versionierungssystem auflösen. Darüber hinaus erleichtern kleine Einheiten auch die Wiederverwendung von Texten in mehreren Dokumenten.

Neben der Versionierung der Dokument-Quellen ist auch die Versionierung der Artefakte (also der fertigen Dokumente) ein Thema. Es soll jederzeit nachvollziehbar sein, welche Version der Evaluator erhalten hat. Diese Version soll reproduzierbar sein und es muss – besonders wichtig – ein Differenzdokument zur vorherigen Version erzeugt werden.

Eine nachvollziehbare Versionierung bedingt eine präzise Benennung der Versionen. Beim Versionierungsschema orientieren wir uns an dem, was im Java-Umfeld seit Jahren durch das Build-Management Werkzeug Maven etabliert ist: Zwischenstände enthalten in ihrer Versionsnummer das Suffix „-SNAPSHOT“. Ein Dokument der Version „1.6-SNAPSHOT“ ist zu lesen als: „Ein Zwischenstand vor der Version 1.6“. Wenn ein Release erzeugt wird, entfernt man das Suffix „-SNAPSHOT“. Im Versionsverwaltungssystem wird ein Tag „v1.6“ erzeugt. Damit steht der neue Versionsstand fest und ist reproduzierbar. Schließlich wird die Versionsnummer inkrementiert („1.7“) und das Suffix wieder angehängt („1.7-SNAPSHOT“). Damit befindet man sich „auf dem Weg zum nächsten Release“. Was hier gewöhnungsbedürftig klingt, ist für Entwicklerinnen und Entwickler Teil der täglichen Arbeit und deckt sich zu 100% mit dem, was auch in der Softwareentwicklung praktiziert wird.

Die vom Evaluator geforderten Differenzdokumente erstellen wir mit Adobe Acrobat, mit dem sich selbst umfangreiche PDF-Dokumente komfortabel vergleichen lassen. Mit Hilfe der von Acrobat eingebrachten Annotationen kann der Evaluator schnell die Änderungen seit der letzten Version nachvollziehen. Adobe Acrobat ist das einzige Werkzeug in der Kette, das nicht unter einer OpenSource-Lizenz verteilt wird.

Das gesamte Releaseverfahren ist ein kritischer Prozess, bei dem viel kaputtgehen kann. Releases zurück zu rollen und neu zu erstellen ist fehlerträchtig und aufwändig. Deswegen war es für uns wichtig, diesen Prozess nicht nur gut zu dokumentieren, sondern weitgehend zu automatisieren. Dies haben wir in Form eines Shell-Skripts implementiert, das aus der Datenbank die aktuellen Versionsnummern ausliest, selbständig die notwendigen Anpassungen vornimmt und im Git-Repository persistiert. So wird Anzahl der manuellen Schritte reduziert.

7 Deployment und Continuous Delivery

Nicht zu vernachlässigen ist der Aufwand, die Werkzeuge für ein Verfahren an einem Arbeitsplatz zu installieren und neue Autorinnen und Autoren in den Dokumentationsprozess einzubeziehen. Die gesamte Dokumentationsplattform steht in Form einer virtuellen Maschine zur Verfügung, die sich innerhalb von 20 Minuten auf einem Entwicklerarbeitsplatz klonen lässt. So können alle beteiligten Personen in identischen Umgebungen arbeiten. Das reduziert die Fehleranfälligkeit, weil bei Schwierigkeiten nicht in individuellen Installationen der Software gesucht werden muss.

Je mehr Personen und Organisationen an einem Projekt beteiligt sind, desto schwieriger wird es, den Überblick über den „aktuellen Stand“ der Dokumente zu behalten. Für die Versionierung der Quellen und der Releases sind Verfahren etabliert. Doch es bleiben zwei Fragen: In welcher Form werden die fertigen Dokumente zur Verfügung gestellt? Und wie kommen Interessierte an die Zwischenversionen, wenn sie nicht die Entwicklungsumgebung installiert haben?

Wir haben zu diesem Zweck eine Website im Intranet eingerichtet, über die der aktuelle Snapshot der Dokumente abrufbar ist. So haben alle am Projekt Beteiligten (und alle, die nicht direkt mit dem Projekt zu tun haben) Zugriff auf den aktuellen Stand der Dokumente. Zusätzlich publizieren wir auf dieser Website alle bisher erstellten Releases sowie die Differenzdokumente. Damit wird die komplette Projekthistorie transparent gemacht.

Die Bestückung dieser Website verläuft automatisiert ohne Interaktion durch einen Benutzer. Auch für diesen Zweck nutzen wir ein Verfahren, das sich in der Softwareentwicklung etabliert hat: Continuous Delivery. Das bezeichnet ein Verfahren, bei dem mit jeder Codezulieferung ein Build gestartet und ein auslieferbares Ergebnis erzeugt wird, das in Test- oder sogar Produktionsumgebungen zum Einsatz gebracht werden kann. Ziel ist die Abkehr von umfangreichen Releases, die in ihren Abhängigkeiten und Auswirkungen kaum handhabbar sind. In unserem Kontext bedeutet Continuous Delivery, dass mit jedem zugelieferten Textbeitrag automatisch ein neuer Snapshot aller Dokumente produziert wird, der unmittelbar auf der Intranet-Site publiziert wird.

