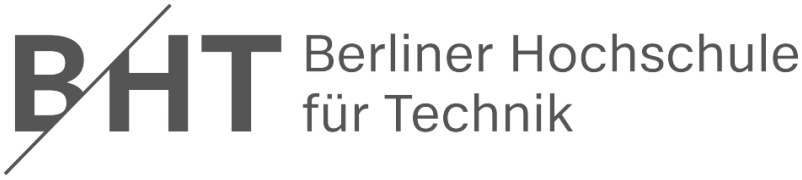

Thema:

Analyse und Bewertung von Software-Code-Qualität von Erweiterungen in Content-Management-Systemen anhand von TYPO3

Bachelor-Arbeit

zur Erlangung des Grades Bachelor of Science in Medieninformatik
des Fachbereichs Informatik und Medien an der
Berliner Hochschule für Technik

Prüfungskommission:
Gutachter*in 1: Prof. Dr. Dragan Macos
Betreuer*in extern: Sebastian Kreideweiß

Renate Ubartas
Bachelor of Science
Studiengang Medieninformatik Online

S82371@bht-berlin.de
Beginn: 01.06.2023
Ende: 01.09.2023

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen anderer Autoren beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und noch nicht veröffentlicht.

Berlin, 31.08.2023
Renate Ubartas

Inhaltsverzeichnis

Tabellenverzeichnis

Glossar

Begriff	**Definition / Erklärung**
Backend & Frontend	Backend und Frontend beschreiben die Trennung des Programmcodes einer Software in Funktionsbereiche, wobei das Backend dem funktionalen Teil ohne Benutzeroberfläche entspricht und Frontend mit der Entwicklung der Benutzeroberfläche vergleichbar ist.
Clean Code	Hat sich als Begriff etabliert, um gut lesbaren und verständlichen Code zu beschreiben. Kann durch verschiedene Merkmale und Prinzipien beschrieben werden.
CMS	Bezeichnet ein Content-Management-System. CMS werden für die Erstellung und Verwaltung von Content genutzt. Im allgemeinen Sprachgebrauch ist CMS häufig als System für das Web-Content-Management gemeint.
Code Smell	Wird als Begriff für Codeabschnitte verwendet, die eine mangelhafte Qualität oder andere negativ auffallende Eigenschaften aufweisen.
Core	Die Core-Entwicklung bezieht sich auf die Weiterentwicklung des zentralen Teils oder Kerns des TYPO3-Systems.
Extension	Extensions sind im Kontext von TYPO3 Erweiterungen des CMS. Diese Erweiterungen können als eigene kleine Programme angesehen werden, die die Funktionen von TYPO3 erweitern.
Legacy-Code	Legacy bedeutet im Deutschen „Vermächtnis“ oder „Altlast“ (Krypczyk & Bochkor, 2022). Der Begriff beschreibt im Kontext des Software-Programmcodes einen Code, der einem veralteten Stand entspricht.
MVC	Das Model View Controller Designmuster (MVC) stellt ein Architekturkonzept dar, das die klare Trennung der einzelnen Software-Komponenten in Funktionsbereiche vorschreibt.
Softwareentwicklungszyklus	Beschreibt den Entwicklungsprozess von Software. Der Zyklus kann eingeteilt werden in die Bereiche Anforderungsanalyse, Design, Planung, Umsetzung, Testing, Integration und Wartung.
Statische Analyse Tools	Programme, mit denen Software-Code auf festgelegte Regeln geprüft werden kann. Statische Prüfung bedeutet, dass die Analyse des Codes durchgeführt wird, ohne diesen auszuführen.
Technische Schuld	Bezeichnet das Nichtdurchführen oder das Aufschieben bestimmter Handlungen wie das Erstellen von Dokumentationen oder Tests.

Abstract

Diese Bachelorarbeit zielt darauf ab Software-Code-Qualität in TYPO3-Extensions[1] durch relevante Merkmale und Kriterien zu beschreiben. Es soll herauskristallisiert werden welche Aspekte entscheidend sind, um die Code-Qualität zu bewerten und wie diese wesentlichen Punkte von Entwicklern und Entwicklerinnen durch die Unterstützung von statischen Analyse-Tools[1] sichergestellt werden können.

Mit Hilfe von Interviews mit Experten und Expertinnen wurden Merkmale und Kriterien erfasst, die bei der Bewertung von Software-Code von TYPO3-Extensions eine entscheidende Rolle spielen. Es wurde eine Inhaltsanalyse der Interviews durchgeführt, um dadurch wesentliche Aspekte der Code-Qualität zu erfassen und zu beschreiben. Die praktischen Empfehlungen, die aus dieser Forschung hervorgehen, bieten Entwicklungsteams wertvolle Orientierungspunkte, um effektivere Entwicklungsprozesse zu gestalten und die Fehleranfälligkeit zu minimieren. Damit kann nicht nur die Entwicklungsqualität gesteigert, sondern auch die Instandhaltung von TYPO3-Extensions optimiert werden. Darüber hinaus eröffnen die gewonnenen Erkenntnisse Unternehmen die Möglichkeit, eine nachhaltige Entwicklung zu fördern und gleichzeitig eine produktive Arbeitsatmosphäre für Entwickler und Entwicklerinnen zu schaffen, insbesondere in einer Zeit, in der die Komplexität der Softwareentwicklung stetig zunimmt.

Die Ergebnisse dieser Forschungsarbeit verdeutlichen zudem, dass gerade im Kontext von TYPO3-Extensions eine beträchtliche Menge an Legacy-Code[1] existiert, deren Präsenz Auswirkungen auf das Zeitmanagement und die Motivation von Entwicklern und Entwicklerinnen haben kann. Insbesondere für Anfänger und Anfängerinnen gestaltet sich der Einstieg in die Entwicklung dadurch mitunter herausfordernd. Dieses Szenario wird durch den Open-Source-Charakter noch komplexer, da eine beträchtliche Menge an Code mit unzureichender Qualität öffentlich verfügbar ist.

Die Untersuchung der Einsatzmöglichkeiten von statischen Analyse-Tools kann somit einen wertvollen Beitrag dazu leisten den Einstieg in die TYPO3-Programmierung zu erleichtern und gleichzeitig dem Problem entgegenzuwirken, dass zusätzlicher Code von minderer Qualität erzeugt wird. Indem diese Tools dabei helfen, bestehenden Code zu überprüfen und auf potenzielle Probleme hinzuweisen, können sie einen bedeutsamen Beitrag zur Förderung eines reibungslosen Entwicklungsprozesses und zur Vermeidung von Qualitätsverlusten leisten.

[1] Definition im Glossar

1. Einleitung

Die Bedeutung der Entwicklung von Webanwendungen wächst kontinuierlich, da diese sich für zahlreiche neue Anwendungsbereiche eignen. Moderne Webanwendungen sind durch technologische Fortschritte in vielerlei Hinsicht auf dem gleichen Niveau wie andere Software-Systeme (Krypczyk & Bochkor, 2022). Eine Schlüsselrolle spielen dabei Content-Management-Systeme (CMS)[2]. Der Begriff CMS bezieht sich fast ausschließlich auf Web-CMS, da heutzutage sämtliche professionelle Webpräsenzen auf CMS basieren (Bühler et al., 2019). Ein besonders in Deutschland bekanntes und gängiges CMS ist TYPO3 (CMScensus - CMScensus, o. D.), welches durch die Nutzung von Extensions individualisierbar und im Funktionsumfang erweiterbar ist.

Die Code-Qualität dieser TYPO3-Extensions kann stark variieren, da diese von unterschiedlichen Programmierern oder Programmiererinnen entwickelt werden, die möglicherweise unterschiedliche Erfahrungen oder Programmierfähigkeiten besitzen. Dies kann beispielsweise zu schwacher Leistung, mangelnder Sicherheit, erschwerter Wartbar- und Erweiterbarkeit der Extensions oder anderen Problemen führen. Zusätzlich kann es vorkommen, dass an der Entwicklung einer Software mehrere Teams und viele Programmierer und Programmiererinnen beteiligt sind. Dies ist einer von vielen Aspekten, warum die Entwicklung von Software-Code in einer hohen Qualität wichtig ist (Ljung & Gonzalez-Huerta, 2022).

Doch wie sieht ein guter Code aus? Die Definition von guter Code-Qualität ist nicht eindeutig, da in die Beurteilung äußere Faktoren einfließen können, wie z. B. welche Personengruppe die Qualität beurteilt. Die Beurteilung von Code-Qualität ist nur subjektiv möglich, sodass eine objektive Vergleichbarkeit schwierig ist. Entwickler und Entwicklerinnen können unterschiedliche Auffassungen darüber haben, welche Merkmale besonders wichtig sind für die Bewertung der Code-Qualität.

Die Anwendung etablierter Praktiken und Normen kann bereits zu einem hohen Maß an Code-Qualität führen, jedoch bleibt die Programmierung von Code ein subjektiver Prozess, der Interpretationsspielraum zulässt. Daher ist es wichtig, gute Code-Qualität möglichst genau zu beschreiben, um von Anfang an Code in einer hohen Qualität zu produzieren und um bestehenden Code zu optimieren.

Mit den Ergebnissen dieser Arbeit sollen Empfehlungen für die Entwicklung von TYPO3-Extensions gegeben werden, die zu einer höheren Code-Qualität und somit zu einer nachhaltigen Entwicklung beitragen können.

[2] Definition im Glossar

1.1. Hintergrund und Kontext

Kirk et al. (2020) führten eine Studie durch, um den Umfang der Behandlung von Code-Qualität in Anfänger-Programmierkursen an Universitäten zu untersuchen. Dabei analysierten sie Universitäten weltweit, die ihre Programme auf Grundlage der ACM-Richtlinien entwickeln und Kurse, sowie Lernergebnisse entsprechend dieser Richtlinien gestalten. ACM ist die „Association for Computing Machinery“, eine große und weltweit aktive Gesellschaft für Bildung und Wissenschaft im Informatik-Bereich (*Association for Computing Machinery*, o. D.). Ihre Stichprobenerhebung ergab, dass es keine Gewährleistung für die Vermittlung von Konzepten im Zusammenhang mit Code-Qualität und ihrer Bedeutung gibt. Nur etwa 30 % der untersuchten Kurse behandelte codequalitätsbezogenen Themen. Da die Lehrpläne von den jeweiligen Lehrkräften individuell erstellt werden, besteht eine hohe Variabilität in Bezug auf die Behandlung von Code-Qualität in den Kursen. Die Studie von Kirk et al. (2020) zeigt, dass das Thema der Code-Qualität eine untergeordnete Rolle in Lehrplänen für Studienrichtungen, in denen Programmieren gelehrt wird, spielen kann. Kirk et al. (2020) stellen fest, dass der Zusammenhang zwischen einer hohen Code-Qualität und sinkenden Kosten eines Softwareprojektes allgemein bekannt ist, die Auseinandersetzung damit während des Lernprozesses der Programmierung jedoch nicht zwangsläufig gewährleistet ist.

Eigene Erfahrungen aus dem Studium, unterstützt durch die Studie von Kirk et al. (2020), führten zu dem Thema der Software-Code-Qualität für diese Bachelorarbeit.

Im Bereich der Softwarequalität und speziell der Code-Qualität werden in der Literatur allgemein gültige Aussagen getroffen. Viele Ansätze und Konzepte zur Verbesserung der Code-Qualität sind unabhängig von einer spezifischen Programmiersprache. Auch viele Forschungen und Literaturen beziehen sich nicht auf eine bestimmte Programmiersprache. Die Methoden und Analysen sind oft allgemein auf objektorientierte Programmiersprachen anwendbar. Diese allgemeinen Prinzipien dienen auch in dieser Forschungsarbeit als Grundlage, um dann später auf die spezifischen Anforderungen der Entwicklung von TYPO3-Extensions mit dem Schwerpunkt der PHP-Programmierung im Backend[3] einzugehen. Obwohl auch im Frontend[3] die Code-Qualität von großer Bedeutung ist, konzentriert sich diese Bachelorarbeit primär auf den Backend-Bereich der Extension-Entwicklung, um den umfangreichen Bereich der Software-Code-Qualität einzugrenzen.

1.2. Relevanz und Zielsetzung

Qualitätskosten, die für die Aufrechthaltung oder Verbesserung der Softwarequalität dienen, teilen (Krypczyk & Bochkor, 2022) in die zwei Hauptkategorien Fehlerverhütungs- und Fehlerkosten ein. Laut Krypczyk und Bochkor (2022) betragen diese schätzungsweise zwischen

[3] Definition im Glossar

10 % und 30 % des Umsatzes. Die Entwicklung eines Programms, das einer hohen Software-Code-Qualität entspricht, kann zum einen dessen Qualitätskosten senken, indem Fehler vermieden werden und die Fehlersuche durch gut verständlichen Code erleichtert wird. Zum anderen kann hochwertiger Code von nachhaltigem Nutzen sein, da das Ergebnis wiederverwendbar sein kann, in manchen Fällen sogar ohne Anpassungen (Krypczyk & Bochkor, 2022).

Nachhaltigkeit spielt in der heutigen Zeit eine erhebliche Rolle, weshalb es umso wichtiger ist Software-Code in einer hohen Qualität zu produzieren, welcher dadurch leicht wartbar, erweiterbar und wiederverwendbar ist. Dadurch können Ressourcen eingespart und eine nachhaltige Softwareentwicklung gewährleistet werden.

Das Ziel dieser Bachelorarbeit besteht darin, die Kriterien und Aspekte von guter Software-Code-Qualität in TYPO3-Extensions zu untersuchen, die für Entwickler und Entwicklerinnen besonders relevant sind und anhand dessen ihre Definitionen von Software-Code-Qualität zu ermitteln. Es soll zudem herausgefunden werden, wie die Bedeutung von Software-Code-Qualität in Projekten für die Entwicklung und Weiterentwicklung von TYPO3-Extensions wahrgenommen wird und ob Entwickler und Entwicklerinnen bereits ein Bewusstsein für die Qualität und deren Auswirkungen haben. Des Weiteren soll untersucht werden, wie die von den Entwicklern und Entwicklerinnen genannten Aspekte und Merkmale von guter Code-Qualität durch den Einsatz statischer Analyse-Tools gewährleistet werden können.

Die Untersuchung zielt darauf ab, einen Beitrag zur Nachhaltigkeit der TYPO3-Entwicklung zu leisten, indem Empfehlungen gegeben werden, wie der Software-Code wiederverwendbarer, verständlicher und wartbarer gestaltet werden kann. Diese Empfehlungen können nicht nur zu einer nachhaltigen Entwicklung von TYPO3-Extensions beitragen, sondern auch zu Kosteneinsparungen durch die Reduzierung von Fehlern. Des Weiteren kann damit Entwicklern und Entwicklerinnen eine angenehme Arbeitsatmosphäre und damit ein positives Arbeitsklima geboten werden.

1.3. Forschungsfrage

Es gibt bereits verschiedene Ansätze und Möglichkeiten, um die Software-Code-Qualität zu verbessern. Dazu gehören unter anderem die Anwendung von Best Practices und Standards, regelmäßige Code-Reviews[4] und die Durchführung automatisierter Tests. Weiterführend dienen Schulungen und Trainings für Entwickler und Entwicklerinnen einer Schärfung des Bewusstseins für die Bedeutung von Code-Qualität und der Vermittlung von erforderlichen Fähigkeiten und Kenntnissen zur Erstellung von qualitativ hochwertigem Code.

[4] Code-Reviews bezeichnen die manuelle Prüfung des erstellten Programmcodes durch andere Personen.

Ein weiterer wichtiger Aspekt kann der Einsatz von Software-Tools zur statistischen Code-Analyse sein. Diese Tools ermöglichen es Entwicklern und Entwicklerinnen potenzielle Probleme in Bezug auf Code-Qualität, Sicherheit und Performanz frühzeitig zu erkennen. Sie bieten Unterstützung und Hinweise, um Code in einer hohen Qualität zu schreiben. In dieser Bachelorarbeit liegt der Fokus auf der Analyse der Merkmale und Kriterien, die eine gute Software-Code-Qualität von TYPO3-Extensions beschreiben. Daraus ergibt sich folgende Forschungsfrage:
„Welche Merkmale und Kriterien sind wichtig für die Bewertung der Code-Qualität von TYPO3-Extensions und wie können statische Software-Analyse-Tools wie SonarQube Entwicklern und Entwicklerinnen dabei helfen, guten Code anhand dieser Kriterien zu schreiben?“
Es wird untersucht, welche Qualitätsmerkmale identifiziert, wie häufig sie genannt und wie sie beschrieben werden. Es wird weiterhin analysiert, wie Entwickler und Entwicklerinnen von Analyse-Tools profitieren können, um die von ihnen definierte Code-Qualität einzuhalten.
Diese Bachelorarbeit befasst sich mit den Aspekten der Code-Qualität von Extensions für CMS, am Beispiel von TYPO3-Extensions. Der Fokus dieser Arbeit liegt auf der Analyse von Kriterien und Merkmalen guter Code-Qualität von TYPO3-Extensions. Die Datenerhebung erfolgt mittels Interviews mit Fachexperten und -expertinnen. Die Analyse und Auslegung der Kriterien und Merkmale sollen den Hauptfokus dieser Arbeit bilden. Die Exploration der Art und Weise, wie Analyse-Tools diese Erkenntnisse umsetzen können, soll die Bachelorarbeit um einen weiteren Aspekt erweitern und zielt darauf ab, eine Methode zu erkunden, durch die Code-Qualität effektiver erreicht oder gewährleistet werden kann.

1.4. Hypothesen

Es ist anzunehmen, dass die befragten Personen die Verwendung von speziell für TYPO3 geeigneten Frameworks als Merkmal hoher Code-Qualität erwähnen werden. Die Befragten werden voraussichtlich auf die Abwägung der Nutzung solcher Frameworks eingehen. Insbesondere für Einsteiger könnte die Nutzung als hilfreich angesehen werden. Diese Annahme basiert auf der Tatsache, dass die Nutzung von Frameworks wie Extbase (siehe Kapitel 2.2.4 Frameworks) in der TYPO3-Dokumentation als nützliche Werkzeuge beschrieben werden, insbesondere für Anfänger und Anfängerinnen. Deren Nutzung jedoch auch negative Aspekte mitbringe, die in Betracht gezogen werden müssten.
Es wird erwartet, dass die Coding Guidelines von TYPO3, die eine breite Palette von Aspekten abdecken (siehe 2.2.3 TYPO3 Coding Guidelines) und ebenfalls in der der TYPO3-Dokumentation verfügbar sind, in den Interviews angesprochen werden. Diese Erwartung basiert darauf, dass insbesondere eher unerfahrene Entwickler und Entwicklerinnen bei der Entwicklung auf die offizielle Dokumentation zurückgreifen und im weiteren Verlauf ihrer Entwicklungstätigkeit das Gelernte fortlaufend anwenden.

Bei der Betrachtung des Konzepts der Software-Code-Qualität ist es unvermeidlich auf den Begriff Clean Code[5] zu stoßen. Daher ist zusätzlich anzunehmen, dass die Interviewten insbesondere die Aspekte von Clean Code ansprechen werden. Die Grundprinzipien von Clean Code sind vielfältig und spiegeln sich unter anderem in zentralen Aspekten wie Verständlichkeit, durchdachter Namensvergabe und bestimmten Leitprinzipien wieder (siehe 2.1.5 Clean Code Development).
Es wird weiterhin erwartet, dass die Verwendung von statischen Analyse-Tools wie SonarQube dazu beiträgt, die Code-Qualität von TYPO3-Extensions zu verbessern, indem die Tools Entwicklern und Entwicklerinnen dabei helfen, die von den Fachexperten identifizierten Merkmale von gutem Code umzusetzen. Diese Hypothese basiert auf der Annahme, dass die Tools durch automatisierte Analyse und Rückmeldungen die Code-Qualität verbessern können.
Die Ergebnisse dieser Arbeit werden Empfehlungen für die Entwicklung von TYPO3-Extensions liefern, die zu einer höheren Code-Qualität beitragen und somit eine nachhaltige Extension-Entwicklung fördern können. Diese Annahme basiert darauf, dass hochwertiger Code leichter verständlich, nachvollziehbar und einfacher zu erweitern ist, was wiederum den Zeitaufwand für die Entwicklung und Bearbeitung reduziert. Durch eine hohe Code-Qualität können Ressourcen eingespart und Kosten reduziert werden, was sich positiv auf die Nachhaltigkeit des Software-Projektes auswirkt.

1.5. Aufbau der Arbeit

Im ersten Abschnitt dieser Arbeit wird das Fundament für die Analyse und Bewertung der Code-Qualität gelegt. In diesem theoretischen Teil der Bachelorarbeit werden relevante Literaturquellen, sowie Forschungsarbeiten untersucht und somit der aktuelle Stand der Forschung ermittelt. Ziel ist es, theoretische Konzepte und Modelle zur Software-Code-Qualität vorzustellen, um ein umfassendes Verständnis für das Thema zu schaffen.
Im darauffolgenden Kapitel wird die geeignete Methodik detailliert beschrieben, mit deren Hilfe die Datenerhebung durchgeführt wird. Die Auswahl der Methode berücksichtigt dabei die Zielsetzung der Forschungsfrage.
Nach Abschluss der Datenerhebung werden die Ergebnisse präsentiert, interpretiert und darauffolgend diskutiert. Dabei werden die Erkenntnisse aus dem theoretischen Teil, sowie die erhobenen Daten zusammengeführt, um eine umfassende Bewertung der Code-Qualität von TYPO3-Extensions vorzunehmen. Die Diskussion ermöglicht es, die Ergebnisse kritisch zu reflektieren, die Forschungsfragen auf Grundlage der Ergebnisse zu beantworten und mögliche Impulse für die Praxis abzuleiten.

[5] Definition im Glossar

2. Grundlagen

Dieser Abschnitt vermittelt ein Verständnis für den allgemeinen Qualitätsbegriff, die Softwarequalität und die notwendigen Kenntnisse für die Entwicklung von TYPO3-Extensions. Der erste Teil behandelt die Grundlagen der Softwarequalität. Die Eingliederung der Code-Qualität wird untersucht, gefolgt von Maßnahmen zur Sicherstellung der Qualität. Diverse Ansätze und Methoden zur Gewährleistung von qualitativ hochwertigem Code während und nach der Entwicklung werden beleuchtet, um bestehende Wege zur Qualitätserhaltung des Software-Codes zu identifizieren.

Der darauffolgende Abschnitt fokussiert sich präziser auf die Entwicklung von TYPO3-Extensions. Theoretische Modelle und Konzepte im Kontext der TYPO3-Extension-Entwicklung mit der Programmiersprache PHP werden erläutert. Ziel ist es, ein umfassendes Verständnis für das erforderliche theoretische und technische Wissen zu vermitteln, um TYPO3-Extensions mit hoher Software-Code-Qualität zu entwickeln.

Zuletzt werden aktuelle Forschungsarbeiten analysiert um den Stand der Dinge zu beleuchten.

2.1. Qualitätssicherung

Balzert (2008) beschreibt alle Tätigkeiten, die die Qualitätsanforderungen sicherstellen sollen, als Aufgabe des Qualitätsmanagements. Dieses kann in der Softwareentwicklung jedoch nicht verglichen werden mit dem Qualitätsmanagement traditioneller Produktionsprozesse, da bei einer traditionellen Produktion ein iterativer Prozess vorliegt. Software wird jedoch in der Regel individuell erstellt. Die Maßnahmen und Schritte der traditionellen Qualitätssicherung können demnach nicht angewendet werden, da es sich bei der Softwareentwicklung nicht um eine Entwicklung mit einer sich wiederholenden Anwendung desselben, eindeutigen Entwicklungsprozesses handelt (Balzert, 2008).

Die Qualitätssicherung kann als Teil des Qualitätsmanagements gesehen werden und soll im weiteren Verlauf dieser Arbeit technikorientierte Aktivitäten zur Sicherung der Qualität beschreiben. Da der Fokus auf der Software-Code-Qualität liegt, werden insbesondere Maßnahmen und Techniken vorgestellt, mit deren Hilfe die Code-Qualität während, sowie nach der Entwicklung des Codes sichergestellt wird.

2.1.1. Qualitätsbegriff

Die Auffassung von Qualität ist subjektiv. Die Vorstellung davon, was ein qualitativ hochwertiges Produkt ausmacht, kann unterschiedlich sein. Balzert (2008) beschreibt eine mögliche Definition: „Qualität ist die anhand von vorgegebenen, vereinbarten oder erwarteten Merkmalen gemessene Eigenschaft einer Einheit“ (Balzert, 2008, S. 461).

2.1.1.1. Softwarequalität

Die Vorstellung von allgemeiner Qualität kann auf Software übertragen werden, jedoch können in der Theorie unterschiedliche Sichtweisen herangezogen werden (Krypczyk & Bochkor, 2022). Krypczyk und Bochkor (2022) beschreiben einige Sichtweisen, mit deren Hilfe der allgemeine Qualitätsbegriff betrachtet werden kann und stellen fest, dass der anwenderbezogene Ansatz am besten für die Softwarequalität geeignet ist. Sie ziehen daraus die Schlussfolgerung, dass unter der Produktqualität von Software meist der anwendungsbezogene Ansatz gemeint ist, also die Sichtweise der Kundschaft, beziehungsweise der Anwender und Anwenderinnen.

„Mängel in der Codierung führen zum Aufbau sogenannter technischer Schulden. Diese gilt es zu vermeiden bzw. schnell wieder abzubauen." (Krypczyk & Bochkor, 2022, S. 763). „[…] Auch die Nichtanwendung von gebotenen Entwurfsmustern, ein schlechter Programmierstil oder das Ignorieren von Coding-Standards gehören dazu." (Krypczyk & Bochkor, 2022, S. 763). Sie beschreiben die Auswirkungen der technischen Schulden ausführlich mit der Erkenntnis, dass technische Schulden[6], zu denen auch ein minderer Programmierstil gehört, später hohe Kosten verursachen kann, da notwendige Reparaturmaßnahmen an der Software aufwendig werden. Krypczyk und Bochkor (2022) verdeutlichen damit den Ansatz, dass die Qualität sich maßgeblich durch die Sichtweise der Kundschaft bestimmen lässt, da diese das Produkt finanzieren. Es gilt somit ihre Anforderungen zu erfüllen, jedoch müssen die inneren Werte, die unter anderem durch die Code-Qualität beeinflusst werden, ebenfalls übereinstimmen, damit den externen Ansprüchen der Kundschaft über einen längeren Zeitraum gerecht werden kann.

Krypczyk und Bochkor (2022) erläutern, dass der prozessbezogene Ansatz in der Softwarequalität deshalb nicht außer Acht gelassen werden darf, da dies langfristig Auswirkungen auf die Gebrauchstauglichkeit haben kann. Dieser Ansatz konzentriert sich hauptsächlich auf den Produktionsprozess, also die tatsächlichen Programmieraufgaben.

Hoffmann (2013) gliedert die Softwarequalität in zwei grundlegende Bereiche. Zum einen die Produktqualität und zum anderen die Prozessqualität (siehe Abbildung 1: Einteilung Software-Qualitätssicherung). Zur Produktqualität zählen dabei alle Maßnahmen, die die Qualität des eigentlichen Produktes während und nach der Entwicklung verbessern. Der Bereich der Prozessqualität umfasst „Maßnahmen, die eine Entwicklung des Software-Produkts in Form eines definierten Prozesses gewährleisten" (Hoffmann, 2013, S. 25). Die Qualität des Software-Codes gehört zu dem Bereich Produktqualität, der weiter eingeteilt wird in die konstruktive und die analytische Qualitätssicherung (Hoffmann, 2013). Weitere Analysen und Untersuchungen dieser Bachelorarbeit sollen diese beiden Bereiche fokussieren.

[6] Definition im Glossar

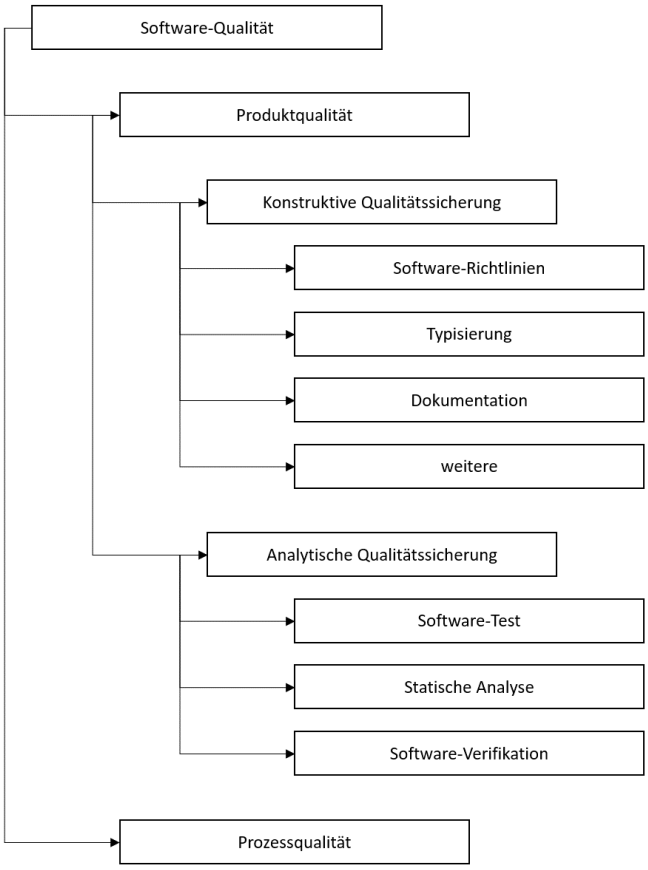

Abbildung 1: Einteilung Software-Qualitätssicherung
Quelle: In Anlehnung an Hoffmann, 2013, S.20

2.1.1.2. Code-Qualität

Auch Balzert (2008) unterscheidet die zwei Bereiche der konstruktiven und analytischen Maßnahmen zur Qualitätssicherung in der Softwareentwicklung. Das können Instrumente, Prinzipien, Methoden und weitere Hilfsmittel sein, mit denen die Qualität des eigentlichen Software-Produktes eingehalten und sichergestellt werden kann.

Konstruktive Maßnahmen sorgen dafür, dass das entstehende Produkt, also die Software bestimmte Qualitätseigenschaften besitzt. Balzert (2008) führt hierfür einige Beispiele für Maßnahmen auf, die die Programmiersprache betreffen, wie zum Beispiel eine statische Typüberprüfung aber auch Richtlinien, wie ein fest vorgegebenes Gliederungsschema für ein Pflichtenheft. Im Gegensatz zu diesen konstruktiven Maßnahmen beschreibt Balzert (2008) als zweite Maßnahmen-Sammlung analytische Maßnahmen, die als diagnostische Maßnahmen dem Produkt keine Qualität im eigentlichen Sinne bringen, jedoch das existierende Qualitätsniveau messen. Ziel dieser Maßnahmen ist es, dass innere Produkt im Hinblick auf die Qualität zu prüfen und zu bewerten.

Schneider (2012) führt neben der konstruktiven und analytischen Qualitätssicherung noch den dritten Bereich der organisatorischen Qualitätssicherungsmaßnahmen auf. Er beschreibt

diese Maßnahmen, zu denen Aufbau- und Ablauforganisation gehören, als „Voraussetzung dafür, dass analytische und konstruktive Maßnahmen durchführbar und wirksam sind.“ (Schneider, 2012, S. 177).

Die Definition der konstruktiven Qualitätssicherung „Maßnahmen, die bereits bei der Konstruktion von Software auf die Verbesserung ausgewählter Qualitätsaspekte abzielen und nicht erst nachträglich durch Prüfung und Korrektur“ (Schneider, 2012, S. 177) geht darauf ein, dass Fehler gar nicht erst entstehen sollen, damit sie im späteren Verlauf nicht gesucht und mühsam entfernt werden müssen. Des Weiteren betont Schneider (2012), dass laut dieser Definition ein Zusammenhang bestehen sollte zwischen der eingeleiteten Maßnahme und dem angepeilten Qualitätsziel. Als wichtigste Qualitätsaspekte, die häufig adressiert werden nennt Schneider (2012) die Fehlerfreiheit, Flexibilität, Bedienbarkeit, bzw. Usability und Laufzeiteffizienz.

Die Qualitätsaspekte, die aus der Literaturrecherche hervorgegangen sind, sollen bei der Auswertung der im Rahmen dieser Forschungsarbeit erhobenen Daten für die Kategorien-Bildung der qualitativen Inhaltsanalyse genutzt werden. Die Forschung und der Inhalt dieser Bachelorarbeit werden sich primär auf die Aspekte und den Bereich der konstruktiven Qualitätssicherung beziehen.

2.1.2. Messung der Qualität

Softwarequalität kann durch Qualitätsmerkmale, die weiter eingeteilt werden können in Teilmerkmale, beschrieben werden. Diese Merkmale werden durch Qualitätsindikatoren messbar und bewertbar gemacht. Die Indikatoren sind bestimmte Attribute des Softwareproduktes, die zu den Qualitätsmerkmalen in Beziehung stehen. Diese können mit Hilfe von Qualitätsmaßen gemessen werden. Das Maß ist dabei eine quantitative Skala, mit deren Hilfe der Wert bestimmt werden kann, den ein Indikator aufweist. (Balzert, 2008). Solche Qualitätsmodelle können herangezogen werden, um die Qualität einer Software zu messen.

In der umfangreichen Literatur zu diesem Thema finden sich weitere zahlreiche Konzepte zur Messung der Softwarequalität, von denen einige bereits lange existieren, aber dennoch relevante Ansätze liefern. Boehm et al. (1976) unterteilen die Qualitätskriterien in verschiedene Gruppen und nennen spezifische Merkmale, die oft mithilfe von Metriken gemessen werden können. Zu den Qualitätsgruppen gehören beispielsweise die Portabilität, die Testbarkeit, sowie die Verständlichkeit, Lesbarkeit und Änderbarkeit (siehe Abbildung 2: Qualitätskriterien nach Boehm et al. (1976)). Diese Kriterien können nicht nur auf die äußere Softwarequalität angewendet werden, sondern auch auf die innere Qualität, den eigentlichen Code der Software.

In dieser Bachelorarbeit liegt der Fokus nicht auf der Betrachtung der gesamten Softwarequalität, sondern vielmehr auf dem Qualitätsbereich des Software-Codes. Diesem Bereich können einige Metriken, Tests und andere Verfahren zugeordnet werden, um den Qualitätsgrad zu erfassen. Allerdings können sich diese Ansätze innerhalb der einzelnen Bereiche unterscheiden. Im weiteren Verlauf wird nicht näher auf Metriken und Messungen der Qualität eingegangen, der Fokus wird darauf gelegt Qualitätsmerkmale zu erfassen und zu analysieren, ob und wie diese sichergestellt werden können.

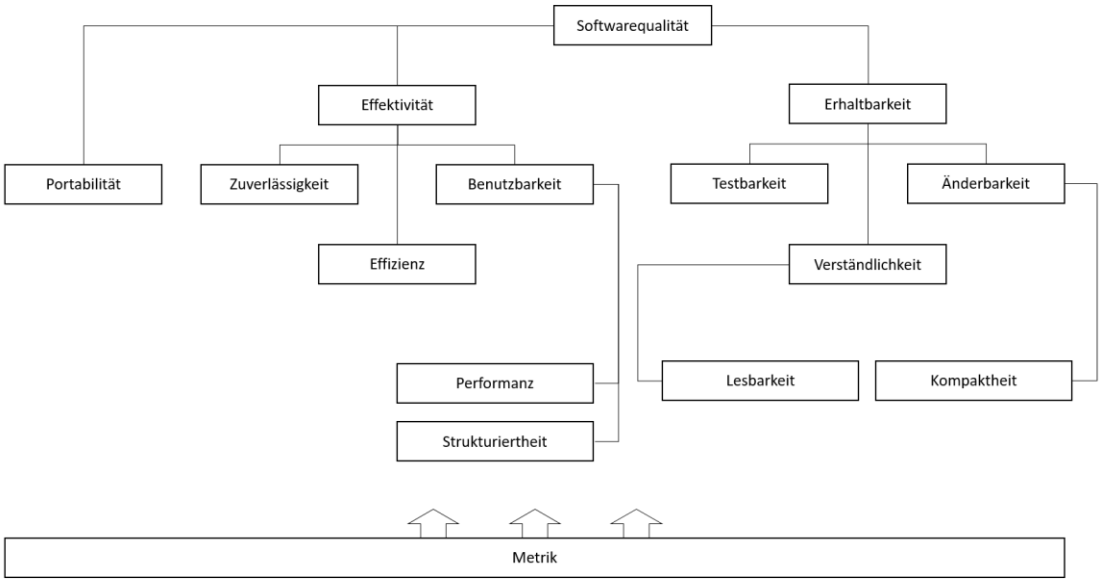

Abbildung 2: Qualitätskriterien nach Boehm et al. (1976)
Quelle: In Anlehnung an Boehm et al. (1976), S. 595

2.1.3. Konstruktive Qualitätssicherung

Methoden und Techniken aus dem Gebiet der konstruktiven Maßnahmen zur Qualitätssicherung sind darauf ausgerichtet die Qualitätsmerkmale des Software-Produktes a priori zu erfüllen (Hoffmann, 2013). „Im Besonderen fallen hierunter alle Maßnahmen, die entwicklungsbegleitend zur Anwendung kommen und dem Software-Entwickler damit die Möglichkeit eröffnen, Fehler frühzeitig zu erkennen oder gar von vorne herein zu vermeiden.“ [7] (Hoffmann, 2013, S. 65).

2.1.3.1. Richtlinien

Hoffmann (2013) zählt als Maßnahmen und Vorgehensweisen der konstruktiven Qualitätssicherung mehrere Aspekte auf, unter anderem Software-Richtlinien. Hoffmann (2013) beschreibt Richtlinien als Menge von Konventionen, die den Gebrauch einer bestimmten Programmiersprache regelt. Diese Regelungen gehen über die eigentlichen

[7] Das Zitat wurde an dieser Stelle unverändert wiedergegeben, weshalb es keine gendergerechte Schreibweise enthält.

syntaktischen und semantischen Beschränkungen dieser Programmiersprache hinaus. Mit Hilfe von Software-Richtlinien können Eckpunkte für die Verwendung von bestimmten Programmiersprachen festgelegt werden. Damit soll geregelt werden welche Konstrukte einer Sprache verwendet werden sollen. Sie können in unternehmensspezifischen Vorschriften vorkommen. Darüber hinaus haben sich auch firmenübergreifende, allgemeine Standards etabliert (Hoffmann, 2013). Eine Umsetzung von Richtlinien gibt es im Bereich von PHP, die „PHP Standard Recommendation“ (PSR), die von der Gruppe „Framework Interoperability Group“ (FIG) erstellt wurden und weiterentwickelt werden. Die Idee hinter dieser Gruppe ist, dass Vertreter und Vertreterinnen verschiedener Projekte über Gemeinsamkeiten ihrer Projekte sprechen und darüber sprechen, wie sie zusammenarbeiten können (*Frequently asked questions - PHP-FIG*, o. D.). Auch TYPO3 ist Mitglied dieser Gruppe. Es gibt eine Reihe von Regeln, die in unterschiedliche Bereiche eingeteilt sind, wie beispielsweise Basic Coding Standards.

2.1.3.2. Typisierung

Ein weiterer Aspekt, den Hoffmann (2013) aufführt ist die Typisierung. Mit dem Einsatz typisierter Programmiersprachen können Fehler bereits frühzeitig identifiziert werden. Programmiersprachen der früheren Generationen kamen ohne den Einsatz von Datentypen aus. Neben der Erkennung von Inkonsistenzen im Code nennt Hoffmann (2013) als Vorteil die Erhöhung der Lesbarkeit, da sie damit einen dokumentierenden Charakter bekommen. Hoffmann (2013) beschreibt die Typisierung in Programmiersprachen als eines der mächtigsten Hilfsmittel der konstruktiven Qualitätssicherung.

Die Typisierung unterteilt er in zwei grundsätzlich unterschiedliche Arten. Zum einen die statischen und zum anderen die dynamischen Typen. Bei den statischen Typen werden Variablen zusammen mit ihrem Datentyp deklariert, dieser Variable kann während ihrer gesamten Lebensdauer nur ein Objekt zugewiesen werden, welches mit dem statisch festgelegten Datentyp übereinstimmt. In Programmiersprachen, die statische Typisierung benutzten, können Daten-Inkonsistenzen frühzeitig entdeckt werden. Im Gegensatz dazu stehen die dynamischen Typisierungen. In Programmiersprachen, die dynamisch typisieren, sind Variablen nicht an einen expliziten Datentyp gebunden. Eine Variable kann dabei zu unterschiedlichen Zeitpunkten unterschiedliche Objekte referenzieren (Hoffmann, 2013).

PHP gehört zu den Programmiersprachen, die diese Typisierung nutzten. Hoffmann (2013) nennt als Vorteile dieser Typisierung unter anderem die Flexibilität und eine mögliche Steigerung der Produktivität, da Variablen vor der Verwendung nicht deklariert werden müssen. Jedoch bringt diese auch Nachteile mit sich. Als Beispiel für negative Aspekte führt Hoffmann (2013) die Möglichkeit von Inkonsistenzen auf, die nur in begrenztem Umfang aufgedeckt werden.

Programmiersprachen können weiterhin hinsichtlich der Rigidität unterschieden werden. In diesem Zusammenhang wird von einer schwachen oder starken Typisierung gesprochen (Hoffmann, 2013). Hoffmann (2013) teilt die Programmiersprache PHP in die Kategorie der schwachen Typisierung ein. Dabei kann einer Variablen mit Hilfe von in den Code eingefügten Cast-Methoden ein anderer Datentyp zugewiesen werden. Neben einem hohen Maß an Flexibilität als Vorteil nennt Hoffmann (2013) als Nachteil eine gesteigerte Fehleranfälligkeit, da die Verantwortung der Einhaltung der Typkonsistenz und der damit einhergehenden Vermeidung von Inkonsistenz-Fehlern, vollständig auf den Programmierer oder die Programmiererin übertragen wird. Hoffmann (2013) betont, dass der Einsatz einer typisierten Sprache helfen kann, Inkonsistenzen zu identifizieren.

2.1.3.3. Dokumentation

Die Programmdokumentation gehört zu den internen Dokumenten, die nicht zugänglich sind für die Auftraggebenden. Dieses Dokument muss von den Entwicklern und Entwicklerinnen selbst erstellt werden und gehört für diese zu den wichtigsten Aufgaben (Hoffmann, 2013). Ein großer Zeitdruck in Software-Projekten kann dazu führen, dass die Erstellung einer Dokumentation nur spärlich durchgeführt oder gar gänzlich weggelassen wird.
Ein weiterer Aspekt der Dokumentation sind Kommentare. Kommentare sind Textabschnitte, die direkt im Quellcode eingebunden werden. Sie werden beim Ausführen des Quellcodes ignoriert und haben mit der eigentlichen Programmerstellung nichts zu tun. Ziel der Kommentare soll sein, den Code später lesbarer zu gestalten (Krypczyk & Bochkor, 2022). Sie können bei der Verständlichkeit unterstützen, wenn der produzierte Code über einen längeren Zeitraum nicht bearbeitet wurde. Krypczyk und Bochkor (2022) beschreiben Kommentare außerdem als Unterstützung für Personen, die sich in fremden Code einarbeiten müssen.
Es gibt Unterscheidungen in der Art von Kommentaren, wie einzeilige oder mehrzeilige und in jeder Programmiersprache kann es Unterschiede in der Erstellung geben.

2.1.4. Analytische Qualitätssicherung

Analytische Maßnahmen umfassen alle Methoden und Techniken, die ein Software-Produkt bezüglich der Qualitätseigenschaften nach erfolgter Entwicklung bewerten. Im Gegensatz zu den konstruktiven Maßnahmen, handelt es sich um diagnostische Maßnahmen, die Qualitätseigenschaften a posteriori untersuchen (Hoffmann, 2013). Hoffmann (2013) unterscheidet hierfür die Teilgebiete Software-Test, statische Analyse und Software-Verifikation (siehe Abbildung 1: Einteilung Software-Qualitätssicherung). Im weiteren Verlauf dieser Bachelorarbeit werden die beiden Punkte Software-Test und statische Analyse untersucht. Software-Verifikationen beziehen sich auf mathematische Methoden und

Techniken, die darauf abzielen, die Korrektheit von Software nachzuweisen (Hoffmann, 2013). Da dieser Bereich ebenfalls sehr umfassend ist, erfolgt in dieser Bachelorarbeit keine vertiefende Untersuchung.
Balzert (2008) beschreibt die Software-Tests als testende Verfahren, in denen die zu testende Software ausgeführt wird. Die statische Analyse, die gezielt Informationen sammelt ohne die Software auszuführen, wird analysierendes Verfahren genannt.
Balzert (2008) verdeutlicht, dass konstruktive und analytische Maßnahmen voneinander abhängig sind, da fehlende oder geringe konstruktive Maßnahmen in der Regel aufwendige analytische Maßnahmen erfordern. Laut Balzert (2008) kann eine vorausschauende, konstruktive Qualitätslenkung viele analytische Maßnahmen ersparen.

2.1.4.1. Software-Test

Hoffmann (2013) bezeichnet die Software-Tests als gängigste Methode, um Software-Fehler zu entdecken. Dabei wird die zu untersuchende Software mit vordefinierten Eingabewerten ausgeführt und die Ausgabe mit einem erwarteten Ergebnis verglichen. Da nicht alle Eingabekombinationen explizit testbar sind, müssen die Tests dabei auf eine bestimmte Auswahl an Testfällen beschränkt werden. In der heutigen Softwareentwicklung stehen zahlreiche Tests zur Verfügung, die eingesetzt werden können, um die Qualität von Software zu überprüfen. Eine umfassende Analyse und Beschreibung dieser Tests würde den Umfang dieser Bachelorarbeit überschreiten.

2.1.4.2. Statische Analyse

Mit der statischen Analyse können syntaktische oder semantische Programmeigenschaften aus den Quelltexten abgeleitet werden. Diese Analyseart ist von statischer Natur, was bedeutet, dass sie durch das Durchsehen des Quellcodes erfolgt, ohne diesen in ein ausführbares Programm zu wandeln. Im Gegensatz zu den Methoden der konstruktiven Qualitätssicherung wird sie immer auf bereits erstellten Softwarecode angewendet (Hoffmann, 2013).
Der Bereich der statischen Analyse besteht aus mehreren Unterkategorien, wie Software-Metriken, der Konformitätsanalyse oder der manuellen Softwareprüfung (Hoffmann, 2013).
Software-Metriken wurden in einem vorherigen Kapitel bereits erwähnt (siehe Kapitel 2.1.2 Messung der Qualität). Metriken sind ein Hilfsmittel, mit dem Aspekte einer Software systematisch und quantitativ erfasst werden können. Auch der Bereich der Metriken ist sehr umfangreich und wird im weiteren Verlauf der Bachelorarbeit nicht näher beschrieben.
Hoffmann (2013) unterteilt die Konformitätsanalyse in Bereiche, zu denen die syntaktische und die semantische Prüfung gehören. Er beschreibt die syntaktische Prüfung als Analyse der Einhaltung von lexikalischen und syntaktischen Vorgaben der jeweiligen Programmiersprache.

Die semantische Prüfung geht über die syntaktische hinaus und soll fragwürdige oder potentiell fehlerhafte Programmstrukturen aufdecken (Hoffmann, 2013).
Zur Unterstützung der statischen Codeanalyse können Tools genutzt werden, die eine automatisierte Überprüfung durchführen. Diese Tools werden im Verlauf der weiteren Bachelorarbeit betrachtet.

2.1.5. Clean Code Development

Clean Code Development beschreibt eine Vorgehensweise, in der die Entwicklung eines „sauberen" Codes im Mittelpunkt steht. Krypczyk & Bochkor (2022) beschreiben Clean Code Development als Einstellung der Entwickler und Entwicklerinnen „sauberen" Code zu schreiben.
Beneken et al. (2022) betonen die Bedeutung von Verständlichkeit, sowie Lesbarkeit und Änderbarkeit im Softwarecode als zentrale Aspekte eines Clean Codes. Sie erläutern, dass Verständlichkeit sich darauf bezieht, wie gut ein Code von Entwicklern und Entwicklerinnen in ihrer Entwicklungsumgebung verstanden und im begrenzten Arbeitsgedächtnis verarbeitet werden kann. Eine wesentliche Rolle spielt hierbei die Wahl der Namen für Klassen, Methoden und Attribute. Durch die Verwendung aussagekräftiger Namen wird das Verständnis des Codes erleichtert (Beneken et al., 2022).
Änderungen sollten nur wenige Stellen betreffen. Als Faustregel geben Beneken et al. (2022) an, dass eine gute Code-Änderbarkeit daran erkennbar ist, wie wenige Entwicklungsfenster geöffnet werden müssen, um eine Änderung durchzuführen. Um unerwünschte Seiteneffekte zu verhindern, wird von Beneken et al. (2022) beispielsweise empfohlen, auf änderbare globale Variablen zu verzichten. Zudem sollte eine Änderung leicht lokal und ohne großen Aufwand testbar und anschließend auch auslieferbar sein, ohne das gesamte Produkt neu ausliefern zu müssen (Beneken et al., 2022).
Krypczyk und Bochkor (2022) benennen ebenfalls einige Aspekte, die einen „sauberen" Code ausmachen. Das sind zum einen die Namensgebung. Eine durchdachte Namensvergabe bei Bezeichnern, wie Variablen oder Klassen sollte eingeführt und eingehalten werden. Als weiteren Aspekt nennen sie verständliche Funktionen. Krypczyk und Bochkor (2022) betonen, dass die Zeilenzahl in Funktionen begrenzt sein sollte, damit andere Entwickler oder Entwicklerinnen, die Funktion nach dem Durchlesen vollständig verstehen.
Es sollten weiterhin keine Aktionen innerhalb von Funktionen vermischt werden und es sollten lange Parameterlisten vermieden werden. Des Weiteren führen Krypczyk und Bochkor (2022) auf, dass Kommentare nicht als überflüssig angesehen werden sollten und dass Wiederholungen im Code vermieden werden sollten. In diesem Kontext erwähnen sie auch Clean-Code Prinzipien, wie KISS und SOLID. KISS steht für „Keep It Simple Stupid" und soll dafür

sorgen, dass Code möglichst einfach gehalten wird. Eine generische Lösung soll nur dann sinnvoll sein, wenn es tatsächlich mehrere Anwendungsfälle gibt.
„SOLID“ steht für die Prinzipien Single-Responsiblity, Open-Closed-Prinzip, Liskov’sches Substitutionsprinzip, Interface-Segregation-Prinzip und Dependency-Inversion-Prinzip. Die Prinzipien sind grundlegende Leitlinien im Bereich des Clean Code (Krypczyk & Bochkor, 2022). Auf die einzelnen Aspekte des SOLID-Prinzips wird nicht näher eingegangen.
Im Zusammenhang mit Clean Code wird häufig der Begriff Code Smell[8] verwendet. Martin (2009) erstellte eine Liste von Dingen, die im Code negativ auffallen, diese werden als Code Smell bezeichnet. Der Begriff hat sich etabliert, um auf Probleme oder Anomalien im Programmcode aufmerksam zu machen.

2.2. TYPO3

In diesem Abschnitt erfolgt eine kurze Einführung in das CMS TYPO3 und eine Beschreibung der grundlegenden Struktur einer Extension für TYPO3. Dabei werden die Bestandteile einer Extension vorgestellt, die wichtig sind für die Betrachtung der Code-Qualität.

2.2.1. Open Source

TYPO3 ist eine Open Source Software. Open Source beschreibt einen bestimmten Lizenztypen. Open Source Software wird üblicherweise dezentral und mit Hilfe von modularen Konzepten entwickelt, an denen sich sowohl Unternehmen, als auch Privatpersonen beteiligen können. Der Unterschied zu anderen Lizenztypen liegt hauptsächlich darin, dass Open Source-Anwendungen keine Gebühren für Lizenzen erheben können und dass der Quellcode frei zugänglich ist und jederzeit bearbeitet werden kann (Krypczyk & Bochkor, 2022).

2.2.2. Bestandteile TYPO3-Extension

Das TYPO3 CMS ist vollständig auf dem Konzept von Extensions aufgebaut. Sogar der Kern von TYPO3 besteht aus Extensions, die als System Extensions bezeichnet werden. Einige davon sind notwendig und immer aktiviert. Andere Extensions werden von den Mitgliedern der TYPO3-Community entwickelt.
Eine TYPO3-Extension besteht aus mehreren zusammenarbeitenden Komponenten. Der grundlegende Bestandteil ist der Extension-Ordner, denn jede TYPO3-Extension hat einen eigenen Ordner. In diesem Ordner befinden sich Standard-Dateien mit reservierten Namen für die Konfiguration. Darüber hinaus können weitere Dateien vorhanden sein, um die Funktionalität der Extension zu unterstützen (*Extension Development — TYPO3 explained 12.4 documentation*, 2023).

[8] Definition im Glossar

Der Backend-Bereich einer TYPO3-Extension besteht aus verschiedenen Dateien. In dieser Forschung wird die Betrachtung jedoch auf den PHP-Dateien einer Extension liegen, um zu analysieren, wie hohe Qualität von PHP-Code beschrieben und produziert wird.

2.2.3. TYPO3 Coding Guidelines

Es gibt Coding-Guidelines, die in der TYPO3-Dokumentation hinterlegt sind. Die Einhaltung dieser Regeln ist zwar obligatorisch, Extension-Entwickler und -Entwicklerinnen werden jedoch auf der TYPO3-Dokumentations-Seite der Coding-Guidelines dazu ermutigt diese Richtlinie zu befolgen.
Durch die Einhaltung der Richtlinien soll das Lesen des Codes erleichtert und die Verständlichkeit des Codes verbessert werden. Sie sollen auch dabei helfen typische Fehler im TYPO3-Code zu vermeiden (*Introduction — TYPO3 explained main documentation*, 2023).
Es wird empfohlen die IDE so einzustellen, dass die Standards automatisch überprüft werden. Als Möglichkeit die IDE zu konfigurieren wird die Nutzung einer Editor-Config genannt. Im TYPO3 Source-Code ist eine solche Datei enthalten, in der Vorgaben für die verschiedenen Programmiersprachen, die in TYPO3-Extensions vorkommen angegeben werden.
Die Coding-Guidelines beinhalten Empfehlungen die für Programmierung in PHP. Diese Leitlinien sind eingeteilt in die Bereiche Formatierung, Architektur und Best Practices.

2.2.4. Frameworks

Auf der TYPO3 Dokumentations-Seite wird Extbase als Framework beschrieben, welches unter anderem auf dem MVC-Pattern[9] basiert. Auf der einen Seite müssen Entwickler und Entwicklerinnen mit Nutzung dieses Frameworks weniger Code schreiben. Auf der anderen Seite könnte jedoch die Performanz darunter leiden (*ExtBase Introduction — TYPO3 explained 12.4 documentation*, 2023).
In der TYPO3-Dokumentation wird zur Nutzung von Extbase erwähnt, dass von jedem Entwickler und jeder Entwicklerin selbst entschieden werden soll, ob sie dieses Framework nutzen wollen, es wird jedoch insbesondere für Anfänger und Anfängerinnen empfohlen. Die Nutzung dieses Frameworks wird nicht als Merkmal guter Code-Qualität beschrieben, sondern muss situationsbedingt betrachtet werden.
Ein weiteres erwähnenswertes Framework, in diesem Kontext ist Symfony. Es ist ein PHP-Framework, das verwendet wird, um die Entwicklung von TYPO3-Extensions und Komponenten zu erleichtern. Symfony ist eine umfassende Plattform im Bereich der Webentwicklung, bestehend aus einer Sammlung von PHP-Komponenten, einem Webanwendungs-Framework und einer Gemeinschaft von Entwicklern und Entwicklerinnen.