Technisch setzen wir diesen Prozess mit Hilfe unseres Git Servers GitLab und des Continuous Integration Servers Jenkins um. Bestimmte Änderungen an den Quellen in Git lösen einen Job auf dem Jenkins Server aus. Dieser Job lädt den neuen Stand der Dokumentquellen aus dem Versionierungssystem, startet die LaTeX Prozesse und erstellt so die PDF-Dokumente. Wenn alle Dokumente fehlerfrei übersetzt werden konnten, werden sie auf die Intranet-Website kopiert und die Links dort aktualisiert. So vergehen zwischen der Annahme einer Änderung durch den leitenden Redakteur bis zur Auslieferung des neuen Dokumentenstandes im Intranet nur wenige Minuten.

Abbildung 5: Website für CC-Dokumente im Intranet

8 Abschluss und Ausblick

Das beschriebene Verfahren ist als Experiment gestartet, um zu prüfen, ob sich LaTeX als Dokumentationswerkzeug in CC-Verfahren verwenden lässt. Nach knapp zwei Jahren ist aus dem Experiment eine Dokumentationsplattform geworden, die es uns ermöglicht, komplexe Dokumente zu erstellen und diese in sich und in Relation zueinander konsistent zu halten.

Bei einem solchen Verfahren stellt sich unmittelbar die Frage nach der Wirtschaftlichkeit. Da die Plattform entlang der Dokumente konzipiert und entwickelt wurde, fällt es schwer, im

Nachhinein die Kosten zu berechnen. Zwar entstand initialer Aufwand und auch bei der Fortentwicklung sind immer wieder Justierungen notwendig. Doch die Mechanismen sparen sehr viel manuellen, zeitraubenden und fehlerträchtigen Aufwand ein. Besonders im Bereich der Kollaboration und im Management der Dokumente entfallen viele Arbeiten, die in der Vergangenheit viel Stress und Unruhe ins Projekt gebracht haben. Dass diese Arbeiten weggefallen sind, lässt die Stimmung im Projekt steigen.

Weiterhin kommt – bis auf Adobe Acrobat – ausschließlich Open Source Software zum Einsatz, die kostenfrei verwendet werden kann.

Das Verfahren skaliert sehr gut. Neue Dokumente lassen sich einfach hinzufügen. Zurzeit werden 15 Dokumente mit insgesamt ca. 3.000 Seiten darüber verfasst. Auch lange Dokumente sind kein Problem: Die Übersetzungszeiten sind linear und der Speicherverbrauch ist überschaubar. Die Akzeptanz des Verfahrens ist bei Autorinnen und Autoren sehr groß. Das liegt an den hier umgesetzten best practices der Softwareentwicklung, mit denen sich die Autorinnen und Autoren hervorragend auskennen. Nicht zuletzt die Continous Delivery stärkt bei uns im Haus das Vertrauen in das Verfahren: Wenn jederzeit der offizielle, aktuelle Zwischenstand ersichtlich ist, entfällt viel Suchen und Nachfragen, wo denn der neueste Stand der Dokumente gerade liegt. Es gibt eine gemeinsame Arbeitsgrundlage.

Auch der Evaluator weiß die Dokumente zu schätzen: Die eindeutigen Versionen, die Differenzdokumente und der streng definierte Workflow beim Erstellen der Dokumente erhöhen das Vertrauen in das Verfahren und somit in das evaluierte Produkt. Die Navigierbarkeit über Hyperlinks machen die Arbeit mit den Dokumenten sehr effizient.

Die Plattform wurde für eine spezifische CC-Evaluation entwickelt. Es ist mit geringem Aufwand möglich, die verfahrensspezifischen Elemente zu entfernen und die Plattform in einem anderen Verfahren zu verwenden. Darüber hinaus lassen sich die grundsätzlichen Mechanismen, die zum Einsatz kommen, auch auf andere technische Dokumentationen anwenden.

Literatur

[BSI-PP47] Bundesamt für Sicherheit in der Informationstechnik. Schutzprofil 1: Anforderungen an den Netzkonnektor. BSI-CC-PP-0047. Common Criteria Schutzprofil (Protection Profile). Version 3.2.2. Bundesamt für Sicherheit in der Informationstechnik (BSI), 11. Apr. 2016.

[BSI-PP98] Bundesamt für Sicherheit in der Informationstechnik. Schutzprofil 2: Anforderungen an den Konnektor. BSI-CC-PP-0098. Common Criteria Schutzprofil (Protection Profile). Version 1.1. Bundesamt für Sicherheit in der Informationstechnik (BSI), 11. Dez. 2017.

Alexander Krumeich

Studium der Computerlinguistik/Künstliche Intelligenz an der Universität Osnabrück.

Softwareentwickler bei n-design GmbH, Köln. Dort mitverantwortlich für die Sicherheitszertifizierung der *KoCoBox Med+*, einem Konnektor für die Telematikinfrastruktur im Gesundheitswesen. Konzeption und Entwicklung der beschriebenen Dokumentationsplattform.

Kontakt

Alexander Krumeich

n-design GmbH
Alpenerstr. 16
50825 Köln
Tel. +49 221 22289616
E-Mail: alexander.krumeich@n-design.de