[9] Definition im Glossar

Die zugrundeliegende Philosophie von Symfony betont bewährte Praktiken sowie die Standardisierung und Interoperabilität von Anwendungen. Diese Technologie wird von vielen PHP-Anwendungen bereits genutzt (Symfony, o. D.). TYPO3 hat ebenfalls begonnen, Symfony-Komponenten in seine Codebasis zu integrieren, um von den Vorteilen dieses Frameworks zu profitieren.

2.3. Aktuelle Forschungsarbeiten

Im Bereich der Software-Code-Qualität existiert bereits eine Vielzahl umfangreicher Forschungsarbeiten und Literatur.

Ein sehr bekanntes und häufig empfohlenes Buch, welches den mit Code-Qualität in Zusammenhang stehenden Begriff Clean Code sehr geprägt hat, ist „Clean Code: A Handbook of Agile Software Craftsmanship“ von Robert C. Martin. In diesem Buch werden Prinzipien, Techniken und Best Practices zur Erstellung von sauberem und wartbarem Code vorgestellt. Es ist eine häufig empfohlene Ressource, die Entwicklerinnen und Entwicklern als Leitfaden dienen kann, um ihre Programmierfähigkeiten zu verbessern und qualitativ hochwertigen Code zu schreiben. Die Veröffentlichung des Buches im Jahr 2008 und die Neuauflage im Jahr 2019 zeigen deutlich, dass das Thema Code-Qualität in der Softwareentwicklung nicht nur schon seit langer Zeit von großer Bedeutung ist, sondern auch heutzutage immer noch relevant ist.

In einer Untersuchung stellten Cheng, et al. (2022) fest, dass Code-Qualität, zusammen mit anderen Faktoren, kausal mit der selbstberichteten Produktivität von Entwicklern und Entwicklerinnen verbunden ist. In dieser Studie wurden zwei Analysen durchgeführt. Die erste Analyse zeigt, dass unter anderem Code-Qualität, sowie technische Schulden eine kausale Verbindung zur selbstberichteten Entwicklerproduktivität haben. Die zweite Analyse wurde genutzt, um die kausalen Schlussfolgerungen zu stärken. Sie belegt, dass gesteigerte wahrgenommene Code-Qualität normalerweise von einer erhöhten wahrgenommenen Entwicklerproduktivität begleitet wird. Diese Ergebnisse liefern eine Evidenz dafür, dass Code-Qualität die individuelle Entwicklerproduktivität beeinflusst.

Hochwertige Code-Qualität kann somit positive Auswirkungen auf die Softwareentwicklung haben und ist damit ein Forschungsgebiet, das viel Potenzial hat.

Martin (2009) identifiziert verschiedene Aspekte der Code-Qualität, wie beispielsweise die Verständlichkeit und Lesbarkeit von Code. Diese Aspekte, sowie weitere relevante Faktoren, dienen als Merkmale des Leitfadens für die Befragung der Fachexperten und -expertinnen in dieser Bachelorarbeit.

3. Methodik

Die Erhebung der Daten dieser Bachelorarbeit wurde mit qualitativen Forschungsmethoden durchgeführt. Es wurden überwiegend nicht standardisierte Verfahren eingesetzt mit dem Ziel, Aussagen über möglichst viele Merkmale der Software-Code-Qualität im Kontext von TYPO3-Extension-Programmierung zu erheben (Scheufele & Engelmann, 2009).
Für die Datenerhebung wurde die Methode der Interviews angewendet. Als Interviewpartner und –partnerinnen wurden Personen ausgewählt, die im Bereich der TYPO3-Extension-Programmierung tätig sind und über Fachwissen in der Entwicklung von Software-Code verfügen, sei es bei der Erstellung neuer TYPO3-Extensions oder bei der Erweiterung und Anpassung bestehender Extensions.
Zur Auswertung der Befragungsantworten wurden die durchgeführten Interviews transkribiert und anschließend einer strukturierten Inhaltsanalyse unterzogen, die in einem folgenden Kapitel näher erläutert wird. Dabei wurde die qualitative Inhaltsanalyse verwendet, da die Interviews als Leitfadengespräche durchgeführt wurden und um relevanten Argumente zu erkunden und daraus wichtige Themen im Kontext der Code-Qualität abzuleiten (Scheufele & Engelmann, 2009).

3.1. Datenerhebung

Für die Beantwortung der Forschungsfrage und die Gewinnung geeigneter Daten ist ein entsprechendes Vorgehen erforderlich, das auf die spezifische Fragestellung abgestimmt ist. Die Art der Datenerhebung spielt dabei eine entscheidende Rolle, da verschiedene Methoden die Ergebnisse beeinflussen können (Aeppli et al., 2016).
In dieser Bachelorarbeit wurden die Daten mithilfe von Interviews erhoben. Die Wahl dieser Methode ermöglicht es, detaillierte Einblicke und persönliche Erfahrungen der Experten und Expertinnen zu gewinnen. Im weiteren Verlauf wird das Vorgehen bei der Durchführung der Interviews genauer beschrieben und erläutert.

3.1.1. Auswahl der Interviewten

Insgesamt wurden sechs Interviews mit Fachexperten und -expertinnen durchgeführt. Die Auswahl der geeigneten Personen erfolgte anhand mehrerer Kriterien. Zum einen wurde darauf geachtet, dass die ausgewählten Personen über langjährige Erfahrung in der Entwicklung und Weiterentwicklung von TYPO3-Extensions verfügen.
Ein weiteres Auswahlkriterium war, dass die Interviewpartner und -partnerinnen einen signifikanten Anteil ihrer Arbeit im Bereich der Backend-Programmierung verrichten oder relevante Erfahrungen in diesem Kontext besitzen.
Zusätzlich wurde berücksichtigt, dass die Befragten aus verschiedenen beruflichen Umfeldern stammen, um sicherzustellen, dass firmeninterne Arbeitsweisen oder Standards, die

möglicherweise nicht im weitläufigen TYPO3-Umfeld üblich sind, die Ergebnisse nicht einseitig beeinflussen.

3.1.2. Befragung

Es wurden strukturierte Interviews durchgeführt, die jedoch nicht standardisiert sind, da den Befragten keine Antwortmöglichkeiten vorgegeben wurden. Diese Art des Interviews eignet sich besonders gut für diese Forschung, da der Schwerpunkt dieser Bachelorarbeit nicht auf der Erhebung von statistisch auswertbaren Daten liegt (Balzert et al., 2017), sondern darauf abzielt den Themenbereich der Software-Code-Qualität im Kontext von TYPO3-Extension-Entwicklung zu erkunden.

Der Gesprächsverlauf wurde flexibel gestaltet, um Raum für umfangreiche Aussagen zu lassen. Es wurden gegebenenfalls Nachfragen gestellt, um das Verständnis zu sichern. Obwohl bereits vorher eine Forschungsfrage für diese Bachelorarbeit formuliert wurde, wird mit dieser Art der Befragung Platz gelassen für weitere Ideen und Ansätze (Aeppli et al., 2016).

Die Befragungen wurden als Leitfadeninterviews durchgeführt. Diese Form der Befragung verwendet einen Leitfaden als teilstandardisiertes Instrument. Der Leitfaden enthält alle zentralen Themen, die für die Datenerhebung wichtig sind, er enthält ebenfalls alle Hauptfragen, sowie weiterführende Detailfragen. Diese Methode gibt den Interviewten den Spielraum selbst zu entscheiden, bei welchen Fragen und Themen sie näher ins Detail gehen wollen. Diese Methode ist besonders geeignet, um Subkulturen zu befragen, wie in diesem Fall TYPO3-Entwickler und -Entwicklerinnen (Scheufele & Engelmann, 2009).

Der Leitfaden enthält die vorher definierten Fragen. Hiermit werden die Vorteile einer wenig strukturierten Befragung durch die fehlenden Antwortmöglichkeiten, mit den Vorteilen einer strukturierten Befragung mit schriftlich formulierten Fragen kombiniert.

Da der Gesprächsverlauf flexibel ist, werden umfangreiche Aussagen der Befragten erfasst. Durch die strukturierten Fragen ist die Durchführung der Interviews relativ einfach, da den Teilnehmenden dieselben Fragen gestellt werden (Aeppli et al., 2016).

Um den Befragten den Raum zu geben frei antworten zu können und um möglichst viele Informationen zu sammeln, wurden die Fragen offen gestellt. Die Befragten konnten auf die offenen Fragen ausführlich eingehen oder sich kurzhalten. Mit dieser Art von Fragen wurde den Befragten ein großer Antwortspielraum ermöglicht. Die offenen Fragen wurden mit einigen geschlossen Fragen kombiniert. Bei den geschlossenen Fragen handelt es sich um vertiefende Fragen zu bestimmten Themen, um eine klare und vergleichbare Antwort zu bekommen (Balzert et al., 2017).

3.1.3. Interviewleitfaden

Während der Vorbereitung für diese Bachelorarbeit und insbesondere während der Literaturrecherche wurden Themen und Fragen, die zum Thema der Software-Code-Qualität allgemein und im Kontext von TYPO3-Extension-Entwicklung passen, gesammelt und festgehalten. Es wurden Teilthemen definiert und mögliche weiterführende Fragen notiert. Anschließend wurden die gesammelten Themen geordnet und aussortiert. Mit dem ordnen nach inhaltlichen Kategorien wurde überprüft, ob der aus der Literaturrecherche entstandene Inhaltsbereich sich in den Kategorien wiederspiegelt. (Aeppli et al., 2016). Der Interview-Leitfaden wurde in vier Hauptkategorien eingeteilt (siehe):

- Hintergrundfragen
- Code-Qualität
- Statische Software-Analyse-Tools
- Empfehlungen und Schlussfolgerungen

Es wurden zu den Fragen zusätzlich Stichworte, Formulierungen und Abbildungen vorbereitet, falls die befragten Personen keine Informationen hätten liefern können oder die Frage unverständlich schien. Das Gespräch entlang des Interviewleitfadens gewährleistet, dass alle für die Beantwortung der Forschungsfrage zentralen Themenbereiche angesprochen werden (Aeppli et al., 2016).

3.1.4. Analyse-Tools

Um eine Vergleichsbasis zwischen den durch die Interviewten genannten Merkmalen und Kriterien für hohe Code-Qualität und den Fähigkeiten von Analyse-Tools herzustellen, wurden Daten zu spezifischen Tools gesammelt. Hierzu wurden die Dokumentationen von ausgewählten und frei verfügbaren Tools analysiert. Dabei wurde nicht nur das am häufigsten erwähnte Tool aus den Interviews berücksichtigt, sondern auch ein weiteres Tool zur besseren Übersicht einbezogen. Dieses wurde bereits vor Beginn der Arbeit ausgewählt, weil es in Vorgesprächen mit Entwicklern und Entwicklerinnen als besonders umfangreich beschrieben wurde.
Das Ziel besteht darin, einen Überblick über die Analyseregeln dieser Tools zu vermitteln. Den erfassten Regeln dieser Tools wurden ebenfalls Kategorien zugeordnet, um eine Vergleichbarkeit zwischen den genannten Aspekten der Befragten und den Möglichkeiten der Analyse-Tools herzustellen.

3.2. Datenanalyse

Nach der Durchführung der Interviews, wurden die digital aufgezeichneten Gespräche in schriftlicher Form transkribiert. Der resultierende Text wurde einer Inhaltsanalyse unterzogen, die es ermöglicht, relevante Merkmale und Aspekte der Interviewinhalte systematisch und objektiv nachvollziehbar zu beschreiben (Scheufele & Engelmann, 2009).
Relevante Merkmale der Software-Code-Qualität im Kontext von TYPO3 wurden aus den Interviews herausgearbeitet. Als relevante Merkmale wurden hierbei Ausprägungen gesammelt, die im Material besonders häufig vorkommen (Mayring, 2015). Dieser Prozess stellt sicher, dass die Schlussfolgerungen aus den Interviews nicht auf emotionalen Äußerungen, sondern auf Erfahrungen der Interviewten basieren.

3.2.1. Vorbereitung der Daten

Vor der eigentlichen Analyse der Interviews wurden diese transkribiert. Hierfür wurden spezifische Transkriptionsregeln festgelegt, die dazu dienten, die spätere Inhaltsanalyse zu erleichtern:

- Wörtliche Wiedergabe: Die Transkription erfolgt so wörtlich wie möglich, um die originalen Aussagen möglichst genau zu erfassen.
- Weglassen von Füllwörtern: Füllworte und Pausen, die keine inhaltliche Bedeutung haben, werden entfernt, um den Fokus auf das Wesentliche zu setzten.
- Erhaltung von Sprache und Umgangssprache: Englische Wörter und Ausdrücke, sowie Umgangssprache - dies schließt auch fehlerhafte Grammatik ein - wurden in der Transkription originalgetreu beibehalten, um die Authentizität zu bewahren und übliche Begriffe in der Programmierung einzubeziehen, die keine oder keine passende Übersetzung in die deutsche Sprache haben.

3.2.2. Kategorienbildung

Für eine systematische Durchführung der Inhaltsanalyse wurden Kategorien gebildet, denen die Inhalte der Interviews zugeordnet wurden. Zunächst wurden Themen und Aspekte, die zum Thema der Software-Code-Qualität passen aus der Literatur-Recherche gesammelt. In einem weiteren Schritt wurden die Interviews gesichtet und es wurden weitere Themen ergänzt. Aus dieser Übersicht an Themen wurden Hauptkategorien gebildet (siehe Anhang 3: Kategorie-Leitfaden – Ebene 1). Diese Hauptkategorien dienen als strukturierendes Gerüst, um die vielfältigen Aspekte der Software-Code-Qualität systematisch zu erfassen und zu ordnen. Jede dieser Hauptkategorien repräsentiert einen bedeutsamen Bereich, der maßgeblich zur Beurteilung der Code-Qualität und zu der Bedeutung von statischen Analyse-Tools beiträgt.

Zu jeder Hauptkategorie wurden anschließend Unterkategorien erstellt, die die Hauptkategorien verfeinert. Die Kategorienbildung orientiert sich an dem von Mayring (2015) beschriebenen Ablauf der Kategorienanwendung.

3.2.3. Analysetechnik

Das Instrument, mit dem diese Analyse durchgeführt wurde, ist der Kodier-Leitfaden. Die analysierten Texte der Interviews dienen als Merkmalsträger, bzw. Analyseeinheiten. In diesem Zusammenhang werden die ermittelten Merkmale dieser Einheiten als Kategorien bezeichnet. Diese Kategorien können als Fragen betrachtet werden, die auf die Inhalte der Interviews angewendet werden, oder als Schubladen, in die vergleichbare Inhalte einsortiert werden. Der zentrale Bestandteil des Kodier-Leitfadens ist das Kategoriensystem, in dem alle Kategorien enthalten sind, die für die Fragestellung dieser Bachelorarbeit wichtig sind (Scheufele & Engelmann, 2009).

Die Kategorienbildung wurde hierbei sowohl induktiv, aus den Daten heraus, als auch deduktiv, auf Basis der vorangegangenen Literatur-Recherche, in Kombination durchgeführt (Aeppli et al., 2016).

Anschließend wurde die eigentliche Analyse durchgeführt, wobei die gesammelten Daten systematisch und methodisch ausgewertet wurden. Hierfür wurden die Daten in mehreren Schritten den Kategorien des Kategorie-Leitfadens zugeordnet. Schließlich wurden die Ergebnisse der Analyse interpretiert und in Bezug auf die Forschungsfrage diskutiert, um relevante Erkenntnisse und Zusammenhänge zu erörtern. Es wurden nur relevante Interviewausschnitte zur Beantwortung der Forschungsfrage benutzt (Aeppli et al., 2016).

Mit Hilfe der qualitativen, strukturierenden Inhaltsanalyse wurden relevante Themen der Software-Code-Qualität im Kontext von TYPO3-Extensions aus den vorhandenen Interview-Texten systematisch extrahiert. (Mayring, 2015).

Diese Art der Analyse ist gut geeignet die Inhalte der Interviews auszuwerten, da sie die Themen und Aspekte im Zusammenhang mit der Erfahrung der interviewten Personen in den Vordergrund stellt und diese Themen in Kategorien gefasst werden. Dieses methodische Vorgehen ermöglicht es, tiefe Einblicke in die Ansichten und Erfahrungen der Interviewteilnehmer und -teilnehmerinnen zu gewinnen.

3.2.4. Durchführung

Die Inhaltsanalyse wurde in mehreren Schritten rekursiv durchgeführt. Es wurden im ersten Schritt alle Interviews untersucht und den Inhalten wurde eine - oder mehrere, wenn keine eindeutige Zuordnung möglich war - der fünf Hauptkategorien zugeordnet. Nach dieser Zuordnung entstanden neue Textabschnitte der jeweiligen Hauptkategorien.

Der Prozess wurde auf die Ergebnisse der aus dieser ersten Kategorienzuteilung entstandenen Textsammlung nochmals mit den jeweilig dazugehörenden Unterkategorien durchgeführt (siehe Anhang 4: Kategorie-Leitfaden – Ebene 2: Kategorie 1 bis Anhang 8: Kategorie-Leitfaden – Ebene 2: Kategorie 5). Daraus entstanden wiederum neue Textabschnitte, die den Unterkategorien zuzuordnen sind. Mithilfe dieses Prozesses entstehen Aussagen der Interviewten, die einer bestimmten Kategorie zugehörig sind, die wiederum analysiert und interpretiert werden.

3.3. Limitation

Im folgenden Abschnitt werden die Limitationen dieser Forschungs-Methode dargelegt, um potentielle Einflüsse auf die Ergebnisse zu berücksichtigen.

3.3.1. Methodische Beschränkung

„Bei der Interpretation qualitativer Daten kann nicht die Objektivität erreicht werden, wie bei der Auswertung quantitativer Daten." (Balzert et al., 2017, S. 270). Es ist möglich, dass persönliche Beiträge oder emotionale Äußerungen in die Antworten der Befragten einflossen, die es erschweren objektive Rückschlüsse zu ziehen. Um dem entgegen zu wirken wurde vorab ein Interviewleitfaden erstellt, mit dessen Hilfe die Interviews durchgeführt wurden (siehe Kapitel 3.1.3 Interviewleitfaden).

Des Weiteren kann es sein, dass die Befragten durch die Anwesenheit der interviewenden Person beeinflusst wurden. Die Ergebnisse sind bei offen gestellten Fragen außerdem weniger vergleichbar als bei geschlossenen Fragen. Durch offene Interviews entstehen weitreichende Interpretationsmöglichkeiten, die sich negativ auf die Objektivität der Auswertung auswirken können (Balzert et al., 2017). Um eine angemessene Vergleichbarkeit und Objektivität in der Auswertung zu gewährleisten, wurden zusätzlich zu den offenen Fragen auch geschlossene, strukturierte und leitende Fragen eingearbeitet.

3.3.2. Befragte Personen

Es wurden ausschließlich Experten und Expertinnen befragt, die über mehrjährige Erfahrung in der Programmierung von TYPO3-Extensions verfügen. Es ist wichtig zu betonen, dass alle Befragten bereits über eine umfangreiche praktische Erfahrung verfügen, ohne Teilnehmer oder Teilnehmerinnen einzubeziehen, die erst kürzlich mit der Programmierung von TYPO3-Extensions begonnen haben oder gerade aus dem Studium kommen. Diese Einschränkung kann dazu führen, dass spezifische Perspektiven oder modernere Ansichten möglicherweise nicht berücksichtigt werden.

Ein weiterer Aspekt der Limitation besteht darin, dass die Befragten aus verschiedenen beruflichen Umfeldern stammen, was unterschiedliche Erfahrungen und Herausforderungen

bei der Arbeit mit TYPO3 mit sich bringt. Um dieser potenziellen Einschränkung entgegenzuwirken, wurde versucht, eine vielfältige Gruppe von Experten und Expertinnen einzubeziehen. Dies umfasste Vertreter und Vertreterinnen aus kleinen bis mittelständischen Unternehmen, Mitarbeitende aus einem Konzernumfeld sowie Personen, die als Freelancer im Kontext von TYPO3 tätig sind. Dadurch wird angestrebt, verschiedene Perspektiven und Erfahrungswerte zu erfassen.

4. Ergebnisse

Die Interviews wurden einer eingehenden Analyse unterzogen, um ein umfassendes Verständnis von Software-Code-Qualität zu erlangen. Im Rahmen dieser Analyse wurden die erfassten Daten sorgfältig kodiert, kategorisiert und interpretiert. Hierbei sollten zentrale Themen erkannt werden, die sowohl hinsichtlich der Code-Qualität als auch der Verwendung statischer Analyse-Tools von Relevanz sind.

4.1. Themen-Analyse

Der Analyseprozess ermöglichte die Extraktion und Identifikation wesentlicher Themen, die im Kontext der Forschungsfrage von besonderer Bedeutung sind. In diesem Abschnitt werden nun diejenigen Ergebnisse genauer untersucht, die einen direkten Beitrag zur Beantwortung der Forschungsfrage leisten.

4.1.1. Code-Qualität

Die Aussagen, die zu dieser Kategorie zusammengefasst wurden zeigen eine Entwicklung der Bedeutung von Software-Code-Qualität hin zu einem Bewusstsein für die Auswirkungen von Code-Qualität auf die verschiedenen Bereiche in der Software Entwicklung. Es werden Aspekte genannt, die die Code-Qualität beeinflusst. Es werden Auswirkungen hoher, sowie niedriger Code-Qualität beschrieben. Nachfolgend werden die Aussagen der Interview-Personen zusammengefasst wiedergegeben.

4.1.1.1. Bedeutung

Die interviewten Personen äußerten verschiedene Aussagen über die Bedeutung der Code-Qualität. Es gab jedoch einige Übereinstimmungen in den Aussagen.

Eine gute Code-Qualität zu schaffen sei anfangs eine große Hürde gewesen, weil eine niedrige Code-Qualität am Anfang nur schwer einzuschätzen sei. Die Bedeutung der Code-Qualität hat bei den befragten Personen zugenommen, am Anfang hätte sie keine besonders große Rolle gespielt, das wäre erst mit der Zeit gekommen. Diese Ansicht wird von den meisten Interviewten geteilt. Einige Mal wird von den Interviewten erwähnt, dass anfangs Code produziert wurde, der funktionierte, ohne auf die Qualität des Codes zu achten. Es wird außerdem erwähnt, dass TYPO3 eine große Einstiegshürde hat.

Aus den Äußerungen der Interviewteilnehmenden geht hervor, dass sich ihre Fähigkeit, Code zu bewerten, im Laufe der Zeit zunehmend verbessert hat. Im Verlauf ihrer beruflichen Entwicklung legten alle Befragten verstärkt Wert auf die Code-Qualität, was von jedem Einzelnen und jeder Einzelnen betont wurde. Dieses gesteigerte Bewusstsein wird auf verschiedene Faktoren zurückgeführt. Zum einen spiele die wachsende Erfahrung eine entscheidende Rolle, die im Laufe der Zeit gewonnen wurde. Ein weiterer wichtiger Aspekt sei

die Kommunikation mit anderen Entwicklern und Entwicklerinnen. Zusätzlich könnte die Beschäftigung mit statischer Codeanalyse dazu beigetragen haben, gezielter auf Qualitätsaspekte zu achten.
Code-Qualität sei ein sehr wichtiger Punkt, da Softwareentwicklung im Team durchgeführt wird. Es sei wichtig, dass der Code für alle Team-Mitglieder verständlich ist. Das wurde ebenfalls von einigen Interviewten betont. In diesem Zusammenhang wurde erwähnt, dass es jedoch auch zu internen Spannungen kommen könne, wenn Kollegen oder Kolleginnen keinen Wert auf gute Code-Qualität legen.
Außerdem trage sie dazu bei, dass Entwickler und Entwicklerinnen sich auf wichtigere Sachen konzentrieren können, wenn eine gute Code-Basis vorhanden ist. Ein weiterer genannter Begriff ist Open Source. Hierbei sei es wichtig eine gute Code-Qualität zu produzieren, weil der produzierte Code auch von anderen genutzt werden soll.
Ein weiterer genannter Aspekt ist, dass Code-Qualität eine immer größere Rolle spielen wird, da die Komplexität steigt.
Code-Qualität könne jedoch auch eine untergeordnete Rolle spielen, wenn etwas programmiert wird, was nur für eine Aktion einmalig gebraucht wird oder das Projekt lediglich für eine kurze Zeitspanne betreut wird. Wenn das Projekt jedoch für einen längeren Zeitraum vorgesehen ist, spiele Code-Qualität eine große Rolle, das haben mehrere Interviewte ausgesagt.
Qualitativ hochwertiger Code wird als Code bezeichnet, der vom zukünftigen Selbst und von Kollegen und Kolleginnen verstanden wird.
Es wurde einige Male erwähnt, dass Code-Qualität insbesondere in der Verantwortung von Entwicklern und Entwicklerinnen liege und dass diese die Zeit und das Budget dafür einfordern sollten. Auf die Frage, welche Bedeutung die Code-Qualität in Projekten hat, haben viele Interviewte ausgesagt, dass es außerhalb der Entwicklungsumgebung, beispielsweise für die Kundschaft oder die Projektorganisation schwer zu fassen sei. Mittelfristige und langfristige Auswirkungen seien schwer zu erkennen für diejenigen, die selbst keinen Code produzieren.

4.1.1.2. Auswirkungen

Die Interviewten äußerten verschiedene Einschätzungen zu den Auswirkungen von Code-Qualität. Die gesammelten Aussagen zu den Auswirkungen sind überwiegend den späteren Bereichen des Softwareentwicklungszyklus[10], wie dem Testing oder dem Wartungsbereich, zugeordnet worden.
Einige genannte Aspekte beschreiben jedoch auch Auswirkungen zum Anfang des Softwareentwicklungszyklus. Diese charakterisieren sich als erleichterte Erweiterbarkeit und

[10] Definition im Glossar

Anpassbarkeit der Software, wenn die Software von Anfang an gut designt würde. Weiterhin könnten frühe Überlegungen zu Testfällen eine positive Auswirkung auf den Entwicklungsprozess haben.
Einige genannte Aspekte wurden als Auswirkungen auf den eigentlichen Entwicklungsprozess beschrieben - besonders häufig wurden Auswirkungen auf die Erweiterbarkeit genannt. Zu viele Abhängigkeiten, eine mangelnde Testabdeckung, sowie veralteter Code wurden als Ursachen genannt, die zur Folge haben, dass viel Zeit investiert werden müsste bei Änderungen, was zu einer Frustration bei Entwicklern und Entwicklerinnen führe, insbesondere, wenn Zeit oder Budget zur Verbesserung fehle.
Zu viele Abhängigkeiten könnten weiterhin negative Auswirkungen auf den Testing-Bereich im Softwareentwicklungszyklus haben. Es wurde außerdem erwähnt, dass eine niedrige Code-Qualität zu einer Überforderung führen könne, da viel Zeit und Energie aufgewendet werden müsste, um den vorhandenen Code überhaupt zu verstehen. Das wiederum könne dazu führen, dass neue Fehler produziert werden.
Es wurden viele Auswirkungen genannt, die in den Instandhaltungsbereich eingeordnet wurden. Hierbei wurde mehrfach betont, dass minderwertiger Code eine lange Fehlersuche zur Folge haben könne. Fehler könnten schneller gefunden werden, wenn der Code eine hohe Verständlichkeit aufweise. Es wurde ebenfalls mehrfach erwähnt, dass Fehler durch eine hohe Code-Qualität reduziert werden könnten.
Weitere Auswirkungen wurden auch für die Kundschaft genannt. Diese könnten häufig nicht nachvollziehen, warum eine Änderung oder Weiterentwicklung viel Zeit und damit höhere Kosten verursache.
Es wurden auch Auswirkungen einer hohen Code-Qualität genannt. Es gibt Extensions, die ein positives Beispiel seien, die inspirierend sein könnten oder aufzeigen, wie bestimmte Probleme gelöst werden. Ebenfalls wurde erwähnt, dass eine hohe Code-Qualität, bei der sich beispielsweise an die Standards gehalten wird oder bei der Klassen- und Methodennamen ordentlich benannt sind, dazu beitragen könnte, dass andere den Code gut verstehen.

4.1.2. Merkmale und Kriterien von Code-Qualität

Die Befragten äußerten ihre Meinungen und Einschätzungen zu verschiedenen Aspekten der Code-Qualität, sowohl allgemein als auch spezifisch für das TYPO3-Umfeld. Dabei gingen sie auf unterschiedliche Kriterien ein, die zur Beurteilung der Code-Qualität herangezogen werden können. Hierbei wurden sowohl positive als auch negative Merkmale beleuchtet, die einen Einfluss auf die Qualität des Codes haben.

4.1.2.1. Hohe Code-Qualität

Im Anschluss an die ersten beiden Kategorienzuordnungen der Interview-Inhalte wurden die erfassten Textabschnitte, der Kategorie „k21 - Merkmale hoher Software-Code-Qualität“ weiter verfeinert und in weitere Unterkategorien unterteilt (siehe Tabelle 1 Merkmale hoher Software-Code-Qualität). Die Erstellung der Kategorien erfolgte auch hier sowohl induktiv, als auch deduktiv.

Tabelle 1 Merkmale hoher Software-Code-Qualität

Code	Kategorie	Häufigkeit	pro Person
k2119	Struktur	6	4
k2123	Einfachheit	5	3
k2121	Slim-Controller	3	2
k2115	Standards	3	3
k2113	Typisierung	3	2
k2125	Dependency Injection	2	2
k2112	Richtlinien	2	2
k216	Testbarkeit	2	2
k218	Lesbarkeit	2	2
k2120	Wiederverwendbarkeit	1	1
k2126	Zuverlässigkeit	1	1
k2111	Naming	1	1
k2110	Formatierung	1	1
k214	Performanz	1	1
k217	Verständlichkeit	1	1
k219	Änderbarkeit & Weiterentwicklung	1	1
K215	Portabilität	0	0
K2114	Dokumentation	0	0
K2118	Wartbarkeit	0	0

Quelle: Eigene Darstellung

Die aufgeführte Häufigkeit in „Tabelle 1 Merkmale hoher Software-Code-Qualität“ ist eine Aufzählung, wie oft die Begriffe genannt wurden. Zu 16 Kategorien wurden von den Befragten Aussagen getätigt. Des Weiteren ist aus der Tabelle ersichtlich, wie viele Interviewte den Begriff erwähnt haben, um einzuarbeiten, dass eine Person einen Begriff mehrfach genannt hat.

Die Analyse der Häufigkeit der genannten Kriterien und Merkmale ergibt, dass der Begriff Struktur mit sechs Nennungen am Häufigsten als Merkmal hoher Code-Qualität genannt wurde. Der Begriff wurde von vier Personen an verschiedenen Stellen genannt. Es wird mehrfach das MVC-Pattern in diesem Kontext hervorgehoben. Dabei wird von den Befragten

betont, dass allein die Betrachtung dieser strukturellen Aufteilung bereits Rückschlüsse auf die Qualität des Codes zulasse, unabhängig von der reinen Ausführbarkeit des Codes. Weiterhin wird unterstrichen, dass eine übersichtlich definierte Ordnerstruktur ein wichtiges Merkmal darstelle, wenn eine Analyse der Code-Qualität vorgenommen wird.

Der Begriff Einfachheit wurde mit fünf Nennungen ebenfalls häufig genannt. Drei Personen nannten diesen Begriff an unterschiedlichen Stellen. Die Einfachheit wird in den Nennungen durch einige Aspekte vermittelt. Es wird darauf hingewiesen, dass der Code nicht übermäßig komplex sein sollte, was in zweifacher Hinsicht explizit als „nicht komplex" erwähnt wird. Dies würde durch die Verwendung kleiner und übersichtlicher Methoden manifestiert. Des Weiteren wird im Kontext der Einfachheit ebenfalls auf die Bedeutung einzelner und klar abgegrenzter Service-Klassen hingewiesen. Ein weiteres zentrales Merkmal der Einfachheit solle darin liegen keine besonders langen oder tief verschachtelten Abschnitte im Code zu verwenden, ebenso wie die Vermeidung aufwändiger if-else-Bedingungen. Die Betonung der Einfachheit ist eng gekoppelt mit Aussagen im Kontext der Struktur des Codes.

Das Slim-Controller-Prinzip, Standards und Typisierung wurden jeweils dreimal genannt. Slim-Controller und Typisierung wurden von zwei Befragten an verschiedenen Stellen im Interview hervorgehoben. Der Begriff Standards wurde jeweils von drei Personen genannt. Besondere Betonung wurde darauf gelegt, dass die Anwendung von kurzen und schlanken Methoden im Controller ein Kennzeichen für hochwertige Code-Qualität darstelle. Diese Einschätzung teilt die Auffassung, dass Controller keine umfangreiche Logik enthalten sollten.

Zudem wurde betont, dass die Befolgung von Standards ein Anzeichen für eine hohe Code-Qualität sein könne. Diese Standards wurden auf unterschiedliche Weisen beschrieben, sowohl als vom TYPO3-Core[11] bereitgestellte Standards, als auch als Vorgaben für die Ordnerstruktur.

In den Interviews wurde auch erwähnt, dass die Programmiersprache PHP in Bezug auf Typisierung Fortschritte gemacht habe. Die Bedeutung der Typsicherheit als Qualitätsmerkmal wurde ebenso betont wie die Empfehlung, dass Funktionsparameter keine Typen haben sollten.

Die Begriffe Dependency Injection (DI), Richtlinien, Testbarkeit und Lesbarkeit wurden jeweils zwei Mal erwähnt. DI wurde als neuer Stand der Dinge, sowie als TYPO3-Core Standard bezeichnet. Richtlinien wurden in Bezug auf die Einhaltung von Code-Guidelines und PHP-Qualitätskriterien genannt. Als weitere wichtige Merkmale wurden eine gute Testbarkeit, sowie Lesbarkeit des Codes genannt. „Gute Lesbarkeit, also ein guter Code spricht zu einem und sagt einem was er tut, braucht keine Erklärung" (M, Zitat aus dem Interview).

Wiederverwendbarkeit, Zuverlässigkeit, Naming, Formatierung, Performanz, sowie Verständlichkeit wurden nur als Merkmale genannt und nicht näher erläutert.

[11] Definition im Glossar

Die Begriffe Fehlerfreiheit, Flexibilität, Portabilität, Dokumentation und Wartbarkeit wurden nicht direkt als Aspekte guter Software-Code-Qualität genannt.

4.1.2.2. Mangelnde Code-Qualität

Ebenso wie bei der Identifizierung der Merkmale hoher Code-Qualität wurden auch für niedrige Code-Qualität entsprechende Merkmale und Kriterien erfasst. Die Kategorien wurden deduktiv erstellt. Dabei wurden separate Unterkategorien erstellt (siehe Tabelle 2 Merkmale niedriger Code-Qualität), jedoch erfolgte bei der Analyse auch ein Abgleich mit den Kategorien der Merkmale hoher Software-Code-Qualität („k21 - Merkmale hoher Software-Code-Qualität“), um festzustellen, ob die genannten Aspekte einer niedrigen Code-Qualität mit dem Nichtvorhandensein von Merkmalen hoher Code-Qualität übereinstimmen. Diese Merkmale werden in den folgenden Tabellen als Negation des entsprechenden Merkmals hoher Code-Qualität mit dem Symbol ~ gekennzeichnet.

Es gibt elf Kategorien, die Merkmale von niedriger Code-Qualität beschreiben (siehe Tabelle 2 Merkmale niedriger Code-Qualität) und zwölf Kategorien wurden mit ihrem Nichtvorhandensein eines Merkmals hoher Code-Qualität als Merkmal niedriger Code-Qualität beschrieben (siehe Tabelle 3 Merkmale niedriger Code-Qualität ~k21).

Tabelle 2 Merkmale niedriger Code-Qualität

Code	Kategorie	Häufigkeit	pro Person
k224	Keine klar abgegrenzten Aufgaben	4	2
k221	Veralteter Code	3	2
k2210	Großer Programmumfang	3	2
k223	Ein- und Ausstiege	2	1
k225	Verwendung von Variablen	1	1
k226	Unklare Rückgabewerte	1	1
k227	Tiefe Vererbung	1	1
k228	Abhängigkeiten	1	1
k229	Externe Pakete	1	1
k2211	Copy & Paste	1	1
k2212	Inkonsistenz im Programmierstil	1	1

Quelle: Eigene Darstellung

Eine fehlende Verständlichkeit des Programmcodes wurde am häufigsten als Merkmal niedriger Code-Qualität genannt. Sechs Mal wurde dieser Aspekt von fünf Interviewten an verschiedenen Stellen genannt. In dieser Kontextualisierung wurde die Bedeutung der Verständlichkeit hervorgehoben, indem sie zweimal als eine Reaktion auf zu große Komplexität im Code erwähnt wurde. Zudem wurden stark verschachtelte if-Bedingungen,

switch-Anweisungen sowie umfangreiche Schleifen als ungünstige Merkmale angeführt. Ein weiterer Punkt betont, dass der Programmcode nur schwer zu erfassen sei, wenn er nicht den etablierten Standards von TYPO3 entspricht.

Tabelle 3 Merkmale niedriger Code-Qualität ~k21

Code	Kategorie	Häufigkeit	pro Person
k217	~Verständlichkeit	6	5
k2111	~Naming	4	2
k2119	~Struktur	3	3
k2121	~Slim-Controller	3	3
k2113	~Typisierung	3	2
k2114	~Dokumentation	3	2
k2115	~Standards	2	2
k218	~Lesbarkeit	1	1
k216	~Testbarkeit	1	1
k2112	~Richtlinien	1	1
k214	~Performanz	1	1
k2120	~Wiederverwendbarkeit	1	1

Quelle: Eigene Darstellung

Weiterhin wurde von je zwei Interviewten jeweils vier Mal erwähnt, dass eine fehlende klare Abgrenzung von Aufgaben in Methoden und Klassen, sowie eine unpassende Benennung von Variablen und Klassen ein Merkmal für niedrige Code-Qualität sei. Die fehlende Abgrenzung von Aufgaben könne dazu führen, dass sie zu umfangreich werden. Auch lange Funktionen mit mehreren Aufgaben oder überladene Methoden und Klassen, die zu viele Themen behandeln, seien verbreitete Schwachpunkte.
Es wird betont, dass veralteter Code schwer zu lesen sei. In diesem Zusammenhang wird häufig der Begriff Legacy Code verwendet. Weitere Aspekte, die erwähnt werden sind ein zu großer Programmumfang und eine fehlende Struktur. Auch das Nichteinhalten des Slim-Controller Prinzips wird einige Male als Merkmal schlechten Codes genannt, sowie eine fehlende Typisierung, eine unvollständige oder abweichende Dokumentation des Codes und das Nichteinhalten von Standards. Weitere Aspekte werden nur einmal genannt, eine schlechte Lesbarkeit, Testbarkeit, die Nichteinhaltung von Richtlinien, eine schlechte Performanz, sowie schlechte Wiederverwendbarkeit.

4.1.3. Erfahrungen und Herausforderungen

Alle befragten Personen weisen eine mehr als zehnjährige Erfahrung in der Entwicklung und Weiterentwicklung von TYPO3-Extensions vor. Nachfolgend werden die Aussagen

zusammengefasst wiedergegeben, in denen die Befragten von ihren Erfahrungen und Herausforderungen bei der Entwicklung berichten.

4.1.3.1. Erfahrungen mit mangelhafter Code-Qualität

Alle Interviewten sagten aus, dass sie bereits Erfahrungen gemacht haben mit Code, der eine minderwertige Qualität hatte. Häufig wurden dabei Projekte erwähnt, die von anderen Agenturen oder Firmen übernommen wurden, die eine niedrige Code-Qualität aufwiesen. Einige Befragte gaben dabei an, dass in bestimmten Umfeldern die Zeit, die Mittel oder das Verständnis fehlten, um den Code zu überarbeiten und zu verbessern. Es wurde ebenfalls von einigen Interviewten erwähnt, dass es motivieren würde den schlechten Code aufzuarbeiten und zu verbessern. Alle Interviewten gaben an, dass sie bereits Zeit investiert haben, um die Code-Qualität zu verbessern.

4.1.3.2. Erfahrungen mit Zeitdruck

Auch mit Zeitdruck in Projekten haben alle Befragten Erfahrungen gesammelt. Es würde immer wieder dazu kommen, selbst wenn die Projektorganisation versucht es zu verhindern. Es würde auch wiederkehrend bei Projektübernahmen dazu kommen, dass die Zeit, die für die Verbesserung von Code benötigt würde, nicht einkalkuliert wird.

Wenn Entwickler oder Entwicklerinnen dennoch an dem vorhandenen, schlechten Code arbeiten, führe das dazu, dass der Zeitdruck wächst und die Gefahr besteht, zusätzlichen schlechten Code zu produzieren. Wenn die Zeit fehlt, könne dies Auswirkungen auf die Code-Qualität haben. Bei einfachen Aufgaben sei das noch kein Problem, bei komplexen Codelösungen könne es jedoch schwierig sein den Code qualitativ hochwertig zu produzieren.

4.1.3.3. Schwierigkeiten

Verschiedene Herausforderungen wurden von den Befragten genannt. Ein signifikanter Punkt ist Legacy Code. Es wird hervorgehoben, dass die Überarbeitung von schlechtem Code zeitaufwendig sei, da eine systematische kontinuierliche Arbeit daran erforderlich wäre. Besonders im Kontext von Legacy Code mangele es oft an der dafür benötigten Zeit. Zusätzlich wird darauf hingewiesen, dass es eine Herausforderung darstelle, den vorgefundenen veralteten Code zu bewerten. Es sei schwer zu entscheiden, ob der Code fehlerhaft ist, vergessen wurde zu aktualisieren oder bewusst nicht mehr benötigt wird und noch nicht entfernt wurde.

Eine weitere häufig genannte Herausforderung ergibt sich im Kontext der Code-Qualität im Agenturumfeld. Innerhalb dieser Umgebung, die auf die Entwicklung von Websites und digitalen Lösungen spezialisiert ist, würden bestimmte zeitliche und finanzielle Beschränkungen auf die Umsetzung von Code-Qualität treffen. Oft würde eine pragmatische

und schnelle Lösung bevorzugt, auch wenn sie nicht immer den höchsten Qualitätsansprüchen genügt. Allerdings würden die Kunden langfristig weiterentwickelbare und robuste Websites erwarten. In diesem Spannungsfeld wird betont, dass Entwickler und Entwicklerinnen den Anspruch auf angemessene Zeit für die Verbesserung des Codes einfordern müssten, um eine nachhaltige Qualität zu gewährleisten. Diese Situation könne mitunter Stress verursachen, da das Gefühl entstehen könnte, dass zu viel Zeit für die Codeverbesserung aufgewendet wird.
Es wird hervorgehoben, dass im Bereich des Webentwicklungsumfeldes das Bewusstsein für die Bedeutung von Code-Qualität bisher noch nicht vollständig etabliert sei. Dies könnte darauf zurückzuführen sein, dass viele Menschen Websites eher als Plattformen für Inhalte und weniger als komplexe Softwareprojekte wahrnehmen. Infolgedessen nehme die Code-Qualität oft eine untergeordnete Rolle ein.
Es werden auch Herausforderungen erwähnt, die spezifisch im Kontext von TYPO3 auftreten. Es wird darauf hingewiesen, dass TYPO3 äußerst offen und umfangreich sei. Dies führe dazu, dass verschiedene Entwickler und Entwicklerinnen unterschiedliche Programmierstile und Lösungsansätze verwenden, die teilweise stark voneinander abweichen würden. Während dies bei kleineren Extensions möglicherweise noch unproblematisch sei, gestalte es sich bei umfangreichen und komplexen Extensions schwieriger, konsistente Programmiermuster aufrechtzuerhalten, wenn unterschiedliche Ansätze verfolgt würden. Des Weiteren wird betont, dass in der TYPO3-Entwicklung oft viel über Konfiguration erreicht würde, was eigentlich durch Programmierung abgedeckt werden könnte. Dieses Vorgehen könne für externe Beobachter wie eine Art Magie erscheinen, da es nicht immer offensichtlich sei, wie bestimmte Funktionalitäten aufgebaut und umgesetzt würden.
Die Entwicklung von pragmatischem Code, der nicht unbedingt eine gute Qualität aufweist, sei für viele erreichbar. Die Herausforderung liege darin, qualitativ hochwertigen Code zu schreiben, der wiederverwendbar ist. Dies wird als zeitaufwändiger Prozess beschrieben. Entwickler und Entwicklerinnen strebten danach, den Code zu verbessern, ähnlich dem Gedanken, einen Campingplatz sauberer zu hinterlassen als man ihn vorgefunden hat. Dabei bestehe die Gefahr von Nebenwirkungen, wenn die Code-Qualität nicht optimal ist. Das Verfassen von klar verständlichem Code sei eine anspruchsvolle Kunst, die von Entwicklern und Entwicklerinnen unterschiedlich umgesetzt wird. Dabei könne es frustrierend sein, Mängel im Code zu erkennen, jedoch keine Möglichkeit zur sofortigen Verbesserung zu haben. Die Identifikation guter Codebeispiele gestalte sich schwierig, da die Gefahr bestehen würde, fremden Code zu übernehmen, ohne ihn vollständig zu verstehen.

4.1.4. Qualitätssicherung

Die genannten Aspekte zur Sicherung der Code-Qualität, die von den Interviewten genannt wurden, werden eingeteilt in Maßnahmen, um Software-Code-Qualität sowohl a priori, als auch a posteriori zu schaffen, bzw. zu erhalten.

4.1.4.1. Konstruktive Qualitätssicherung

Es wurde besonders häufig die Einhaltung von Standards als Maßnahme genannt, um Code-Qualität vor, beziehungsweise während der Entwicklung zu schaffen. Vier befragte Personen haben auf die Frage, wie man Code in einer hohen Qualität produzieren kann geantwortet, dass Standards, die im Team etabliert wurden zu einer guten Code-Qualität beitragen. Weiterhin wurde auch auf die TYPO3-Coding-Guidelines verwiesen.

4.1.4.2. Analytische Qualitätssicherung

Weitere Aspekte, die aus der Frage zur Sicherung der Code-Qualität hervorgegangen sind, können analytischen Maßnahmen zugeordnet werden. Hierbei haben vier von den sechs befragten Personen Aussagen getätigt. In allen Aussagen wird das Schreiben von Tests als Maßnahme für eine gute Code-Qualität angegeben. Dabei wird das Testing als Methode beschrieben, mit der am besten herauszufinden sei, ob der Code gut ist. Wenn er nicht gut sei, könne er schlecht getestet werden. Weiterhin soll diese Methode zusammen mit statischer Codeanalyse helfen eine Sprache dafür zu finden, Code einzuschätzen und in Worte zu fassen, was an diesem Code nicht gut sei.

Statische Codeanalyse wurde ebenfalls von vier Interviewten genannt. Es würde helfen, wenn beim Committen - Commit ist ein Befehl in der Versionsverwaltung, mit dem lokale Änderungen dem gemeinsamen Code hinzugefügt werden - eine Rückmeldung käme, dass es einen Fehler im Code gibt. Es wird empfohlen die Analyse nicht optional, sondern verpflichtend einzurichten.

4.1.5. Tool-Unterstützung

Aus den Aussagen der Interviews geht hervor, dass PHPStan das am häufigsten genannte und verwendete Analyse-Tool ist. Weitere Aussagen belegen, dass das Tool SonarQube besonders gut geeignet sei für statische Analysen.

Diese beiden Tools wurden untersucht, dabei wurden die erfassbaren Merkmale zur Code-Qualität aufgelistet. Diese Merkmale wurden analysiert, dabei wurde jedem Merkmal eine Kategorie zugeordnet, die einem Merkmal hoher oder niedriger Software-Code-Qualität entspricht. Wenn keine eindeutige Zuordnung möglich war, wurde ein entsprechender passender Begriff gewählt (siehe Anhang 9: PHPStan PHP-Analyseregeln und Anhang 9: SonarQube PHP-Analyseregeln).

4.1.5.1. PHPStan

PHPStan ist ein frei verfügbares Analyse-Tool, welches PHP-Code analysieren kann. Es wird über die Eingabeaufforderung aufgerufen. Es können verschiedene Parameter übergeben werden, wie beispielsweise das Regellevel (*Command line usage*, o. D.). Die Analyse-Tiefe wird in 10 Level eingeteilt, wobei Level 0 die niedrigste Stufe ist und Level 9 die strikteste (*Rule levels*, o. D.). Insgesamt trägt PHPStan laut ihrer Website dazu bei, die Code-Qualität durch Durchsetzung einer strengen Typisierung und verschiedener Überprüfungen zur Verbesserung der Zuverlässigkeit und Wartbarkeit der Software zu steigern. Dies spiegelt sich auch in den Ergebnissen der Analyse wieder (siehe). Merkmale, die zu einer besseren Typisierung beitragen sind am häufigsten vertreten (siehe Tabelle 4 PHPStan – Kategorie-Übersicht).

Tabelle 4 PHPStan – Kategorie-Übersicht

Kategorie	Anzahl
Typisierung	6
Unbekannter Code	2
Verwendung von Variablen	2
Ein- und Ausstiege	1
„Magische“ Methoden	1
PHPDocs	1
Dead Code	1

Quelle: Eigene Darstellung

4.1.5.2. SonarQube

Die statische Analyse mit SonarQube kann dabei helfen, Software-Code in einer hohen Qualität zu erstellen und zu pflegen. Das Tool ist für verschieden Programmiersprachen, unter anderem für PHP einsetzbar. Es kann verschieden implementiert werden, beispielsweise als Integration auf DevOps-Plattformen, wie GitHub (Sonar, o. D.).

Die Analyse der zu erfassenden Qualitätsmerkmale ergab eine Vielzahl an Aspekten. Auf der Website werden 271 vorgefertigte Regeln aufgeführt, die bei der statischen Analyse von PHP-Code angewendet werden. Diese Regeln werden in Themen aufgeteilt, 145 Regeln sind dem Bereich Code Smell zugeordnet.

Nach der Analyse dieser Code Smell Regeln, (siehe Anhang 9: SonarQube PHP-Analyseregeln) lässt sich aus der Übersicht (siehe Tabelle 5 SonarQube Regeln Code Smell – Kategorie-Übersicht) die Häufigkeit der Merkmale ableiten.

Die Auswertung dieser Übersicht verdeutlicht, dass die Mehrheit der Regeln dem Konzept der Best Practices entspricht. Diese Regeln geben Empfehlungen, die auf den aktuellen Standards beruhen. Die Nichteinhaltung dieser Regeln würde nicht zwangsläufig Fehler verursachen, jedoch könnten sich potenziell im weiteren Sinne Probleme ergeben.
Besonders häufig konnten Regeln den Kategorien „Verständlichkeit“, „Formatierung“ und „Veralteter Code“ zugeordnet werden. Es finden sich ebenfalls mehrere Nennungen von weiteren Merkmalen, wie „Fehlervermeidung“, „Naming“ oder „Verwendung von Variablen“, sowie weiteren relevanten Merkmalen.

Tabelle 5 SonarQube Regeln Code Smell – Kategorie-Übersicht

Kategorie	Anzahl
Best Practise	30
Verständlichkeit	22
Formatierung	17
Veralteter Code	16
Fehlervermeidung	8
Naming	8
Verwendung von Variablen	7
Großer Programmumfang	5
Testbarkeit	5
Copy & Paste	3
Dokumentation	3
Performanz	3
Struktur	3
Keine klar abgegrenzten Aufgaben	2
Lesbarkeit	2
Unklare Rückgabewerte	2
WordPress	2
Abhängigkeiten	1
Dependency Injection	1
NOSONAR	1
Parse Fehler	1
Tiefe Vererbung	1

Quelle: Eigene Darstellung

4.1.5.3. Erfassbare Merkmale

Die Interviewten wurden im Laufe des Interviews gefragt welche Aspekte ihrer Meinung und ihrer Erfahrung nach von statischen Analyse-Tools erfasst werden können. Es wurden folgende Aspekte genannt:

- Umfang des Codes
- Komplexität des Codes
- Syntaktische Korrektheit
- Größere Blick auf Klassen-Strukturen
- Enge Kopplungen
- Viele Abhängigkeiten
- Formatierung
- Doppelter Code
- Potentielle Fehlerquellen
- Sicherheits-Aspekte
- Typen

Anschließend wurden die Interviewten ebenfalls gefragt, welche Aspekte nicht erfasst werden könnten. Hierbei wurden folgende Punkte genannt:

- Verständlichkeit
- Verschiedene Programmierstile unterscheiden und daraus Probleme erkennen
- Naming
- Architektur
- Komplexe, verschachtelte Funktionen
- Semantik
- Bedeutung hinter dem Code
- Entwurfsmuster
- Laufzeit Ereignisse
- Trennung Konfiguration und Business-Logik
- Variablen in Konstanten auslagern
- Extension-Umfang
- Klassen-Umfang
- Abstraktion

4.1.5.4. Bedeutung

Aus den Antworten der Interviewten ist herauszulesen, dass obwohl Analyse-Tools einen Lerneffekt hätten, einige Befragte dennoch die direkte Kommunikation mit anderen Entwicklern oder Entwicklerinnen als den maßgeblichen Faktor für ihr Lernen betonten. Die

Nutzung statischer Analyse-Tools wurde überwiegend von den Befragten mit einigen positiven Aspekten verbunden.
Sie würden einen Überblick über Risiken und mögliche Maßnahmen zur Verbesserung der Code-Qualität verschaffen. Zusätzlich ermöglichten sie die Quantifizierung technischer Schulden. Durch die Verwendung von statischen Analyse-Tools könnten Entwickler und Entwicklerinnen unabhängige Beurteilungen erhalten, die nicht auf subjektiven Einschätzungen beruhen. Vor allem für Anfänger und Anfängerinnen könnten Analyse-Tools nützlich sein, indem sie auf spezifische Aspekte hinweisen.
Die Verwendung solcher Tools fördere die Teamarbeit, da ein gemeinsames Verständnis entsteht und Teamstandards besser eingehalten werden können. Sie identifizierten nicht nur Fehler, sondern auch potenzielle Sicherheitsrisiken.
Dennoch bleibe es eine Herausforderung für Maschinen, das Verständnis für Code nachzuvollziehen, den ein Mensch erstellt hat. Es könne außerdem frustrierend sein, wenn Analyse-Tools nicht auf eine geeignete Baseline abgestimmt seien und den Code dadurch als vermeintlich sehr minderwertig bewerten.
Insgesamt bewerten die Befragten Analyse-Tools eher positiv, da sie Entwicklern und Entwicklerinnen die Möglichkeit bieten würden sich auf andere, bedeutsamere Aufgaben zu konzentrieren.

4.2. Interpretation

In den Äußerungen der Interviewpartner und -partnerinnen zur Bedeutung von Software-Code-Qualität lassen sich gemeinsame Trends erkennen. Anfangs habe die Schaffung hochwertiger Qualität keine oder eine geringe Bedeutung für die Befragten dargestellt. Erst im Laufe der Jahre und mit einem steigenden Erfahrungswert sei die Bedeutung größer geworden. Diese Ansicht wurde von allen Befragten geteilt. Es wurde genannt, dass es schwierig gewesen sei als Anfänger Code gut einzuschätzen. Außerdem wurde erwähnt, dass TYPO3 eine hohe Einstiegshürde habe. Diese Aussagen legen nahe, dass die Fähigkeit zur Codebewertung mit wachsender Erfahrung steigt und dass Neulinge in der TYPO3-Entwicklung zunächst Zeit benötigen, um qualitativ hochwertigen Code zu erstellen.
Die Kommunikation mit anderen Entwicklern und Entwicklerinnen wurde mehrmals als ein wichtiger Aspekt in diesem Zusammenhang genannt. Da Softwareentwicklung in Teamarbeit stattfinde, wäre eine gute Code-Qualität von entscheidender Bedeutung. Der Code sollte von allen Teammitgliedern gleichermaßen verstanden werden können. Aus diesen Aussagen ist zu entnehmen, dass eine hochwertige Code-Qualität sich positiv auf die Zusammenarbeit der Entwickler und Entwicklerinnen in Software-Projekten auswirken kann. Bedingt durch die wachsende Komplexität und den wachsenden Umfang an Programmcode gewinnt dieser Aspekt immer mehr an Bedeutung. Aus der Literaturrecherche ist hervorgegangen, dass der

Themenbereich der Software-Code-Qualität aus diesen Gründen eine stärkere Fokussierung bekommt.

Die Bedeutung guter Code-Qualität scheint nicht nur in der Planung von Software-Projekten stärker zu werden, es spiegelt sich auch in den Aussagen der Interviewten wieder, dass eine hohe Qualität viele Vorteile bringt. Alle befragten Personen teilen die Auffassung, dass die Einhaltung von Code-Qualität einen wichtigen Aspekt in der Softwareentwicklung darstellen sollte. Es ist herauszulesen, dass die Befragten sich zwar einig darüber sind, dass Maßnahmen zur Einhaltung eines gewissen Qualitätsstandards wichtig sind, jedoch äußerten einige, dass das Verständnis dafür außerhalb des Entwickler- und Entwicklerinnenkreises schwerer zu fassen sei.

Einige Aussagen der Befragten spiegeln wieder, dass Entwickler und Entwicklerinnen die Verantwortung für die Code-Qualität tragen würden und Zeit und Budget dafür einfordern sollten. Aus diesen Aussagen ist zu interpretieren, dass die Befragten das Gefühl haben, dass in der weitreichenderen Projektebene wenig oder kein Verständnis für die Vorteile einer hohen Code-Qualität vorhanden ist. Das könnte sich auf das Stresslevel der Entwickler und Entwicklerinnen auswirken.

Die Aussagen der Befragten zu den Auswirkungen von Software-Code-Qualität konzentrierten sich überwiegend auf den Wartungsbereich im Softwareentwicklungszyklus. Die Meinung, dass eine hohe Code-Qualität nicht nur die Zusammenarbeit im Team fördert, sondern auch Fehler reduziert und Fehler gar nicht erst entstehen lässt, wurde von allen Interviewten geteilt. Eine niedrige Qualität führt dazu, dass viel Zeit in die Fehlersuche investiert werden muss, was Ressourcen verbraucht und somit Kosten verursacht und eine negative Auswirkung auf die Nachhaltigkeit hat.

Ebenso häufig wurden Auswirkungen auf die Erweiterbarkeit von Software genannt. Probleme, wie zu viele Abhängigkeiten, eine unzureichende Testabdeckung und veralteter Code wurden als Ursachen genannt. Diese könnten dazu führen, dass Änderungen an der Software zeitaufwendig sind, was wiederum Frustration bei Entwicklern und Entwicklerinnen verursachen kann, vor allem wenn Zeit und Budget für Verbesserungen begrenzt sind. Auffällig ist, dass fast alle Befragten von Erfahrungen berichteten, dass eine mindere Code-Qualität zu Frustration geführt hat. Sie berichteten davon, dass in Projekten oft die Zeit oder das Budget fehlen würde um eine vorhandene schlechte Codebasis zu verbessern und das dazu führen kann, das weiterhin schlechter Code produziert wird, was sich auf die Erweiterbarkeit der Software auswirken kann und damit letztendlich höhere Kosten verursachen kann. Diese Aussagen bestätigen die Rechercheergebnisse aus der Literatur, dass schlechte Code-Qualität die Kosten eines Software-Projektes erhöhen kann.

Es ist insgesamt ein Trend aus den Interview-Antworten herauszulesen, dass alle Befragten sich einig darüber sind, dass eine niedrige Code-Qualität indirekt negative Auswirkungen auf das gesamte Projekt haben kann.
Es wurden überwiegend negative Auswirkungen von niedriger Code-Qualität genannt, jedoch gab es auch Aussagen zu positiven Auswirkungen von hoher Qualität. Eine hinreichende Codebasis kann dazu beitragen, dass das Projekt an sich eine höhere Qualität aufweist, da sich Entwickler und Entwicklerinnen nicht auf die Verbesserung des Codes oder auf die Fehlersuche konzentrieren müssen.
Alle Befragten waren sich weiterhin darin einig, dass es auch positive Extension-Beispiele gibt, an denen sich Entwickler und Entwicklerinnen orientieren können. Dies wurde von vielen ebenfalls als Maßnahme empfohlen, um zu lernen guten Code zu produzieren. Da TYPO3 auf einer Open-Source-Basis besteht, wäre das ein angebrachtes Mittel. Es wurden Extension-Beispiele angebracht, die als Inspiration dienen und zeigen können, wie bestimmte Probleme gelöst werden. In diesem Kontext wurde jedoch von einigen Befragten genannt, dass es auch problematisch werden kann, wenn sich unerfahrene Entwickler und Entwicklerinnen an vermeintlich guten Extensions orientieren. Aus diesen Aussagen geht hervor, dass TYPO3, mit einer Open-Source-Basis besonders viele Möglichkeiten bietet fremde Code-Teile zu betrachten und als Inspiration zu verwenden. Die von den Befragten genannte Schwierigkeit, dass besonders Anfänger und Anfängerinnen Code noch nicht gut einschätzen können, bietet jedoch ein hohes Potential für weiterhin schlecht produzierten Code.

4.2.1. Verständlicher Code

Die Ansichten der Befragten zu den Merkmalen von Code-Qualität sind teilweise unterschiedlich, es finden sich jedoch Gemeinsamkeiten in den Inhalten. Es wurden Aussagen zu 16 Kategorien getätigt, die Merkmale und Kriterien hoher Code-Qualität beschreiben. Im Gegensatz dazu sind aus den Inhalten der Interviews nur elf Kategorien hervorgegangen, die eine mindere Code-Qualität beschreiben. Die restlichen zwölf Kategorien beschreiben lediglich das Fehlen des entsprechenden Merkmals hoher Code-Qualität. In einem direkten Vergleich der am häufigsten genannten Merkmale für hohe Code-Qualität und den am häufigsten genannten Merkmalen niedriger Code-Qualität, beschrieben durch das Nichtvorhandensein eines Merkmals hoher Code-Qualität ergibt sich ein Vergleich, der die Wichtigkeit einiger genannter Aspekte unterstreicht (siehe Tabelle 6 Merkmale hoher und niedriger Code-Qualität).
Nachdem die Aussagen zur niedrigen Code-Qualität, die einem Fehlen von Merkmalen einer hohen Qualität entsprechen, eingearbeitet wurden, ergeben sich für die Aussagen, welche Merkmale und Kriterien besonders wichtig sind für die Befragten, neue Schwerpunkte (siehe Tabelle 7 Zusammenfassung Merkmale hoher Code-Qualität).

Tabelle 6 Merkmale hoher und niedriger Code-Qualität

Merkmale guter Code-Qualität	Häufigkeit	pro Person	Merkmale schlechter Code-Qualität	Häufigkeit	pro Person
Struktur	6	4	~Verständlichkeit	6	5
Einfachheit	5	3	~Naming	4	2
Slim-Controller	3	2	~Struktur	3	3
Standards	3	3	~Slim-Controller	3	3
Typisierung	3	2	~Typisierung	3	2
Dependency Injection	2	2	~Dokumentation	3	2
Richtlinien	2	2	~Standards	2	2
Testbarkeit	2	2			
Lesbarkeit	2	2			

Quelle: Eigene Darstellung

Weiterhin ist die Struktur das am meisten genannte Merkmal, jedoch ist eine fehlende Verständlichkeit des Codes ein häufig genanntes Merkmal.

Tabelle 7 Zusammenfassung Merkmale hoher Code-Qualität

Merkmale guter Code-Qualität	Häufigkeit	Pro Person
Struktur	9	7
~Verständlichkeit	6	5
Slim-Controller	6	5
Typisierung	6	4
Einfachheit	5	3
Standards	5	5
~Naming	4	2
~Dokumentation	3	2
Dependency Injection	2	2
Richtlinien	2	2
Testbarkeit	2	2
Lesbarkeit	2	2

Quelle: Eigene Darstellung

Es lässt sich aus vielen Aussagen der Befragten zu allen anderen Aspekten, die nicht der Verständlichkeit zugeordnet wurden, ableiten, dass diese im weiteren Sinn zu einem besser verständlichen Code beitragen sollen.
Viele Befragte nannten die Einfachheit von Code als wichtiges Qualitätsmerkmal. Hierbei wird betont, dass der Code weder übermäßig komplex noch unnötig verschachtelt sein sollte. Die Befragten erläutern zahlreiche Aspekte und Beispiele, die dazu beitragen sollen, die

Einfachheit des Codes zu gewährleisten. Diese Betrachtungen können im weiteren Sinn der Verständlichkeit des Codes zugeordnet werden. Obwohl die genannten Aspekte der Kategorie „Einfachheit“ zuzuordnen sind, tragen sie zur besseren Verständlichkeit des Codes bei.
Zudem führen andere Kategorien letztendlich ebenfalls zu dem Ziel der Codeverständlichkeit. Die Äußerungen zur Kategorie „Slim-Controller“ beispielsweise beschreiben Methoden, die darauf abzielen, die Verständlichkeit des Codes zu erhöhen. Hierbei wird empfohlen, kurze und klar strukturierte Methoden zu verfassen. Die Äußerungen, die dieser Kategorie zugeordnet wurden, tauchten gleichermaßen häufig bei den Befragten auf, sowohl in Bezug auf Merkmale von hoher als auch, bei Nichtbeachtung, von niedriger Code-Qualität.
Diese Betonung der Verständlichkeit als Qualitätsmerkmal deckt sich mit den generellen Aussagen der Befragten zur Bedeutung und den Auswirkungen von Code-Qualität. Aus diesen Betrachtungen kristallisiert sich die zentrale Bedeutung der Zusammenarbeit im Team heraus, wobei jeder Entwickler und jede Entwicklerin in der Lage sein sollte, den Code problemlos zu verstehen.
Verständlichkeit wurde als konkreter Begriff zwar nur einmal im Zusammenhang hoher Code-Qualität erwähnt, jedoch deuten viele Aussagen darauf hin, dass Verständlichkeit ein sehr wichtiges Merkmal für die Qualität darstellt. Dies wird dadurch unterstützt, dass die Befragten als häufigstes Merkmal niedriger Qualität eine geringe oder fehlende Verständlichkeit des Codes nannten. Dadurch bekommt dieser Aspekt eine höhere Bedeutung in der Interpretation der Interviewergebnisse.

4.2.1.1. Struktur

Die Aussagen der Kategorie „Struktur“, welche von den Befragten als entscheidendes Kriterium für guten Code betrachtet wird, zielt ebenfalls in einem weiterführenden Blick durch eine beschriebene übersichtliche und logische Anordnung darauf ab, den Code für andere Entwicklerinnen und Entwickler verständlich zu gestalten.
Die in diesem Zusammenhang genannten Konzepte, wie MVC scheinen eine wichtige Rolle bei der Bewertung von Code-Qualität zu spielen. Die Struktur wird jedoch nicht von allen Befragten mit dem MVC-Pattern gleichgesetzt. Es wird häufig ein „gut strukturierter Code“ genannt. Im Interpretationsspielraum dieses Begriffes kann damit auch eine für den Betrachter verständliche und logische Strukturierung der Klassen gemeint sein.
Die Kategorie „Standards“ enthält Aussagen, die eng in Zusammenhang mit der Kategorie „Struktur“ stehen. Die Standards werden von den Befragten teilweise als Struktur, die von TYPO3 vorgesehen ist beschrieben. Andere Befragte gehen nicht näher auf die Beschreibung dieser Standards ein. Gemäß den Aussagen der Befragten wird die Kategorie „Struktur“ dadurch zusätzlich aufgewertet.

Das häufig genannte Merkmal „Keine klar abgegrenzten Aufgaben“ von niedriger Code-Qualität spiegelt sich hier ebenfalls wieder. Die Befragten berichteten häufig davon, dass niedrige Code-Qualität dadurch beschrieben wird, dass Funktionen oder Klassen zu viele Aufgaben erfüllen und eine Abgrenzung nicht zu erkennen ist. Diese Aussagen können der Kategorie „Struktur“ zugeordnet werden und bekräftigen die Aussagen der Befragten zur Struktur als wichtiges Qualitätsmerkmal.

4.2.1.2. Naming

Ein wichtiges und oft genanntes Merkmal von niedriger Code-Qualität ist eine unzureichende Benennung von Variablen, Funktionen oder Klassen. Es wurden keine Aussagen getätigt die beschreiben, dass eine sinnvolle Namensvergabe ein wichtiges Kriterium für eine hohe Code-Qualität ist. Dennoch scheint dieser Aspekt wichtig zu sein, da die Befragten es bei Nichtbeachtung häufig als Kriterium minderer Qualität nannten. Auch diese Kategorie kann in einem umfassenderen Blick der Verständlichkeit von Code zugeschrieben werden. Viele Aussagen der Interviewten beschreiben, dass schlecht gewählte Namen negative Auswirkungen haben.

4.2.2. Interpretation und Analyse-Tools

Vier der befragten Personen gaben an, bereits statische Analyse-Tools zu verwenden, während die anderen beiden Interviewten angaben, vorwiegend nur die Analyse-Funktionen ihrer integrierten Entwicklungsumgebung (IDE) zu nutzen.
Insgesamt bewerten die Befragten, die bereits Analyse-Tools nutzen, die Verwendung dieser positiv und sehen darin eine sinnvolle Unterstützung für die Softwareentwicklung.
Dennoch wurde häufig erwähnt, dass solche Analyse-Tools noch nicht ausgereift genug sind und eine Kommunikation mit anderen Entwicklern und Entwicklerinnen nicht ersetzten können. Aus diesen Aussagen geht hervor, dass die Befragten zwar einen hohen Nutzen in der Verwendung von Analyse-Tools sehen, jedoch andere Methoden der Codeverbesserung nicht ausschließen wollen.
Die Analyse der Möglichkeiten von PHPStan und SonarQube ergibt eine umfassende Sicht darauf, welche Bereiche der Code-Qualität geprüft werden können. Ein Vergleich mit den Ergebnissen, welche Merkmale und Kriterien die Befragten als besonders wichtig erachten, ergibt, dass Tools ein breites Spektrum dieser Ansichten abdecken können. Der große Aspekt der Verständlichkeit, der sich als wichtiges Kriterium für hochwertige Code-Qualität aus den Interviews ergeben hat, kann insbesondere mit SonarQube gut abgedeckt werden.
Die wichtigsten Merkmale für gute Code-Qualität, mit mehr als zwei Nennungen sind:

- Struktur
- ~Verständlichkeit
- Slim-Controller
- Typisierung
- Einfachheit
- Standards
- ~Naming
- ~Dokumentation

Hierbei fallen insbesondere Regeln von SonarQube auf, die eine zu komplexe oder zu tiefe Verschachtelung von Funktionen, Methoden oder allgemeinem Code verhindern sollen. Diese Aspekte wurden auch von Interviewten als wichtige Kriterien der Verständlichkeit genannt. Im Vergleich mit den Merkmalen und Kriterien, die durch die beiden Tools SonarQube und PHPStan erfassbar sind, ergeben sich viele mögliche Aspekte, die diese Tools für die wichtigsten Merkmale der Befragten bereitstellen.

Tabelle 8 Wichtigste Merkmale der Befragten & Abdeckung in Analyse-Tools

Kategorie	Tool	Anzahl
Verständlichkeit	SonarQube	22
Naming	SonarQube	8
Typisierung	PHPStan	6
Großer Programmumfang	SonarQube	5
Dokumentation	SonarQube	3
Struktur	SonarQube	3
PHPDocs	PHPStan	1
Summe		48

Quelle: Eigene Darstellung

Von den 158 (14 Möglichkeiten von PHPStan + 144 Möglichkeiten von SonarQube) erfassten Regeln, die die beiden Analyse-Tools durchlaufen können, können 48 den wichtigsten Kriterien guter Code-Qualität der Befragten zugeordnet werden. Bereits diese Anzahl an Möglichkeiten ist hoch. Es gibt noch viele weitere erfasste Aspekte, die in indirektem Zusammenhang ebenfalls zu einer besseren Verständlichkeit des Codes beitragen.
Die Ergebnisse zeigen, dass Analyse-Tools in der Lage sind Entwickler und Entwicklerinnen darin zu unterstützen guten Software-Code zu produzieren, indem sie den Code analysieren und Vorschläge machen, die den Vorstellungen einer hochwertigen Code-Qualität der Befragten Experten und Expertinnen entsprechen.

4.3. Diskussion

In diesem Abschnitt werden die Hypothesen, die zu Beginn dieser Arbeit erstellt wurden näher betrachtet. Es wird geprüft, ob die erwarteten Ergebnisse eingetreten sind. Das Ziel ist es die Erkenntnisse aus der Literaturrecherche mit den tatsächlichen Ergebnissen zu vergleichen. Zudem werden einige kritische Aspekte der Forschungsarbeit beleuchtet.

4.3.1. Hypothesen-Überprüfung

Die vorliegende Studie hat Einblicke in die Bedeutung von Code-Qualität in der Softwareentwicklung, insbesondere der Entwicklung von TYPO3-Extensions gewährt.

Die Befragten haben die Nutzung von Frameworks in den Interviews nur selten erwähnt. Im Gegensatz dazu wurde die Einhaltung der TYPO3-Standards als eines der häufigsten Merkmale für hohe Code-Qualität genannt. Dies bestätigt die Erwartungen, die Erwähnung von Frameworks entsprach hingegen nicht der erwarteten Häufigkeit.

Die Erkenntnisse aus den Interviews zeigen, dass die Befragten teilweise unterschiedliche Perspektiven auf die Code-Qualität haben. Jedoch zeichnet sich ein deutlicher Trend ab, der darauf hindeutet, dass die Verständlichkeit des Codes einen wesentlichen Bestandteil darstellt, der sich auf vielfältige Weisen in den Aussagen der Befragten widerspiegelt. Clean Code wurde als Begriff zwar nicht häufig erwähnt, jedoch spiegeln sich die Prinzipien definitiv in den Aussagen der Befragten wieder. Besonders die Verständlichkeit des Codes wurde von den Befragten hervorgehoben, ebenso wie eine sinnvolle Namensvergabe und Einfachheit im Sinne von geringer Komplexität. Insgesamt bestätigen die tatsächlichen Ergebnisse die anfänglichen Erwartungen.

Im Rahmen der Untersuchung hat sich weiterhin herausgestellt, dass statische Analyse-Tools in der Lage sind bei der Entwicklung eine hohe Code-Qualität sicherzustellen und ein wichtiges Werkzeug für Entwickler und Entwicklerinnen sind. Es ist jedoch wichtig anzumerken, dass diese Tools nach Meinung der Befragten keine vollständige Lösung bieten können und insbesondere tiefgreifende Bedeutungen und übergreifende Komplexität nicht prüfen können. Mehrere Interviewte betonten, dass die direkte Kommunikation mit anderen Entwicklern und Entwicklerinnen nach wie vor unverzichtbar sei und dass Analyse-Tools keine menschlichen Urteile und Abwägungen ersetzen können. Dieses Ergebnis verdeutlicht, dass während Analyse-Tools eine wertvolle Unterstützung bieten, sie nicht alle Facetten der Code-Qualität abdecken können und daher in Kombination mit anderen Methoden verwendet werden sollten. Die Befragten hatten durchweg unterschiedliche Vorstellungen davon, was Analyse-Tools leisten können und welche Merkmale erkannt werden können. Hier spiegelt sich die Ansicht der Befragten wieder, dass Analyse-Tools noch nicht ausgereift genug sind, um die menschliche Sicht auf Verständlichkeit des Codes zu begreifen. Jedoch können bereits

viele Aspekte guter Code-Qualität sichergestellt werden, was sie zu einem wertvollen und wichtigen Werkzeug in der Softwareentwicklung macht.
Im Rahmen der Untersuchung wurden nur die Standardregeln von PHPStan und SonarQube betrachtet. Insbesondere SonarQube bietet die Möglichkeit über diese standardisierten Analyseregeln hinaus den Software-Code weiter zu untersuchen. Es können eigene Regeln erstellt und in das Tool eingebaut werden, somit sind die Möglichkeiten noch umfangreicher.
Insgesamt bestätigt sich die anfängliche Hypothese, dass aus den Interviews hervorgeht, dass statische Analyse-Tools Entwickler und Entwicklerinnen dabei unterstützen können Code in hochwertiger Qualität zu produzieren.

4.3.2. Kritische Betrachtung

Ein zu erwähnender Aspekt ist, dass die Befragten nur eine begrenzte Stichprobe darstellen. Trotz der mehr als zehnjährigen Erfahrung aller sechs befragten Personen in der Entwicklung und Weiterentwicklung von TYPO3-Extensions, handelt es sich um keine repräsentativ große Gruppe. Daher ist es durchaus möglich, dass die erfassten Meinungen zufällig übereinstimmen und bestimmte Perspektiven nicht ausreichend abgebildet wurden.
Die Fragen zur Code-Qualität wurden teilweise allgemein im Kontext der Software-Code-Qualität gestellt, was bedeutet, dass die Interviewpartner und -partnerinnen möglicherweise nicht immer ausschließlich im Zusammenhang mit der TYPO3-Extension-Programmierung geantwortet haben. Dies spiegelt sich auch in den Interviewinhalten wieder, in denen die Befragten an einigen Stellen über generelle Aspekte der Code-Qualität oder über Frontend-Themen der TYPO3-Extensions sprechen, die nicht im Fokus dieser Forschungsarbeit liegen. Es wurde jedoch darauf geachtet, nur solche Inhalte aus den Interviews zu verwenden, die direkt auf die spezifische Forschungsfrage zutreffen.
Ein weiterer kritischer Faktor ist, dass es im Rahmen dieser Arbeit nicht möglich ist, sämtliche Merkmale und Kriterien zu erfassen, da es zahlreiche Aspekte gibt, die über die genannten Aspekte der Befragten hinaus zur Verbesserung der Code-Qualität beitragen können. In dieser Arbeit wurden die essenziellen Aspekte der aus der Literatur hervorgegangenen Punkte der Qualitätssicherung in Softwareprojekten und die Ergebnisse der Interviewbefragungen betrachtet. Es wurde ein Schwerpunkt auf einige Merkmale gelegt, die von den Befragten benannt wurden. Diese Merkmale könnten auch in einem viel tieferen Umfang analysiert werden, als es in dieser Forschungsarbeit erfolgt ist. Jedoch stand vorrangig im Fokus, die Meinungen der Interviewten zum Thema Code-Qualität einzufangen und zu objektivieren.
Die Ergebnisse verdeutlichen, dass die Code-Qualität für Entwickler und Entwicklerinnen eine große Bedeutung hat und dass sie sich der Auswirkungen schlechter Code-Qualität bewusst sind. Sie kritisierten jedoch, dass die Kundschaft und auch Personen aus der Projektorganisation häufig kein Verständnis für Code-Qualität ausweisen. Ein weiterer

Personenkreis, der mehr in der Organisation von Projekten tätig ist, hätte in die Untersuchung einbezogen werden können, um herauszufinden, welche Bedeutung und welchen Stellenwert Code-Qualität außerhalb des Entwicklungsbereiches hat. Bezieht man sich auf den anwenderbezogenen Ansatz von Krypczyk und Bochkor (2022) in die Interpretation ein, mit der sich die Qualität durch die Sicht der Kundschaft definieren lässt, ergibt sich der wichtige Aspekt, dass eine größere Transparenz der Bedeutung und der Auswirkungen von Code-Qualität hergestellt werden sollte.

4.4. Ausblick

Es ist relevant zu beachten, dass sich die Fragen zur Code-Qualität von TYPO3-Extensions speziell auf den Backend-Bereich der Extensions bezogen. Die Analyse konzentrierte sich auf die PHP-Programmteile der Extensions. Jedoch besteht eine Extension aus weiteren Komponenten. Daher wurden lediglich bestimmte Teile von TYPO3-Extensions analysiert und bewertet. Um die Analyse und Bewertung umfassend abzurunden, wäre es ratsam, auch die übrigen Bestandteile der Extensions in die Betrachtung einzubeziehen.

Die Ergebnisse dieser Studie sind im Kontext der TYPO3-Extension-Programmierung mit PHP angewandt worden, jedoch können die Ergebnisse auch übergreifend auf andere objektorientierte Sprachen angewandt werden. Statische Analyse-Tools gibt es bereits für viele verschiedene Programmiersprachen. Die Forschungsergebnisse können dazu inspirieren andere Analyse-Tools und deren Möglichkeiten zur Sicherstellung der Code-Qualität zu untersuchen. Neben TYPO3 gibt es noch viele weitere CMS, auch über den Open Source Bereich hinaus. Ein Vergleich der Möglichkeiten von anderen Systemen könnte weitere wertvolle Sichtweisen bieten.

Eine hohe Code-Qualität kann positive Auswirkungen auf Entwickler und Entwicklerinnen haben und sie dazu motiviert, ebenfalls qualitativ hochwertigen Code zu schreiben. Die Auswertung der Interviews hat ergeben, dass Entwickler und Entwicklerinnen in einem Software-Projekt, eher dazu neigen, einen hohen Code-Qualitätsstandard beizubehalten und zu erweitern, wenn sie auf einer bereits qualitativ hochwertigen Codebasis aufbauen können. Dieser Aspekt ist insbesondere für Projekte, die unter Zeitdruck durchgeführt werden sehr relevant. Alle Befragten haben bereits Erfahrung mit zeitkritischen Projekten gehabt. Eine hohe Code-Qualität kann dazu beitragen, dass selbst zeitkritische Projekte mit wenig Stress für die Entwickler und Entwicklerinnen durchgeführt werden können. Die direkten Auswirkungen einer hohen Code-Qualität auf die Zufriedenheit der Entwickler und Entwicklerinnen, sowie auf das Zeitmanagement sind Themenbereiche, die über diese Forschungsarbeit hinaus betrachtet werden können.

Auch der Aspekt der statischen Analyse ist ein interessanter und wichtiger Punkt. Viele Befragte sagten aus, dass sie in der Zukunft eine vermehrte Anwendung von Analyse-Tools

sehen und sich vorstellen können, dass zu den statischen Analyse-Möglichkeiten dynamische hinzukommen, die insbesondere durch KI (Künstliche Intelligenz) unterstützt werden.

Literaturverzeichnis

Aeppli, J., Gasser, L., Tettenborn, A. T. & Gutzwiller, E. (2016). *Empirisches wissenschaftliches Arbeiten: Ein Studienbuch für die Bildungswissenschaften*. UTB.

Association for Computing Machinery. (o. D.). https://www.acm.org/

Balzert, H. (2008). *Lehrbuch der Softwaretechnik: Softwaremanagement*. Spektrum Akademischer Verlag.

Balzert, H., Schäfer, C., Schröder, M. & Kern, U. (2017). *Wissenschaftliches Arbeiten: Ethik, Inhalt & Form wiss. Arbeiten, Handwerkszeug, Quellen, Projektmanagement, Präsentation*.

Beneken, G., Hummel, F. & Kucich, M. (2022). *Grundkurs agiles Software-Engineering: Ein Handbuch für Studium und Praxis*. Springer Vieweg.

Boehm, B., Brown, J. R. & Lipow, M. (1976). Quantitative evaluation of software quality. *International Conference on Software Engineering*, 592–605. https://doi.org/10.5555/800253.807736

Bühler, P., Schlaich, P. & Sinner, D. (2019). *Digital Publishing: E-Book – CMS – Apps*. Springer Vieweg.

Cheng, L., Murphy-Hill, E., Canning, M., Jaspan, C., Green, C., Knight, A., Zhang, N. & Kammer, E. (2022). What improves developer productivity at Google? Code quality. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. https://doi.org/10.1145/3540250.3558940

CMScensus - CMScensus. (o. D.). Abgerufen am 23. August 2023, von https://cmscensus.eu/

Command line usage. (o. D.). PHPStan. https://phpstan.org/user-guide/command-line-usage

ExtBase Introduction — TYPO3 explained 12.4 documentation. (2023, 14. August). https://docs.typo3.org/m/typo3/reference-coreapi/12.4/en-us/ExtensionArchitecture/Extbase/Introduction/Index.html

Extension Development — TYPO3 explained 12.4 documentation. (2023, 11. Juli). Abgerufen am 12. Juli 2023, von https://docs.typo3.org/m/typo3/reference-coreapi/12.4/en-us/ExtensionArchitecture/Index.html#extension-development

Frequently asked questions - PHP-FIG. (o. D.). Abgerufen am 6. Juli 2023, von https://www.php-fig.org/faqs/

Hoffmann, D. W. (2013). *Software-Qualität*. Springer Vieweg.

Introduction — TYPO3 explained main documentation. (2023, 20. Juli). https://docs.typo3.org/m/typo3/reference-coreapi/main/en-us/CodingGuidelines/Introduction.html

Kirk, D., Crow, T., Luxton-Reilly, A. & Tempero, E. (2020). *On assuring learning about code quality*. https://doi.org/10.1145/3373165.3373175

Krypczyk, V. & Bochkor, E. (2022). *Handbuch für Softwareentwickler: Das Standardwerk für professionelles Software Engineering* (2. Aufl.).

Ljung, K. & Gonzalez-Huerta, J. (2022). "To clean code or not to clean code" A survey among practitioners. In *Springer eBooks* (S. 298–315). https://doi.org/10.1007/978-3-031-21388-5_21

Martin, R. C. (2009). *Clean code: A Handbook of Agile Software Craftsmanship*. Pearson Education.

Mayring, P. (2015). *Qualitative Inhaltsanalyse: Grundlagen und Techniken*.

PHP Static Code Analysis. (o. D.). https://rules.sonarsource.com/php/

Rule levels. (o. D.). PHPStan. https://phpstan.org/user-guide/rule-levels

Scheufele, B. & Engelmann, I. (2009). *Empirische Kommunikationsforschung*.

Schneider, K. (2012). *Abenteuer Softwarequalität: Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement*.

Sonar. (o. D.). *Code quality tool & secure analysis with SonarQube*. https://www.sonarsource.com/products/sonarqube/

Symfony. (o. D.). *Symfony, high performance PHP framework for web development*. Symfony. https://symfony.com/what-is-symfony

Anhangsverzeichnis

Anhang 1: Interview Leitfaden

Im Rahmen meiner Bachelorarbeit mit dem Thema „Analyse und Bewertung von Software-Code-Qualität von Erweiterungen in Content-Management-Systemen anhand von TYPO3“ möchte ich eine Reihe von Interviews mit Fachexpertinnen und Fachexperten durchführen, die an TYPO3 Extensions programmieren. Das Ziel ist es, relevante Informationen und Einsichten zu erfassen und anhand derer Merkmale und Kriterien guter Code-Qualität zu identifizieren. Es sollen verschiedene Aspekte der Code-Qualität, die Merkmale von gutem und schlechtem Code sowie die Rolle von statischen Software-Analyse-Tools besprochen werden. Die gewonnenen Erkenntnisse und Empfehlungen aus diesen Interviews sollen eine wichtige Grundlage für meine Bachelorarbeit bilden und können Entwicklerinnen und Entwicklern dabei helfen, besseren Code zu schreiben.
Nachfolgend beschreibe ich die grobe Struktur, die als Leitfaden für das Interview dienen soll. Anschließend sind die vorformulierten Fragen gelistet, die in den einzelnen Interviews gestellt werden sollen.

Struktur

1. Einführung und Begrüßung
    a. Kurze Vorstellung
    b. Aufnahmegenehmigung
2. Hintergrundfragen
    a. Erfahrung und Expertise im Bereich der TYPO3-Extensions
    b. Erfahrung mit der Bewertung von Software-Code-Qualität
3. Code-Qualität und TYPO3 Extensions
    a. Diskussion der wichtigsten Merkmale und Kriterien für die Bewertung von Code-Qualität von TYPO3 Extensions
    b. Herausforderungen oder Schwierigkeiten bei der Bewertung der Code-Qualität von TYPO3-Extensions
4. Statische Software-Analyse-Tools und Code-Qualität
    a. Einschätzung zur Verwendung von statischen Analyse-Tools, um den Code von TYPO3 Extensions zu analysieren
    b. Potenzielle Vor- und Nachteile von statischen Analyse-Tools
    c. Nutzungsverhalten statischer Analyse-Tools
5. Empfehlungen und Schlussfolgerungen
    a. Empfehlung für Entwickler und Entwicklerinnen, um von Anfang an guten Code zu schreiben
    b. Abschließende Gedanken oder Anmerkungen, was hat gefehlt?
6. Abschluss
    a. Fragen
    b. Kurzes Feedback zum Interview

Fragen

Hintergrundfragen

1. Wie lange arbeitest du bereits mit TYPO3, insbesondere an der Entwicklung und Weiterentwicklung von Extensions?
2. Als du mit der Entwicklung von TYPO3-Extensions begonnen hast, welche Bedeutung hatte die Code-Qualität für dich?
3. Hat sich diese Bedeutung für dich im Laufe der Zeit verändert?
4. Hast du bereits TYPO3 Extension-Code bewertet? Z. B. in Merge Requests?
    a. Welche Herausforderungen oder Schwierigkeiten sind dir bei der Bewertung der Code-Qualität von TYPO3 Extensions begegnet?
5. Benutzt du in deinem Alltag als Entwickler / Entwicklerin statische Analyse-Tools um die Code-Qualität zu überprüfen? Wenn ja, welche?

Code-Qualität und TYPO3 Extensions

1. Auf welche Bereiche im Software-Lebenszyklus (Beispiele: Testing, Einführung, Wartung/Pflege, Fehlerbehebung) wirkt sich deiner Erfahrung nach Software-Code-Qualität aus?
2. Welche Merkmale oder Kriterien sind aus deiner Erfahrung besonders wichtig, um die Code-Qualität von TYPO3 Extensions zu bewerten?
    a. Wenn es sehr viele Merkmale sind, welche würdest du priorisieren? Nenne z. B. die 3 wichtigsten
3. Hast du schon mal mit fremdem Code gearbeitet, der in deinen Augen von schlechter Qualität war? Wenn ja, was genau hast du an diesem Code als schlecht empfunden?
    a. Würdest du sagen, dass dieser schlechte Code sich auf deine Motivation ausgewirkt hat, weiter an diesem Code zu arbeiten?
    b. Hast du Zeit investiert, um diesen schlechten Code aufzuarbeiten und zu verbessern? Wenn ja, wie viel Zeit hat es dich gekostet?
    c. Gibt es bestimmte Merkmale von schlechtem Code, die dir immer wieder begegnet sind, auch in anderem Code?
4. Hast du bereits die Erfahrung gemacht, dass du in einem Projekt unter großem Zeitdruck TYPO3 Extensions entwickeln oder erweitern musstest?
    a. Wie groß ist die Motivation unter diesem Zeitdruck auf eine gute Code-Qualität zu achten?
    b. Wie würdest du die Wichtigkeit von Code-Qualität in Projekten einordnen?
5. Welche Maßnahmen oder Praktiken können dabei helfen von Anfang an guten Code zu schreiben?

Statische Software-Analyse-Tools und Code-Qualität

1. Welche Aspekte von Code-Qualität können deiner Erfahrung nach statische Analyse-Tools in TYPO3 Extensions erfassen und bewerten?

    a. Würdest du sagen, dass die genannten Aspekte, die für dich eine gute Code-Qualität ausmachen durch so ein statisches Analyse-Tool bewertet werden können?
    b. Gibt es auch Aspekte der Code-Qualität, von denen du denkst, dass sie nicht durch solche Tools zu erfassen sind? Wenn ja, welche?
2. Welche Vor- und Nachteile siehst du bei der Verwendung von statischen Analyse-Tools?
3. Hast du schon einmal erlebt, dass ein statisches Analyse-Tool Verbesserungsvorschläge für deinen Code gemacht hat?
4. Welche Rolle spielen Analyse-Tools bei der der Code-Qualität in TYPO3-Extensions über die Zeit hinweg?

Empfehlungen und Schlussfolgerungen

1. Auf Basis deiner Erfahrungen mit TYPO3 Extension-Programmierung. Welche Empfehlungen würdest du Entwickler / Entwicklerin geben, um die Code-Qualität zu verbessern?
    a. Welche Empfehlung würdest du geben, um von Anfang an guten Code zu schreiben?
2. Welche Rolle spielen deiner Meinung nach statische Analyse-Tools bei der Verbesserung von Code-Qualität? Würdest du sagen, dass diese Tools in der Zukunft eine größere Rolle spielen werden?
3. Fällt dir etwas für zukünftige Forschungen im Bereich der Code-Qualität ein? Gibt es Themen, zu denen du dir mehr Forschung wünschen würdest?

Abschluss

1. Waren die Fragen verständlich?
2. Hattest du das Gefühl, dass sich etwas gedoppelt hat?
3. Fehlt deiner Meinung nach ein Themenbereich oder Aspekte, die ich fragen sollte?

Anhang 2: Transkripte der Interviews

Anhang 2.1: Interview 1 – D

Hintergrundfragen

1. **Wie lange arbeitest du bereits mit TYPO3, insbesondere an der Entwicklung und Weiterentwicklung von Extensions?**

Mein erster Kontakt mit TYPO3 ist ganz früh gewesen, ich glaube mit 3.8 oder sowas, das war aber eher so hobby-technisch und 2008 habe ich dann zum ersten Mal mit Version 4.2 oder 4.4 gearbeitet aber quasi als Integrator. Also in einer bestehen Instanz Dinge versucht rauszukriegen und zu ändern und seit 2009 oder 2010 habe ich angefangen für eine Agentur zu arbeiten als Angestellter, vorher als Einzelkämpfer und da fing dann auch die PHP-Entwicklung an, also Erweiterungen selbst entwickeln. Also schon ziemlich lange, ich könnte dir jetzt nicht sagen wie viele Jahre das sind. 14 - 15 Jahre, sowas die Größenordnung. Seit knapp 9 Jahren bin ich in der jetzigen Agentur und hab da mich quasi spezialisiert auf Backend-Entwicklung. Weniger Integration und mehr, fast nur noch Backend-Entwicklung. Dazu gekommen ist dann noch ein wichtiger Teil bei kontinuierlicher Integration, bei kontinuierlicher Auslieferung, also alles was irgendwie mit DevOps zu tun hat.

2. **Als du mit der Entwicklung von TYPO3-Extensions begonnen hast, welche Bedeutung hatte die Code-Qualität für dich?**

Das ist eine große Hürde gewesen, weil die sehr unterschiedlich ist und es weite Teile des Codes gibt, die eigentlich eine unterirdische Code-Qualität haben, was ich aber als Anfänger nicht so gut einschätzen konnte. Also so eine Anekdote, ich habe irgendwie seit 1998 glaube ich, also wirklich verdammt lange Webseiten gebaut, anfangs halt so HTML-Seiten und hab da ganz viele verschiedene Systeme kennengelernt, Drupal, Joomla, auch Java-Basierte Enterprise Systeme und der erste Kontakt mit PHP zu dieser Zeit, der war auch verheerend, ja ich hatte früher schon Programmiererfahrung in anderen Programmiersprachen und dachte das wird nichts mit PHP. Das ist irgendwie alles Quatsch, das hat sich in der Zeit seitdem vor allen Dingen in den letzten würde ich sagen 5 oder 6 Jahren hat sich PHP meiner Meinung nach zu einer ernst zu nehmenden Programmiersprache entwickelt, also immer weiterentwickelt in Richtung Code-Qualität vor allen Dingen durch das strengere Typisieren von Argumenten, von Rückgabewerten. Speziell TYPO3 finde ich hat, ist halt sehr hat eine sehr inkonsistente Qualität, also es gibt sehr alte Codeteile, die teilweise echt furchtbar sind, schwer zu lesen, schwierig dagegen zu programmieren, weil es lange Zeit keine vernünftigen Interfaces gab und auch die Dokumentation zu wünschen übrigließ. Das sehe ich auch auf dem Weg der Verbesserung, ich habe aber das Gefühl, dass diese Code-Qualität bei der Core-Entwicklung ein echtes Hindernis darstellt, um Tempo aufzunehmen. Ich nehme wahr, dass seit ungefähr Version 6.2 und folgenden Versionen sehr daran gearbeitet wird und sehr viele Verbesserungen gemacht werden. Das Tempo wird aber eben durch die große Menge Legacy Code bestimmt und das hat auch bei manchen Versionen erheblichen Einfluss darauf gehabt wie viel man selbst machen muss und sich selbst erarbeiten musste.

3. **Hat sich diese Bedeutung für dich im Laufe der Zeit verändert?**

Ja also ich habe das Gefühl, dass sich meine Fertigkeiten weiterentwickelt haben, meine Fähigkeiten selbst Code gut einzuschätzen, also Qualität einzuschätzen. Und damit natürlich

auch selbst zu produzieren, das hat sich für mich wesentlich verändert, ich schaue eher, wenn ich ein Paket verwende mir vorher an, welche Code-Qualität das hat, ob eine Dokumentation vorliegt, ob es gut getestet ist, wie es weiterentwickelt wird, also ob die vernünftige Mechanismen haben, um mit Bugs umzugehen, oder Veränderungsmechanismen, also ob sie eine stabile Arbeitsweise pflegen und entscheide dann im Rahmen dessen, ob ich das Paket verwende oder nicht. Für meine eigene Arbeit muss ich sagen, dass das schwankt. Der eigene Anspruch, den kann man nicht immer erfüllen, weil man in einer Agentur halt bestimmten Einschränkungen unterliegt, was die Umsetzung angeht, oft. Das hat glaube ich einen großen Einfluss darauf, wie gut man das machen kann. Was den eigenen Anspruch angeht auf jeden Fall, also ich glaube, dass das eine wesentliche größere Rolle spielt als früher. Also für mich ist eine Lösung, die funktioniert, nicht mehr die Hürde die ich nehmen möchte, sondern eine Lösung, die man gut lesen kann, die man gut weiterentwickeln kann, die stabil ist, die strukturiert ist.

4. **Hast du bereits TYPO3 Extension-Code bewertet? Z. B. in Merge Requests?**

Also einem Merge Request würde ich einen Review geben. Ich würde sagen, dass man beim Arbeiten ständig, wenn man mit Code von Kollegen oder mit eigenem Code konfrontiert ist, den ständig bewertet und bewerten muss. Es gibt doch Gelegenheiten, wo wir das formal tun, wo wir versuchen formale Methoden anzuwenden, um Code-Qualität einzuschätzen. Wenn wir z. B. ein Upgrade machen, haben wir sozusagen einen Mechanismus, wo der Umfang des Codes und die Komplexität und so weiter durch statische Codeanalysen analysiert wird und wo man dann anhand dessen Entscheidungen darüber trifft, wie man bei einem Upgrade damit umgeht. Also z. B., wenn wir eine neue Version einspielen oder selbst Fixes vornehmen oder einen Fork machen oder so. Das ist dann auch systematisch.

   a. **Gibt es bestimmte Merkmale von schlechtem Code, die dir immer wieder begegnet sind?**

Ja, das häufigste Beispiel, das häufigste Problem, das Problem, das man am häufigsten findet, was ich am häufigsten finde und darüber stolpere ist eine zu große Komplexität des Codes, also das heißt, tiefe verschachtelte if-then-Konstruktionen, switches, Schleifen, seltsame Aus- oder Einstiege in irgendwelchen Code. Das sind Dinge, die man sehr häufig beobachtet. Häufig ist auch, dass die Methoden und oder Klassen keine klar abgegrenzte Aufgabe haben und zu viel machen und deswegen dann zu lang werden und dadurch dann eben auch wieder unleserlich oder unsicher. Und was der worst-case ist, wenn in einer Methode Ein- und Aussprünge aus der Methode unüberschaubar sind oder Variablen random verwendet werden. Also das ist so der Klassiker, dass man irgendeine Variable hat, dann ist die Methode mehrere tausend Zeilen lang, die sogenannte Methode und dann wird jede Variable x-Mal wieder verwendet mit jeweils einem unterschiedlichen Typ und man kann dann nicht mehr nachvollziehen zu welchem Zeitpunkt sie mit welchem Wert belegt ist oder so. Das ist ein häufiges Problem. Unklare Rückgabewerte von Methoden und Argumente von Methoden sind auch ein häufiges Problem. Das ist etwas, das PHP sehr lange erlaubt hat, dass man alles Mögliche zurückgeben kann, das ist nicht streng typisiert. Und grundsätzlich, was ich auch im TYPO3-Kontext sehr oft erlebe und was ich für eine schwache Code-Qualität halte, ist, wenn Klassen sehr tiefe Vererbungen haben und sehr viele Abhängigkeiten.

b. **Auf welche Kriterien und Merkmale hast du bei der Bewertung besonders geachtet, besonderen Wert gelegt?**

Es gibt aussagefähige Tests, wie gut ist die Testabdeckung. Das ist ein wichtiges Merkmal, wenn ich fremden Code beurteile. Dann Einfachheit, also im Sinne von nicht Komplex und gute Lesbarkeit, also ein guter Code spricht zu einem und sagt einem was er tut, braucht keine Erklärung. Das ist ein wichtiges Merkmal, weil das ansonsten auch zum Brain-Overload führt, also, wenn du versuchst rauszukriegen, was die Leute wollten oder man selbst erreichen wollte und es nicht in der Struktur des Codes klar ist, dann produziert man viele Fehler.

5. **Benutzt du in deinem Alltag als Entwickler/Entwicklerin statische Analyse-Tools um die Code-Qualität zu überprüfen? Wenn ja, welche?**

Ja, das tue ich. Also ich verwende PHPMessDetector, dann gibt es im PHPUnit die Komplexitätsanalyse, das verwende ich in der Regel und ich habe verschiedene Tools von unterschiedlichen Anwendern ausprobiert, CodeCoff, z. B. war eins womit ich längere Zeit gearbeitet habe. Und am glücklichsten war ich mit SonarQube, was als Dienst für Open Source frei ist, also man es solange benutzten, solange das Repository public ist. Weil das ist eine große Menge an statischer Code-Analyse, also Lines of Code, Meth-Detection, wie auch immer, also eine große Menge an Analysen durchführt und einem einen Überblick gibt über die Risiken und mögliche Aktionen, die man ergreifen kann um das zu verbessern und einen Wert für das technical dept, also die technische Schuld benennt. Damit habe ich an eigenen Extensions viel gearbeitet, mit dem Ergebnis, dass ich meine, dass ich sozusagen lerne den Code besser zu strukturieren. So dass er eben besser weiterentwickelbar ist und auch leichter verständlich ist. Und Testing hat dabei eine große Rolle gespielt, also das ist sozusagen die Methode, mit der man am besten lernt, ob der Code gut ist, weil wenn er nicht gut ist, kann man ihn schlecht testen. Zumindest die Unit-Tests.

Code-Qualität und TYPO3 Extensions

1. **Auf welche Bereiche im Software-Lebenszyklus (Beispiele: Testing, Einführung, Wartung/Pflege, Fehlerbehebung) wirkt sich deiner Erfahrung nach Software-Code-Qualität aus?**

Ja, also definitiv einmal auf die Fehlerbehebung, also Code der schlecht zu identifizieren ist oder schlecht zu verstehen ist, kann man wahnsinnig lange Zeit mit Fehlersuche zu bringen. Und zum zweiten, das kommt allerdings später bei der Wartbarkeit und Erweiterbarkeit also Wartbarkeit im Sinne von ich kann meinen Code ohne großen Impact auf Änderungen von externen Paketen, also dependencies anpassen. Das wäre das Einzige, was ich quasi als Wartbarkeit sehe. Bei der Erweiterbarkeit ist es sozusagen das größte Problem häufig, dass Dinge eng gekoppelt sind und ich sie dann nicht mehr einfach erweitern kann.

2. **Welche Merkmale oder Kriterien sind aus deiner Erfahrung besonders wichtig, um die Code-Qualität von TYPO3 Extensions zu bewerten?**

Komplexität, innere Komplexität im Code selbst, die Anzahl der Abhängigkeiten und die Kopplung an die Abhängigkeiten, d. h. z. B. dass man idealerweise eben nicht von einer konkreten Klasse abhängig ist, sondern von Interfaces und dass man die Menge, die reine Anzahl der Abhängigkeiten in der Klasse möglichst geringhält und Vererbung als Sonderfall von Abhängigkeit, also das wäre meiner Meinung nach die engst mögliche Kopplung, sind sowas wie abstrakte Klassen oder parent Klassen. Was man leider eben auch im TYPO3 Umfeld, also

in dem Extbase Framework auch sehr viel sieht. Das führt dazu, dass man den Code halt schlecht mocken kann, also beim Testen, es erschwert das Testen extrem und dass man bei Änderungen in den Abhängigkeiten relativ große Wahrscheinlichkeiten hat, dass man seinen Code dann auch anpassen muss.

3. **Hast du schon mal mit fremdem Code gearbeitet, der in deinen Augen von schlechter Qualität war? Wenn ja, was genau hast du an diesem Code als schlecht empfunden?**

Ja, leider. Also, das ist Alltag in der Regel ist es entweder so, dass man eigenen oder Code des eigenen Teams vorfindet, der älter ist, der schon lange nicht verändert wurde und aber funktioniert hat. Da ist das größte Problem, was ich häufig treffe, dass eben eine mangelnde Testabdeckung vorliegt, so dass man rätseln muss, was der Code eigentlich sollte und nicht sicher sein kann, ob die Änderungen, die man macht neue Fehler einführen oder irgendwelche Regressionen einführt. Das wird weniger, sag ich mal, da gibt es eine positive Entwicklung, wenn wir Projekte übernehmen und Legacy Code übernehmen, dann ist es besonders heikel, weil dadurch die Aufwände für kleine Änderungen oder kleine Fehlerbehebungen halt extrem wachsen, wenn man den Code nicht versteht und das liegt in der Regel daran, dass er halt nicht gut strukturiert ist, dass keine klugen architektonischen Entscheidungen getroffen worden sind, in dem Sinne von Klassenhierarchien oder eben eben verwendete externe Pakete. Oft ist es so, dass Aufgaben, die man mit einem gut getesteten third-class-pakage lösen könnte, aus Unwissenheit oder Bequemlichkeit, mal eben schnell gemacht werden und dann die Lösung quasi einerseits eine niedrige Umsetzungsqualität hat und andererseits der Komplexität des Problems nicht gerecht wird, weil es nur einen bestimmten Aspekt abdeckt und dann wird es sehr wackelig. Also in der Regel ist es Legacy-Code und was ich da am meisten finde, sind halt extrem große Klassen mit unklaren Aufgaben. Copy and Paste ist ein riesiges Problem, was man immer wieder dort antrifft. Das heißt die Wiederverwertbarkeit von Komponenten ist in der Regel nicht gegeben, sondern es sind dann Code-Abschnitte kopiert, die dann dadurch sehr viele ähnliche Varianten von der gleichen Lösung produziert, die man sehr schlecht wieder vereinheitlichen kann.

   a. **Würdest du sagen, dass dieser schlechte Code sich auf deine Motivation ausgewirkt hat, weiter an diesem Code zu arbeiten?**

Absolut. Also wir haben in unseren Projekten tatsächlich große Konflikte, weil wir in der Regel Projekte mit Legacy Code übernehmen, wenn wir eine Ausschreibung gewinnen oder so. Und der Effekt ist, dass man sich dann mit dem, was andere Leute in einer niedrigen Qualität produziert haben und die haben damit quasi eine Abkürzung genommen, also sie haben sich ein leichtes Leben gemacht und haben eine schnelle Lösung produziert und das bindet wahnsinnig viel Zeit, das trägt große Risiken ins Projekt rein und ist es ist meistens deswegen frustrierend, weil man keine oder geringe Aussichten hat die Codebasis grundlegend zu verbessern und man muss es halt betreiben, so lange wie es irgendwie geht. Das zieht sich teilweise über Jahre hin und das empfinde ich als verschwendetet Lebenszeit und ich bin persönlich sauer darüber, dass ich mit sowas mich beschäftigen muss. Es gibt aber durchaus auch positive Beispiele, man darf das nicht vergessen, es gibt auch immer wieder packages, wo man sagt, das sieht richtig gut aus, das ist eine gute Lösung, wo man dann Inspiration bekommt oder lernt, wie man bestimmte Probleme lösen kann.

b. Hast du Zeit investiert, um diesen schlechten Code aufzuarbeiten und zu verbessern? Wenn ja, wie viel Zeit hat es dich gekostet? (Ungefähre Angabe in Relation zur eigentlichen Aufgabe)

Ja, ich versuche das natürlich immer wieder. In dem Prinzip, ich hinterlasse den Code besser, als ich ihn vorgefunden habe. Die Möglichkeiten sind oft eingeschränkt, weil wenn es nicht getestet ist, weiß man nicht was man kaputt macht, oft würde sich der Aufwand über sehr lange Zeit hinziehen, also man müsste systematisch daran arbeiten die Code-Qualität zu verbessern, eine Architektur reinzubringen und die Zeit steht gerade für Legacy Code oft nicht zur Verfügung oder man meint, dass sie nicht zur Verfügung stünde. Der Effekt ist dann, dass man mehr Aufwand in den alten Code setzt, als sich mit neuem zu beschäftigen, man möchte ja gerne coole neue Sachen machen und möchte gerne guten Code produzieren. Es ist ein bisschen schwierig, dieser Begriff guter Code, schlechter Code, ich finde, dass wenn ich zurückgucke auf meine eigene Entwicklung, habe ich dort eine steile Lernkurve hinter mir, man guckt als Entwickler erst mal irgendwie da drauf und das Mist, das ist Kacke, das hätte ich besser gekonnt. Und es fehlen einem am Anfang meistens die Erfahrungen und die Fähigkeiten, um das belegen zu können und sagen zu können woran liegt das. Deswegen haben mir statische Code-Analysen und Tests, Unit-Tests, gute Unit-Tests sehr geholfen da eine Sprache für zu finden und nicht bei der persönlichen Einschätzung das ist Mist, man tendiert dazu alles was andere geschrieben haben, was man nicht versteht erst mal als schlecht zu bezeichnen, das ist aber nicht hilfreich, weil man bei der Codeproduktion ja auch eine Form von Kommunikation ist. Also man kommuniziert mit seinem eigenen früheren Selbst, man kommuniziert mit Kolleginnen und Kollegen oder eben mit Leuten, die früher irgendwas produziert haben. Und sich da Maßstäbe oder Bewertungskriterien zu erarbeiten mit denen man das quasi objektivieren kann, ok objektivieren ist schwierig, mit denen man das messbar machen kann, das finde ich sehr wichtig und das ist ein schwieriger Prozess, wo ich selbst irgendwie nicht viel Anleitung von anderen bekommen habe, sondern eben eher an dem Code von anderen gelernt habe und an Try and Error, also es gibt packages, die habe ich 3 Mal umgeschrieben, weil ich das Gefühl hatte, ok jetzt weiß ich es besser, jetzt kann ich es nochmal angehen und kriege es in eine stabilere oder lesbarere Form. Also ich finde das sehr schwierig, weil jeder Programmierer, dem man sagt, dein Code ist Mist, sich beleidigt fühlt, zu Recht. Man muss lernen, dass in Begriffe zu packen oder das auch in ein Kategorien-System zu bringen, Bewertungsmaßstäbe, die man selbst akzeptieren kann und die anderen auch akzeptieren können.

4. Hast du bereits die Erfahrung gemacht, dass du in einem Projekt unter großem Zeitdruck TYPO3 Extensions entwickeln oder erweitern musstest?

Ja, ständig. Das große Problem ist, dass man mit funktionierendem Code, also funktionierender Code im Gegensatz zu gutem Code, also etwas was funktioniert hinzubekommen, polemisch ausgedrückt kann jeder Trottel, also so lange rumprobieren, bis das rauskommt was man möchte für den konkreten Fall. Das ist einfach keine Kunst aber Code zu schreiben, der den Anforderungen genügt, also guten Code, der gut lesbar ist, gut wartbar ist, gut strukturiert, gut erweiterbar, das ist ein sehr schwieriges Unterfangen. Das fordert, dass man iterativ arbeitet, dass man sich in das Problem immer tiefer einarbeitet und, dass man den Mut hat Umwege zu gehen und auch Fehler selbst zu erkennen und einzugestehen und die zu beheben. Und das ist erst mal zeitaufwendig. Das führt aber am Ende dazu, idealerweise, dass man wieder verwertbare Pakete hat. Was ich im Agentur-Umfeld oft erlebe ist, dass man quasi einen Schnellschuss, das löst mein Problem jetzt sofort bevorzugt und der Aufwand, die Mühe wieder verwertbare Pakete zu entwickeln, der ist ungleich höher. Aber nur darein lohnt es sich

eigentlich zu investieren. Weil das kann ich dann auch langfristig verwenden, also das Konfliktfeld, also ich möchte ein schnelles Ergebnis haben, die Kunden erwarten aber auch implizit dann, dass ihre Seiten, also unsere Lösungen lange Zeit weiter entwickelbar sind und das kommt nicht von selbst, dafür muss man quasi sich immer ein breites Kreuz machen und sagen ich mach das jetzt. Ich finde, dass das im Alltag zu einem persönlichen Problem wird, wenn man sich nicht mit Kollegen zusammentut und wenn man nicht lernt gemeinsam drüber zu sprechen, Reviews zu machen und sich auszutauschen. Und deswegen das einem dann Stress macht, wenn man das Gefühl hat, ich dürfte damit eigentlich nicht so lange zubringen.

a. Wie groß ist die Motivation unter diesem Zeitdruck auf eine gute Code-Qualität zu achten?

Ja, klar. Also man sieht ja, dass Leute je nachdem welche Vorlieben sie haben oder welche Stärken und Schwächen sie haben unterschiedlich damit umgehen und dass sieht man halt auch immer wieder bei Kollegen und bei mir selbst, dass man da manchmal sagt ach scheiß drauf ich will jetzt, dass es funktioniert und gehe nicht die Extra-Meile, um noch einen guten Test zu schreiben oder sowas.

b. Wie würdest du die Wichtigkeit von Code-Qualität in Projekten ein kategorisieren?

Das kommt extrem auf die Perspektive an. Weil für diejenigen, die selbst keinen Code produzieren, ist es ein abstraktes Thema, bei dem es ihnen nicht klar ist, warum es wichtig sein sollte. Also die mittelfristigen und langfristigen Auswirkungen sind schwer zu erkennen. Und wenn man auf der Ebene der Kollegen guckt, hängt es extrem von der Erfahrung ab und es ist auch eine Typ-Frage, wie wichtig das einem ist. Es gibt Leute, die sagen es ist mir egal, wie gut das ist, Hauptsache ich kann hier ein paar Aufgaben wegrocken. Da entsteht quasi auch so eine interne Spannung. Der eine hat es halt schneller fertig und wenn man dann aber reinguckt graust es. Es gibt aber auch natürlich das andere Extrem, dass man Lösungen produziert, die dem konkreten Problem nicht angemessen sind. Ich würde z. B. entscheiden, ob ich, wenn ich jetzt einen Wizard schreibe, um ein Upgrade einmal auszuführen, von dem ich weiß, dass ich dieses konkrete Problem ziemlich sicher nicht noch einmal haben werde, dann spielt für mich Code-Qualität eine untergeordnete Rolle, wenn ich eine Erweiterung entwickele, von der ich annehme, dass sie für eine längere Zeit gebraucht wird in mehreren Projekten eingesetzt wird, dann spielt es natürlich eine größere Rolle.

5. Welche Maßnahmen oder Praktiken können dabei helfen von Anfang an guten Code zu schreiben?

Code-Reviews und Pair-Programming sind gute Methoden, um einen Wissenstransfer zu ermöglichen und vor allen Dingen ermöglichen, dass man lernt über Code zu sprechen. Das sehe ich, dass das einem am Anfang sehr schwer fällt. Ich finde, dass der goldene Weg zu gutem Code sind Tests. Weil wenn man einen Test versucht zu schreiben und die Klasse ist zu komplex oder die Verantwortung der Klasse ist nicht eindeutig bestimmt, dann kann man keine guten Unit-Tests schreiben. Das ist aber schwer zu lernen, also gute Tests zu schreiben, rauszukriegen, was sind Tests, die mir eine gute Aussage über meine Klasse geben und die mir helfen strukturierter zu arbeiten und strukturierter einen Code zu entwickeln, das ist für sich schon eine große Hürde. Also ich glaube nicht, dass die Beispiele, die verfügbar sind, also die Lernbeispiele, die sind dafür meistens nicht gut geeignet. Es gibt so ein paar Methoden, die das fördern, also z. B. Test-Driven-Development, da gibt es verschiedene Schulungen, um die zu

verstehen, da muss man sich schon eine ganze Zeit mit auseinandersetzten. Aber das würde ich als den Gold-Standard sehen: gute Tests schreiben lernen. Was ich auch glaube ist, dass es oft eine Überforderung gibt, also gerade, wenn man anfängt, fällt es einem schwer die Größe oder die Komplexität eines Problems gut einzuschätzen, das heißt man übernimmt sich quasi und produziert damit dann Dinge, die nicht gut weiterentwickelbar sind oder wartbar sind. Und was tatsächlich helfen würde wäre eine enge Zusammenarbeit mit erfahrenen Kollegen, die klarmachen: das ist eine Klasse, ich möchte, dass du die schreibst. Ich möchte, dass deine Klasse dieses Interface implementiert und du hast nichts Anderes, du schreibst nicht in 95 anderen Klassen rum, sondern das ist dann dein Problem. Also sozusagen kleine Probleme finden. Das gut eingrenzen und gute Interfaces beschreiben und dann gemeinsam den Code durchgehen und dann gucken, was ist daran gut, was ist daran nicht gut. Das wäre meiner Meinung nach eine gute Herangehensweise.

Statische Software-Analyse-Tools und Code-Qualität

1. **Welche Aspekte von Code-Qualität können deiner Erfahrung nach statische Analyse-Tools in TYPO3 Extensions erfassen und bewerten?**

Ganz banal erst mal der Umfang des Codes. Das ist nicht uninteressant. Viel Code, viel Risiko. Zum zweiten die Komplexität des Codes kann mitunter erfasst werden. Syntaktische Korrektheit, das ist vergleichsweise Easy. Schwieriger ist, das ist aber auch glaube ich drin ist der größere Blick auf die Klassen-Strukturen. Also Klassen, die viel verwendet werden oder eben enge Kopplungen oder viele Abhängigkeiten, das kann man bei der statischen Code-Analyse auch gut erkennen.

   a. **Gibt es auch Aspekte der Code-Qualität, von denen du denkst, dass sie nicht durch solche Tools zu erfassen sind? Wenn ja, welche?**

Ja, Verständlichkeit. Da gibt es zwar auch Metriken, mit denen man Verständlichkeit einschätzen kann. Aber ob das tatsächlich dann so ist, also ob die Metrik dann stimmt oder nicht hängt extrem davon ab in welchem Umfeld der Code gerade ist. Ich glaube, dass das schwierigste ist sprechenden Code zu schreiben. Also Code, der wenn man den anguckt man sagt ah das ist easy, das verstehe ich. Das ist eine extreme Kunst und da gibt es sehr unterschiedliche Typen von Entwicklern. Ich hatte jetzt eine Kollegin in der Firma, die hat extrem kurzen Code geschrieben, der trotzdem verständlich war. Also sehr lakonisch. Das ist überhaupt nicht mein Stil aber ich fand das beeindruckend, dass sie das konnte. Und es gibt Leute, die sind sehr geschwätzig beim Programmieren, also man hat das Gefühl das ist alles irgendwie viel zu ausführlich, man wünscht es sich kürzer oder es gibt dann Leute, die dazu neigen „tricky" Code zu schreiben, der schwer zu interpretieren ist. „du hast jetzt eine Konstruktion geschrieben, die niemand versteht, wie wäre es denn mit einem einfachen „if" gewesen". Das sind Dinge, die sind schwierig durch statische Code-Analysen herauszubekommen. Die sind aber oft entscheidend, weil man sonst Missverständnisse produziert.

2. **Welche Vor- und Nachteile siehst du bei der Verwendung von statischen Analyse-Tools?**

Also Vorteil definitiv, dass man eine externe Referenz bekommt, dass man nicht auf persönliche Einschätzung angewiesen ist, sondern eine Basis bekommt, um gemeinsam über Code-Qualität zu sprechen. Ich finde es, zumindest in unserem Umfeld noch relativ schwer die gut einzubauen,

also die in den Projekten gut zu verwenden, weil man hat entweder Tools, die einfach sind, z. B. Lines of Code. Kann man relativ easy machen, kostet als Einzelaufwand nicht so viel, ist aber auch nicht so wahnsinnig aussagekräftig und dann muss man die Daten zusammenführen und sozusagen die Bewertung wo anders machen oder eben Tools wie SonarQube, wenn man die einmal eingerichtet hat, hat man sehr sehr viel aber die Interpretation der Ergebnisse fordert auch Auseinandersetzung mit dem Tool. Es gibt auch praktische Probleme, wo werden die Tests ausgeführt, lokal oder in einer Pipeline, habe ich einen externen Dienst, dann muss ich den irgendwie einrichten, anbinden, das pflegen, vielleicht noch Geld dafür bezahlen. Das sind typische Hindernisse.

3. **Hast du schon einmal erlebt, dass ein statisches Analyse-Tool Verbesserungsvorschläge für deinen Code gemacht hat?**

Ja, permanent. Da habe ich sehr viel davon gelernt, ja. Oft ist es bei diesen ausgefeilten Tools, wie SonarQube so, dass man erst mal davorsteht, hat eine unglaubliche Menge an Meldungen, an Issues und die zu interpretieren und sich damit auseinander zu setzten kostet viel Zeit. Also man muss dann einfach ausprobieren, lohnt sich das oder kann ich es ignorieren und da ist man am Anfang unsicher. Aber das ist definitiv etwas, von dem ich viel lerne und was ich auch immer empfehlen würde. Also es ist besser als dein Bauchgefühl.

Empfehlungen und Schlussfolgerungen

1. **Auf Basis deiner Erfahrungen mit TYPO3 Extension-Programmierung. Welche Empfehlungen würdest du Entwickler / Entwicklerin geben, um die Code-Qualität zu verbessern?**

Lerne Tests schreiben, lern gute Tests schreiben. Ich gehe mal anders herum ran, ich finde es Schwierig, wenn man weniger Erfahrung hat, herauszufinden, was sind gute Code-Beispiele. Es gibt im Core oder auch in Extensions von anderen Entwickler / Entwicklerin gibt es gutes und schlechtes und sich daran zu orientieren ist sehr sehr riskant. Wenn man da sagt „hey ich mach das so wie das in der und der Extension ist“, dann ist man vielleicht bei einem schnellen Ergebnis und merkt dann ein oder zwei Jahre später, dass das keine gute Entscheidung war, weil man eine schlechte Architektur-Entscheidung getroffen hat oder etwas übernommen hat, was man nicht gut verstanden hat oder so. Und das ist aber auch ein Vorgehen, es ist legitim zu sagen „ich orientiere mich an dem und versuche das zu kopieren“. Es ist auch nicht so leicht Kollegen zu finden, von denen man sagt „dessen Urteil kann ich gut vertrauen“, weil das setzt voraus, dass man erstmal lernt über Code zu sprechen und auch dem Kollegen zu wiedersprechen, sich traut. Also meine Erfahrung mit TYPO3 war, ich habe mit Extension-Entwicklung relativ naiv angefangen und ich habe sozusagen dem vertraut, was ich über extbase gelesen habe und habe erst nach längerer Auseinandersetzung damit eine eigene Meinung dazu gefunden, auch eine kritische Position und das ist nicht leicht das in TYPO3 rauszukriegen. Da sind andere Frameworks oder Projekte sind leichter, es ist leichter gute Beispiele zu finden und da ist die Developer Expierience auch wesentlich besser, ob es jetzt im PHP-Umfeld ist oder in anderen Programmiersprachen. Das war auch so ein Punkt, ein großes Hindernis bei der Einführung von Tests ist, dass wir unsere ganze Umgebung aufbauen müssen, also wir müssen einen Webserver haben, wir müssen die IDE einrichten, wir müssen mit einem Remote-System irgendwie Code austauschen, brauchen Daten für die lokale Entwicklung, das ist eine große Komplexität im Setup, was einen sehr daran hindern kann überhaupt erstmal zum Entwickeln zu kommen. Und die Einführung von Tests ist dann je nach Test-Typ, ob jetzt Unit-Test oder funktionaler Test in diesem Umfeld auch schwierig. Die Tests dann nicht nur

lokal auszuführen, nicht einfach nur an seiner Kommando-Zeile auf dem Host, sondern vielleicht in einer Docker-Instanz, in der man eine reproduzierbare Umgebung hat, also das ist glaube ich ein sehr großes Hindernis, eine Umgebung reproduzierbar machen, was man als Anfänger auch total unterschätzt. Da kann es dann unterschiedliche Projekte mit unterschiedlichen Anforderungen geben, da kann man dann plötzlich ins Schleudern kommen. Und dann sind in den letzten Jahren ganz gute Lösungen entstanden, wie DDEV z. B. Das dann zu übertragen in ein Test-Setup, das dann automatisiert abläuft und dann auch noch automatisiert irgendwo deployed wird oder irgendwelche Reports produziert, das ist eine ganz schöne Herausforderung und das ist etwas, da ist das Umfeld Webentwicklung PHP einfach ein großes Hindernis. Das ist in anderen Webprojekten einfacher. Ich erinnere mich an meine erste Begegnung mit X-Code, da lädst du dir das runter, startest das und kriegst das alles frei Haus geliefert und es funktioniert. Das ist in unserem Umfeld nicht so selbstverständlich und ich finde ein wichtiger Fortschritt, den wir bei uns in der Firma gemacht haben, das war auch ein großer Kampfpunkt, das wir eben die Entwicklungsumgebung vereinheitlicht haben, sodass ich mir ein Projekt von einem Kollegen in kürzester Zeit aufsetzen kann. Es ist eine Voraussetzung dafür, dass man sowas wie Tests überhaupt integrieren kann. Das heißt als weiteren Rat an Entwickler / Entwicklerin würde ich sagen Entwicklungsumgebung und Test-Setup ist irgendwie extrem wichtig. Und das ist aber etwas, was man sich schwierig zusammensuchen muss.

2. **Welche Rolle spielen deiner Meinung nach statische Analyse-Tools bei der Verbesserung von Code-Qualität? Würdest du sagen, dass diese Tools in der Zukunft eine größere Rolle spielen werden?**

Ich hoffe das. Das ist die Frage, von welchem Umfeld du ausgehst. Ich glaube, dass man im Webumfeld bis zu einer bestimmten Projektgröße noch gut damit durchkommt ohne Analyse-Tools zu arbeiten. Ab einer bestimmten Komplexität und Anforderung von Kunden für eben Weiterentwickelbarkeit, Wartbarkeit ist es eigentlich nicht drin.

3. **Fällt dir etwas für zukünftige Forschungen im Bereich der Code-Qualität ein? Gibt es Themen, zu denen du dir mehr Forschung wünschen würdest?**

Ich glaube, dass das Wissen in der Softwareindustrie im Allgemeinen vorhanden ist, dass aber im Webumfeld das nicht in allen Bereichen angekommen ist. Oft wird das Produkt nicht als Software wahrgenommen, sondern als Webseite, als Inhalt, als Content-Management und das nur eine untergeordnete Rolle spielt. Ich glaube dort, wo gezielt Webanwendungen programmiert werden und ein Bewusstsein dafür besteht, dass man Programme entwickelt, dass man programmiert und dass damit Komplexität einhergeht, da gibt es das Bewusstsein auch schon. Ich kann mir vorstellen, dass sinnvoll wäre, vielleicht weiß ich aber auch nicht genug darüber, nicht nur statische, sondern auch dynamische Code-Analysen durchzuführen.

Abschluss

1. **Fehlt deiner Meinung nach ein Themenbereich oder Aspekte, die ich fragen sollte?**

Ich bin ein großer Fan davon Metriken zu finden und Metriken systematisch zu erfassen und Zeit rein zu erfassen, weil ich glaube, dass am Ende nicht eine absolute Qualität eine Rolle spielt, sondern Tendenzen, also wenn man feststellt, dass in einem Projekt die Qualität degradiert, das ist ein Warnzeichen und dafür brauchst du Daten um das einschätzen zu können oder umgekehrt möchtest du wahrnehmen „meine Entwicklung hat einen positiven

Impact auf die Code-Qualität“ und da die richtigen Metriken zu finden ist das vorallem ein kommunikativer Prozess, da geht es darum, was sind die Anforderungen des Projekts oder des Kunden. Die tatsächlichen Anforderungen, also nicht das, was der Entwickler meint, dass es notwendig wäre, sondern was der Kunde wirklich braucht rauszukriegen und dafür Metriken in Zeit reinzufassen und zu beobachten. Das finde ich einen sehr wichtigen Punkt. Das finde ich wichtiger, die Entwicklung von Code-Qualität, als den aktuellen Stand. Damit trifft beim Einführen von Code-Analysen oft ein frustrierender Effekt ein, weil man die Tools nicht auf eine bestimmte Baseline abstimmt und sagt das erfassen wir erst mal so. Wir erfassen vielleicht auch erstmal nicht alle Metriken, die uns interessieren könnten und schaffen dann einen guten Weg, um das zu verbessern, sondern man schmeißt erst mal drauf und ist dann erschrocken und frustriert darüber, wie vermeintlich schlecht der Code ist. Das ist ein Thema, das ich für wichtig erachte, wenn man über Code-Qualität und statische Analysen spricht, dass man bei der Einführung schrittweise vorgeht und die Akzeptanz bei den Entwickler / Entwicklerin hoch hält und ihnen das Erlebnis gibt, dass es hilft, dass es kein Selbstzweck ist.

Anhang 2.2: Interview 2 - A

Hintergrundfragen

1. **Wie lange arbeitest du bereits mit TYPO3, insbesondere an der Entwicklung und Weiterentwicklung von Extensions?**

Ja, das ist eine gute Frage. Also die bekannteste Extension von mir heißt PowerMail und die ist auch schon sehr alt und ich schaue gerade in Terra nach, Terra ist das TYPO3 Extension Repository. Da war der erste Upload in 2008, im Januar und das war nicht meine erste Extension, also ich würde sagen seit 2005/2006, so etwas in dem Dreh.

2. **Als du mit der Entwicklung von TYPO3-Extensions begonnen hast, welche Bedeutung hatte die Code-Qualität für dich?**

Als ich begonnen hab damit? Gar keine, weil ich habe mit Code relativ wenig zu tun gehabt und habe zu kämpfen gehabt überhaupt mal in das TYPO3-Tool reinzukommen und war am Anfang eher der Administrator und hab Sachen ausprobiert, Extensions einfach mal installiert und wahrscheinlich Webseiten auch kaputt gemacht durchs ausprobieren und so weiter und musste mich auch erst mal zurechtfinden, also ich habe da noch nicht reingeguckt. Das liegt daran, dass ich aus der Mediengestalter-Branche komme und nicht aus der Informatiker.

3. **Hat sich diese Bedeutung für dich im Laufe der Zeit verändert?**

Ja absolut. Ich bin dann später reingerutscht, also weg vom Optischen, weg vom Design her hin zum Coden, das hat einfach mehr Spaß gemacht und dann wird es immer wichtiger und natürlich als Firma xxx mit 35 Leuten, das ist natürlich immens wichtig für uns. Wenn ich etwas baue, dann muss es auch mein Kollege verstehen oder ich selber muss es auch noch verstehen in 3 oder 4 Wochen und das ist teilweise nicht zu unterschätzen.

4. **Benutzt du in deinem Alltag als Entwickler:inn statische Analyse-Tools um die Code-Qualität zu überprüfen? Wenn ja, welche?**

Ja. Es gibt den PHP CS Fixer. Ich bin mir nicht sicher, ob dass alles statische Code-Analyse ist, ich glaube aber schon. PHP Stan glaube ich gibt es auch. Das nutzen wir bei uns in der Firma, das haben wir bei unseren Projekten mit drin. Das heißt jeder Backend-Entwickler, Backend-Entwickler sind eben die, die die Extensions bauen, die PHP bauen. Immer wenn die etwas commit wird ein pre-commit-hook ausgeführt und dann wird im Prinzip mit statischer Code-Analyse drüber geguckt, ob das halbwegs passt. Früher gab es noch ein Tool von einer anderen Marketing-Agentur, die hat das bereit gestellt gehabt vor 3 Jahren glaube ich zuletzt und die hat die ganzen Extensions gescannt und man konnte den Extension-Namen dort eingeben und hat dann auch Tipps gegeben, also das war wirklich hilfreich. Da hatten wir damals vor 6 oder 7 Jahren zum Beispiel gesagt, bei PowerMail war es damals glaube ich, die soll spezifische Exceptions schmeißen und nicht unspezifische, also nicht einfach nur sagen „da stimmt was nicht" oder ein throw exception, sondern eine eigene Exception anlegen, damit man die später auch besser fangen kann. Das hat mir schon geholfen aber wie gesagt, die haben dann den kostenlosen Dienst eingestellt. War aber hilfreich tatsächlich.

Code-Qualität und TYPO3 Extensions

1. **Auf welche Bereiche in der Softwareentwicklung wirkt sich deiner Erfahrung nach Software-Code-Qualität aus? (Beispiele: Anforderung, Planung, Implementierung, Testing, Veröffentlichung, Wartung)**

Gute Frage. Also auf der Hand liegt natürlich, dass es sich vor allem in der Wartung auswirkt und in der Erweiterbarkeit, also wenn der Kunde neue Wünsche hat oder es gibt von außen irgendwelche Änderungen, also Schnittstellenänderungen, an die wir noch mal ranmüssen. Das ist wahrscheinlich mit das wichtigste. In der Planung natürlich auch, gute Softwarequalität fängt ja bei der Planung eigentlich schon an. Da macht es ja durchaus Sinn sich mit den Kollegen und Kolleginnen zusammenzusetzten um überhaupt erst einmal darüber nachzudenken, wie brauchen wir es denn eigentlich? Da geht es aber natürlich noch nicht so ins Detail wie die Funktionen heißen oder wie die Klassen heißen, sondern eher wie das grundlegende Modell aussieht.

2. **Welche Merkmale oder Kriterien sind aus deiner Erfahrung besonders wichtig, um die Code-Qualität von TYPO3 Extensions zu bewerten?**

Ja, gute Frage. Da hat jeder so seine eigene Art. Also es gibt eine Kollegin von mir da weiß ich, ihr ist besonders wichtig, das automatisiertes Testen am Start ist. Was aber auch nur bedingt aussagt, wie gut der Code ist, teilweise schon, aber schwierig. Also für mich hat sich im Laufe der Jahre herausgestellt, dass wenn man eine Extension aufmacht, dass man da sofort ein Gefühl für bekommt, ob die Extension modern ist und sich an Paradigmen und Konventionen hält oder eben nicht. Also das sind so Beispiele wie ein Slim-Controller in TYPO3 Extbase, ist das wichtig, also der Controller ist das Herzstück einer Extension, weil es zusammenläuft. Und da ist es wichtig, dass man kleine, schlanke Methoden hat, also Actions. Die dann wiederum größere Klassen aufrufen, das heißt die Logik würde sowieso im Bereich Domain landen, folgt dem Domain-Model-View Controller, MVC-Konzept. Eine schlechte Extension hat oftmals, merkt man von Anfängern, ganz viel von diesem Zeug im Controller drin. Also es sind dann große Controller mit großen Funktionen. Sauberer ist es, wenn ich das verpacke in einzelne kleine Service-Klassen, dann kann ich es auch wiederverwenden und weiß auch wofür die bestimmt sind, weil die dann eben einen speziellen Namen haben. Also man kriegt sofort ein Gefühl dafür, wenn man es aufmacht. Oder Dependency Injection ist ja ein neuer heißer Scheiß sozusagen. Man merkt auch schon, wenn man so eine Extension öffnet, ob die Klassen per DI injected werden oder ob die irgendwie anders reinkommen, also auf einem alten Weg. Hier hat man auch schon so ein Gefühl dafür.

3. **Hast du schon mal mit fremdem Code gearbeitet, der in deinen Augen von schlechter Qualität war? Wenn ja, was genau hast du an diesem Code als schlecht empfunden?**

Ja, eigentlich nur. Es gibt so das Sprichwort, dass wenn man ein Projekt von einer anderen Agentur übernimmt, dass da jeder Entwickler sagt „man muss es sofort neu machen". Ganz so ist es zwar nicht aber es geht schon so in die Richtung. Das ist eben auch schwer, weil verschiedene Entwickler verschiedene Herangehensweisen haben um Probleme zu lösen und TYPO3 ist ja sehr offen und sehr umfangreich, jeder kann die Probleme ein bisschen anders lösen. Das ist insofern dann nicht schwierig, wenn es eine einfache kleine Extension wird. Wenn es aber etwas Großes ist, was wir als Agentur oder ich als Entwickler übernehmen muss und pflegen können muss über die Jahre hinweg, das ist so unser Ziel, dann ist es natürlich schwer, wenn es ganz andere Wege geht. Um deine Frage zu beantworten, ja relativ häufig. Das liegt

nicht daran, dass die anderen alle schlecht sind, sondern es ändert sich auch Zeug im Laufe der Zeit. Wenn ich heute etwas programmiere und ich fasse es nicht mehr an, ist es automatisch in 5 Jahren Müll. Also diese riesen Controller ist immer wieder ein Thema, mit Logik innendrin oder anomic models, ich sage mal „dumme" Modelle, sind das. Die haben nur Getter und Setter, die nichts machen, also da ist keine Logik drin, sondern die geben, dass was aus der Datenbank ausgelesen wird 1:1 wieder nach vorne in den view. Das kann man schon mal machen aber gerade bei großen, komplizierten Dingen ist das natürlich sinnvoll da die Logik einzubauen, man hat beispielsweise ein Model, das gibt eine Person zurück ins Frontend und wir wollen wissen, ob die Person 18 Jahre alt ist, also erwachsen, weil in dem Model gibts vielleicht auch noch einen Geburtstag oder Geburtsdatum und im schlimmsten Fall wäre dann noch eine Berechnung im Controller drin, die dann jedes Mal sagt ist der 18? Ja oder nein und im besten Fall ist das im Model verankert und es gibt einen eigenen getter und gibt zurück ist die Person erwachsen. Oder wenn man an TYPO3 vorbei arbeitet ist das auch schwierig. Da haben wir auch das Thema Sicherheit zum Beispiel. Wenn Agenturen eigene Datenbank-Funktionen aufrufen und nicht auf Fluid setzten oder ganz was Eigenes bauen, weil es grad cool ist, dann ist es schwer für jemanden anderes zu verstehen warum, um es absichern zu können. Das haben wir oft, dass der Kunde sagt „wir sind gehackt worden", es ist aber so verbaut und so groß, dass es gar nicht so einfach ist, da schnell mal was abzusichern.

a. Würdest du sagen, dass dieser schlechte Code sich auf deine Motivation ausgewirkt hat, weiter an diesem Code zu arbeiten?

Absolut, ja. Wir haben vor kurzem erst mal ein Projekt übernommen und ich habe mich breitschlagen lassen, wir übernehmen das, wir bauen es nicht neu. Wir hätten es gern neu gebaut, eine Auflistung von Dateien im Frontend, je nach Berechtigung und so weiter. Und das war dem Kunden aber wichtig, dass es sehr schnell passiert, weil die alte Extension sehr viele Sicherheitslücken hatte und wir haben uns breitschlagen lassen und seit über einem Jahr baue ich persönlich und zwei, drei meiner Kollegen an diesem Projekt herum und es ist einfach sehr frustrierend. Eine kleine Änderung dauert sehr lange. Und es ist auch für den Kunden sehr frustrierend, weil er muss sehr viel Geld zahlen und versteht nicht, warum es so lange dauert und stellt uns natürlich auch in Frage auf eine Art und Weise. Es ist ärgerlich, dass wir es nicht neu gemacht haben.

b. Gibt es bestimmte Merkmale von schlechtem Code, die dir immer wieder begegnen?

Immer wieder haben wir Verstöße gegen die Coding-Guidelines von TYPO3 oder PSR-Guidelines. Wir haben immer wieder mit Sicherheitsproblemen zu kämpfen. Und was auch herauslesbar ist für einen Senior-Entwickler ist, wenn man auf große, komplexe Projekte guckt, die verschieden an verschiedenen Stellen verschieden gemacht sind. An einer Stelle ist zum Beispiel eine Datenbankabfrage über einen Query-Builder, an einer anderen Stelle ist eine Datenbankabfrage direkt an TYPO3 vorbei und direkt über PHP und an einer dritten Stelle ist es wieder ganz anders gelöst. Also man merkt dann, dass jeder mitgearbeitet hat und jeder hat seinen eigenen Stil mit eingebracht. Das springt einem ins Auge.

4. Hast du bereits die Erfahrung gemacht, dass du in einem Projekt unter großem Zeitdruck TYPO3 Extensions entwickeln oder erweitern musstest?

Ja klar. Also das versuchen wir natürlich weg zu nehmen. Auch unsere Projektmanager versuchen es wegzunehmen aber es kommt immer wieder mal dazu.

a. **Wie groß ist die Motivation unter diesem Zeitdruck auf eine gute Code-Qualität zu achten?**

Ja. Wir kriegen ein Projekt, das sagen wir mal sehr kaputt ist und müssen daran arbeiten und haben vielleicht noch die Zeit aber dadurch, dass es eben so lange dauert, wird der Zeitdruck immer größer und man will natürlich immer, wenn man ein neues Projekt anfängt sauber arbeiten. Also man möchte jede Datei sauberer verlassen, als man sie vorgefunden hat. Aber der Zeitdruck, der sich dann aufbaut sorgt dann dafür, dass du auch wieder schlecht arbeitest tatsächlich. Weil du es dann nur noch fertigbekommen willst. Das ist absolut so, ja.

b. **Wie würdest du die Wichtigkeit von Code-Qualität in Projekten ein kategorisieren? (Aus deiner Sicht und aus Management-Sicht)**

Das kommt darauf an, ob ich das Projekt noch lange betreuen muss oder ob es danach wieder wegkommt. Es ist tatsächlich so, dass wenn wir einen Kunden haben, mit dem keiner mehr Lust hat zusammenzuarbeiten und wir wollen eigentlich nur noch, dass das Projekt fertig ist und wir hoffen schon, dass der Kunde uns danach wieder verlässt, dann ist mir das nicht so wichtig, wie bei anderen Kunden, wo ich schon weiß, da möchten wir was erreichen und über die Jahre hin besser werden und das ist dann natürlich deutlich wichtiger, dass wir da Zeit investieren, sogar auch uns nehmen und Sachen besser machen als vorher. Das heißt automatisierte Tests einzubauen oder CI-Pipelines oder so etwas. Die Projektleitung sollte auch ein Gefühl dafür haben, wenn nicht, dann sollte man es dringend nachholen. Aber das ist auch die Aufgabe der Senior-Entwickler das der Projekt-Leitung klar zu machen, was passiert denn, wenn wir jetzt hier technische Schulden belassen. Was heißt das eigentlich. Das heißt ja letztlich dann, dass es irgendwann zurückkommt.

5. **Welche Maßnahmen oder Praktiken können dabei helfen von Anfang an guten Code zu schreiben?**

Test-Driven-Development ist natürlich eine gute Sache. Wenn man statische Code Analysen in seine Projekte und in den Workflow einbaut, wenn ich schon beim Commiten die Rückmeldung bekomme, dass da ein Fehler drin ist, das hilft mir natürlich enorm. Was auch super ist, kann ich nur empfehlen ist Pair-Programming oder auch mit mehreren, das ist wahnsinnig hilfreich, da kommen dann Diskussionen auf, wie man Funktionsnamen sinnvoll benennt oder so. Es ist super. Was wir mehr machen wollen, was wir noch zu wenig machen ist, dass wir auch nach dem Projekt noch mal einen Pen-Test fahren, einfach was die Sicherheit angeht. Und was helfen kann aber nicht zwangsweise hilft ist tatsächlich eine KI. Es gibt ja jetzt die Möglichkeit z. B. mit Github Copilot eine KI mit reinzuholen und die hilft einem dann bereits beim Coden und macht Vorschläge, man muss aber echt aufpassen, die KI liefert auch teilweise schlechten Code, also da sind dann zum Beispiel keine Typen-sicheren Vergleiche oder sowas. Man muss schon wissen was man macht. Kann einem aber helfen beim Einstieg.

Statische Software-Analyse-Tools und Code-Qualität

1. **Welche Aspekte von Code-Qualität können deiner Erfahrung nach statische Analyse-Tools in TYPO3 Extensions erfassen und bewerten?**

So eine statische Analyse hilft einem schon bei generellen Dingen. Es zeigt dir teilweise die Komplexität von Klassen und Funktionen. Allerdings auch nur bedingt. Ich kann ja noch so viel statische Code-Analyse draufschmeißen, das hilft mir zum Teil, dass ich es besser verstehe was

da passiert ist aber es kann natürlich trotzdem Müll sein am Schluss. Ich persönlich glaube, dass momentan Pair-Programming oder eine Schulung, das ist das A und O.

   a. **Gibt es auch Aspekte der Code-Qualität, von denen du denkst, dass sie nicht durch solche Tools zu erfassen sind? Wenn ja, welche?**

Ja, na klar. Sinnvoll wären Funktionsnamen, da in der Richtung hatte ich noch nichts. Oder Klassennamen. Damit auch der nächste versteht was zu tun ist. In der PHP-Welt gehen wir ja den Weg, dass wir weniger Code-Kommentare schreiben, sondern mehr kleinere Funktionen bilden und die dann sprechend benennen, damit klar ist, was die Funktion macht, das soll möglichst im Namen schon wiedergegeben werden, dadurch ist es natürlich auch leichter testbar. Und tatsächlich ist das noch nicht überall so. Wir hatten letztens erst eine gerichtliche Auseinandersetzung und da wurde ein Gutachter vom Gericht bestellt und der hat uns dann angekreidet, dass wir zu wenig Kommentare geschrieben haben im Code. Aber ich halte es trotzdem für nicht sinnvoll, wichtig ist kleine sprechende Namen, das ist deutlich hilfreicher. Weil die Kommentare dann in einem Jahr vielleicht auch schon veraltet sind.

2. **Welche Vor- und Nachteile siehst du bei der Verwendung von statischen Analyse-Tools?**

Ja es hilft mir natürlich schon. Auch für die Anfänger auf Sachen zu achten. Oder auch für mich, wenn ich etwas vergessen habe oder mich vertippt habe. Es gibt mir noch einmal einen Hinweis, dass ich etwas irgendwo falsch gemacht habe und erinnert mich daran. Aber es ist kein Allheilmittel. Ein Mensch muss letztendlich verstehen was ein anderer Mensch gemacht hat und das ist noch schwer für eine Maschine zu verstehen.

3. **Hast du schon einmal erlebt, dass ein statisches Analyse-Tool Verbesserungsvorschläge für deinen Code gemacht hat?**

Ja, gab es auch. Logisch. Wobei die meisten guten Vorschläge kamen tatsächlich von den Kollegen, auch von den jüngeren Kollegen, die daneben saßen und nachgefragt haben, warum ich das so benannt habe, damit wurde man gezwungen noch einmal darüber nachzudenken. Die statische Code-Analyse hat mir aber auch schon geholfen. Aber mehr eher die Menschen würde ich fast sagen.

Empfehlungen und Schlussfolgerungen

1. **Auf Basis deiner Erfahrungen mit TYPO3 Extension-Programmierung. Welche Empfehlungen würdest du Entwickler / Entwicklerin geben, um die Code-Qualität zu verbessern?**

Was super hilfreich ist, es gibt einige TYPO3-Extensions auch der TYPO3-Core selber, die machen einiges gut sage ich mal. Und da zu spicken, das ist kein Geheimnis, das machen viele, also zum Beispiel die news-Extension von Georg Ringer aber auch in den Core selbst reinzugucken und zu schauen wie die das machen, wie die das Problem lösen, wie ein Pfad zu einem Bild gebaut wird. Da kann man dann in den View-Helper gucken, der so etwas Ähnliches macht. Da kann ich mir angucken wie das gemacht wird und kann die Funktion dann natürlich wiederverwenden, die TYPO3 da mitbringt. Klauen bei anderen hilft glaube ich tatsächlich. Neben dem Pair-Programming ist das super. Aber ist ja auch gewollt, da wir im Open Source sind.

2. **Welche Rolle spielen deiner Meinung nach statische Analyse-Tools bei der Verbesserung von Code-Qualität? Würdest du sagen, dass diese Tools in der Zukunft eine größere Rolle spielen werden?**

Ich glaube, dass sie weiterverbreitet sein werden als jetzt. Ich glaube, dass es noch sehr viele Agenturen gibt, die nicht damit arbeiten, da bin ich ziemlich überzeugt davon. Ich glaube, dass die Verbreitung zunehmen wird und ich glaube auch, dass es hilft. Ich bin unsicher, ob wir noch was an der Qualität machen können der statischen Code-Analyse. Wobei vielleicht mit KI tatsächlich. Schwer zu sagen, ja möglich wäre es schon.

Abschluss

1. **Fehlt deiner Meinung nach ein Themenbereich oder Aspekte, die ich fragen sollte?**

Also abschließend kann ich sagen, ich habe mir ja auch ein paar Sachen notiert. Was ist überhaupt guter Code? Das ist ja subjektiv. Aus meiner Sicht ist guter Code, Code den mein zukünftiges Ich oder meine Kollegen verstehen werden oder verstehen können und interpretieren können. Und außerdem glaube ich ist guter Code der, der keine technischen Schulden aufbaut beim Schreiben. Das hatten wir auch schon.

Anhang 2.3: Interview 3 – T

Hintergrundfragen

1. **Wie lange arbeitest du bereits mit TYPO3, insbesondere an der Entwicklung und Weiterentwicklung von Extensions?**

Gute Frage. Ich bin überhaupt nicht vorbereitet. Lass mich mal kurz überlegen. Ich meine seit 2009. Das wären jetzt irgendwie 14 Jahre.

2. **Als du mit der Entwicklung von TYPO3-Extensions begonnen hast, welche Bedeutung hatte die Code-Qualität für dich?**

Für mich persönlich? Jeder versucht natürlich immer guten Code zu schreiben. Am Anfang bist du natürlich überfordert. Ich glaube TYPO3 hat auch eine hohe Einstiegshürde, glaube ich. Früher noch höher als heute glaube ich. Manchmal war man dann auch einfach nur froh, wenn es funktioniert hat. Aber ich glaube aus dem Stadium bin ich lange raus. Man hat schon den Anspruch es auch richtig machen zu wollen. Ich glaube, dass dieses „Hauptsache, es funktionier" hast du dann oftmals bei Leuten, die die Skills nicht mitbringen. Dieses „Hauptsache, es funktioniert" kann nicht das Kriterium sein. Das ist auch so eine Agentur-Mentalität auch, Hauptsache es funktioniert, der Kunde ist glücklich, wir können abrechnen, wir verdienen. Nein, das ist nicht der Anspruch, den man haben sollte.

3. **Hat sich diese Bedeutung für dich im Laufe der Zeit verändert?**

Ich glaube das steigt mit der Seniorität, die man bekommt. Dass man einfach besser versteht, was man da programmiert, warum man etwas so macht. Man kennt dann auch mehrere Wege etwas zu tun und bewertet für sich, warum das eine besser für das andere ist. Weil man sich auch austauscht, weil man mehr Erfahrung sammelt. Ich glaube das hat mit steigender Seniorität zu tun. Das einem das dann wichtiger ist. Vielleicht auch weil man mal in Kontakt getreten ist mit statischer Code-Analyse, dass man dann weiß worauf man achten muss aber das hat dann auch eher was damit zu tun, dass man mehr Erfahrung in diesem Bereich sammelt. Ich denke das hängt mit der steigenden Seniorität zusammen, dass man da mehr Wert darauflegt.

4. **Benutzt du in deinem Alltag als Entwickler:inn statische Analyse-Tools um die Code-Qualität zu überprüfen? Wenn ja, welche?**

Nein. Das ist echt schade. Man müsste dort viel mehr machen. Ich bin auch total begeistert, dass du das mit dem SonarQube angehst. Das müsste man immer mal überall mal machen und das ist eines der Sachen, die runterfallen. Also das einzurichten.

Code-Qualität und TYPO3 Extensions

1. **Auf welche Bereiche im Software-Lebenszyklus (Beispiele: Testing, Einführung, Wartung/Pflege, Fehlerbehebung) wirkt sich deiner Erfahrung nach Software-Code-Qualität aus?**

Die richtige Antwort ist glaube ich auf alle Bereiche. Aber die meisten werden vielleicht auch Antworten, dass es hinten raus im Zyklus, merkt man ja, dass mit schlechter Code-Qualität Aufwände, Kosten dementsprechend immer höher werden. Also je besser eine Software von

Anfang an designt ist, je besser die Code-Qualität dann auch ist, umso leichter lässt sie sich refactoren, umso leichter lässt sie sich erweitern, anpassen und so weiter. Und in der Einführungsphase hast du dann meistens ein Team von Entwicklern dran, die die Software gut kennen, die das von Grund auf neu schreiben, die sich dabei etwas gedacht haben, sich auch austauschen und später ist das vielleicht alles schon ein bisschen weg, da hast du dann vielleicht eine Fluktuation im Team, dann sitzt da vielleicht auch nur eine Einzelperson dran, die bei der Einführung und bei den Gedanken, die man sich gemacht hat gar nicht dabei war, die das alles also nicht hat. Dann ist es vielleicht auch nicht dokumentiert. Je schlechter der Code dann ist, umso schlechter ist er dann lesbar, umso schlechter kann er verstanden werden. Und umso eher kommt man dann in Sachen rein, dass man dann Lösungen sucht, wo man sagt „Hauptsache, es funktioniert". Also ich glaube mit guter Code-Qualität kann man den Lebenszyklus einer Software auf alle Fälle nach hinter raus gut verlängern noch mal.

2. Welche Merkmale oder Kriterien sind aus deiner Erfahrung besonders wichtig, um die Code-Qualität von TYPO3 Extensions zu bewerten?

TYPO3 ist da glaube ich auch noch mal so eine spezielle Sache. Im Gegensatz zum Beispiel zum Java-Umfeld hast du bei TYPO3 im Prinzip ein komplett-Paket, wo ja auch immer noch eine Datenbank dazu gehört, wo eine eigene Skriptsprache mit Typoscript dazu gehört, wo mit Fluid eine eigene Template-Sprache mit dabei ist. Es gibt ja dieses Paradigma bei TYPO3 „Convention over Configuration", also, dass auch viel „Magie" passiert für Außenstehende, also viel, was eigentlich Programm-Code wäre passiert eigentlich über Konfigurationen oder Typoscript, das ist ja auch eine Konfigurations-Sprache. Das bessert sich aber bei TYPO3, früher war mein Ansatz bei der Bewertung, das man erst mal guckt, ob sich die TYPO3-Extension an Standards hält. Das ist auch immer noch ein sehr gutes Kriterium. Code-Qualität musste ich in der Vergangenheit oft bewerten, wenn man Webseiten von anderen Agenturen übernimmt und ein z. B. ein Major-Update planen muss und dann muss man ja irgendwie eine Hausnummer sagen. Und um das machen zu können, muss man bewerten wie gut die Code-Qualität ist und wie gut man das updaten kann. Da haben wir ganz oft geguckt, ob sich an Standards gehalten wird. Wo gehört was hin, diese ganzen Conventions, kurz über den Code geguckt. Das ist eines der ersten Kriterien, die ich anwenden würde. Dann, wenn man genauer reingucken würde und dafür fehlt dann bei solchen Sachen oft die Zeit aber was für mich noch mal ein gutes Kriterium ist, das ist so ein bisschen dem PHP geschuldet, dass man typsicher unterwegs ist. Also PHP hat ja nun mal die Eigenschaft, das du halt viel Arrays hin und her schmeißen kannst, oder es gibt irgendwelche Listen, in die man etwas rein gibt. Du bist ja nicht gezwungen typsicher zu programmieren. Aber das wäre für mich mittlerweile auch ein großes und wichtiges Kriterium, um Code-Qualität zu beurteilen. Dass ich auch Models schreibe, auch wenn ich nicht muss. Dass ich nicht auf irgendwelchen Arrays bleibe und irgendwelche Sachen da rausziehe und „Magie" passiert. Dass die Funktionsparameter keine Typen haben und all so ein Kram, was halt so dazu gehört. Typsicher, ist ganz wichtig. Das kommt alles dazu. Und vielleicht bei TYPO3-Extensions, dass ich mir angucke, wie der Code strukturiert ist, also du hast ja bei TYPO3 in den Extensions einen Classes-Ordner und darunter ist ja im Prinzip der ganze Code, also alles was keine Konfiguration oder Frontend ist. Und auf der Ebene kannst du ja gucken, was derjenige für Ordner angelegt hat und wie er das strukturiert hat und wie das alles miteinander verknüpft ist. Schlechter Code wäre für mich z. B. wenn ich mir die Controller angucke in TYPO3 und die sind richtig fett, also die ganze Business-Logik hängt im Controller, ich habe möglicherweise nur einen Controller, tausend actions und innerhalb der actions eine Latte an Programm-Code. Da würde ich dann denken, dass das der Code ist, den ich dann nicht haben möchte. Es gibt ja auch verschiedene Paradigmen dafür, wie zum Beispiel die Slim-Controller-

Philosophie, Controller sollen nicht so viel machen. Und dann gucke ich mir an, was derjenige gemacht hat, schreibt er Utilitys, schreibt er Services, lagert er Sachen in die Models aus und so weiter. Wenn man da drauf gut, das wäre auch noch mal ein Kriterium, wo man sagen könnte, wo ich dann auch unabhängig davon ob es funktioniert oder nicht auf den ersten Blick sagen kann ob es gut aussieht oder nicht.

3. **Hast du schon mal mit fremdem Code gearbeitet, der in deinen Augen von schlechter Qualität war? Wenn ja, was genau hast du an diesem Code als schlecht empfunden?**

Man arbeitet laufend mit fremdem Code, selbst als ich hier bei meiner aktuellen Stelle angefangen habe, der Code, den ich bearbeite ist ja auch fremd, also ich habe den nicht geschrieben, den hat irgendjemand geschrieben, der möglicherweise gar nicht mehr im Team ist, vor Jahren, denn, wenn ich den Code jetzt bearbeite auch nicht mehr ansprechen kann. Also auch das, im Prinzip alles ist ja fremder Code für mich. Der ist immer schlecht, das sagen auch die eigenen Entwickler aber das liegt dann auch wiederum daran, dass man dringend dahin gehen müsste das zu refactorn, laufend und dafür fehlen im Agentur-Geschäft immer das Verständnis und die Mittel. Selbst die eigenen Entwickler sagen dann, das müsste man refactorn, das könnte man anders machen. Da wird jedoch gesagt, es gibt keine Zeit, das könne man nicht machen. Code kann immer verbessert werden. Ich sag auch nicht, dass alles schlecht ist, das ist natürlich auch übertrieben, das stimmt ja nicht. Je besser man sich an die Standards gehalten hat in TYPO3, umso besser kann man damit arbeiten.

   a. **Würdest du sagen, dass dieser schlechte Code sich auf deine Motivation ausgewirkt hat, weiter an diesem Code zu arbeiten?**

Ja, absolut. Wir Entwickler sind ja einfach glücklich zu machen. Immer alles neu, alles frisch, alles sauber, alles grün.

   b. **Hast du Zeit investiert, um diesen schlechten Code aufzuarbeiten und zu verbessern? Wenn ja, wie viel Zeit hat es dich gekostet? (Ungefähre Angabe in Relation zur eigentlichen Aufgabe)**

Ja. Da gibt es ja dann auch diesen Camping Ground Paradigma. Nach dem Motto „Verlasse den Camping Platz immer sauberer, als du ihn vorgefunden hast“ und nach dem Motto gehe ich eigentlich auch vor. Ich habe ja irgendeine Anforderung, die ich umsetzen muss, ich setzte es um und wenn ich dann sehe, dass da noch was ist, dann mache ich das mit. Die Schwierigkeit ist dann eigentlich den richtigen Cut zu finden, also was man noch mitmachen kann, ich muss ja auch sichergehen, dass ich damit keine Seiteneffekte auslöse. Das wäre der Worst-Case, wenn man z. B. ein kleines Feature hat, das in den Test geht und auf einmal bricht da irgendwas zusammen und dann muss ich mich rechtfertigen, warum ich das gemacht habe, oder noch mehr Fehler eingebaut, die man gar nicht gesehen hat. Da muss man gucken, dass man nur das mitmacht, was in die Story zeitlich reinpasst und was natürlich auch möglichst wenig Seiteneffekte hat. Und manchmal merkt man das auch erst nachdem man angefangen hat, nach dem Motto „Was für ein Fass mache ich hier auf“ und dann merkt man, dass es immer größer wird. Aber ich versuche das eigentlich immer zu machen, also mit über den Tellerrand zu gucken und noch mal Code zu verbessern.

c. **Gibt es bestimmte Merkmale von schlechtem Code, die dir immer wieder begegnet sind, auch in anderem Code?**

Typsicherheit, hatte ich schon genannt. Das ist so ein klassisches PHP Ding, also, wenn ich sehe, dass so riesige Arrays, am besten noch verschachtelt von links nach rechts an irgendwelche Funktionen übergeben werden. Das ist für mich eine klassische Sache für schlechten Code. Allgemein nicht typsicher unterwegs sein aber das ist auch eher so eine PHP-Geschichte wieder. Dann, dass Entwickler Concerns nicht auseinanderhalten, also, dass z. B. Sachen mit in einen Controller geschrieben werden, aus Faulheit, weil er gerade da ist, was eigentlich in einen Service ausgelagert werden müsste. Dass Sachen im Frontend in einer Template Engine gemacht werden, die eigentlich dort nicht hingehören, wo man eigentlich bei TYPO3 jetzt einen View-Helper bauen würde. Das ist auch wieder eine Sache aus dem Controller eigentlich in einem Model passieren müsste und dann aus Faulheit macht man es einfach nicht und schreibt das schnell hin und dann hat man auf einmal so einen riesen Controller. Oder wenn wir mal im Frontend sind, dass Aussehen und Verhalten nicht sauber voneinander getrennt werden, also in der Frontend-Entwicklung, dass du eine Design-Klasse hast und daneben brauchst du halt eine Verhaltens-Klasse und dann bindest du das Verhalten an die Verhaltens-Klasse und das Aussehen nur an die andere Klasse. Oder auch im Frontend wieder das Beispiel, Struktur-Klassen, Container, bei Bootstrap diese ganzen Col und Row-Geschichten und das dann wiederum trennen von den eigentlichen Design-Klassen, das nicht miteinander zu vermischen und schlechter Code ist für mich dann, wenn man aus Faulheit oder Unwissenheit das nicht von Anfang an sauber trennt. Das ist natürlich mehr Arbeit aber, wenn ich sowas sehe, dann wäre das für mich auch ein klassisches Zeichen, dafür, dass jemand hier entweder zu schnell oder aus Unwissenheit Dinge nicht sauber getrennt hat. Das sind dann auch die Dinge, die einem dann hinten raus im Refactoring dann auf die Füße fallen. Weil es dann einfach nicht mehr so einfach ist die Sachen auseinander zu ziehen. Also Typsicherheit und Concerns voneinander trennen, in allen Aspekten.

4. **Hast du bereits die Erfahrung gemacht, dass du in einem Projekt unter großem Zeitdruck TYPO3 Extensions entwickeln oder erweitern musstest?**

Klar. Ja gab es auch.

a. **Wie groß ist die Motivation unter diesem Zeitdruck auf eine gute Code-Qualität zu achten?**

Die ist immer gleich groß. Bei mir. Das kommt aber auch erst mit steigender Seniorität glaube ich. Ich habe lange auch für Agenturen gearbeitet, aber das hat eher was mit Seniorität zu tun glaube ich, dass man Aufwände nicht verhandelt, im Nachhinein. Also das ist so ein Mantra irgendwie. Der PO sagt halt was er haben will und der Entwickler entscheidet halt wie er das macht und nur der Entwickler macht die Aufwandsschätzung. Dann kann es solche Sachen wie Zeitdruck eigentlich nur geben, wenn ich mich verschätzt habe. Aber dann leidet da nicht die Code-Qualität drunter, also entweder man macht dann selbst Überstunden oder man streicht Features zusammen oder man einigt sich mit anderen Entwicklern auf weniger Funktionen. Aber Code-Qualität an sich, also ich möchte es nicht ganz ausschließen aber mein Anspruch wäre schon, dass die Code-Qualität nicht sinkt dadurch. Also eher streiche ich an allen anderen Sachen, also am Funktionsumfang oder dass ich Überstunden mache oder dass ich als Entwickler sage „das ist die Zeit für den Aufwand" und die Deadline ist mir dann egal. Also ich kann dann nicht zaubern.

b. **Wie würdest du die Wichtigkeit von Code-Qualität in Projekten einkategorisieren?**

Sollte es. Hat es aber nicht. Das ist halt wenig greifbar für die Management-Ebene. Warum soll ich hier für Dinge bezahlen, die ich nicht kalkulieren kann oder die ich nicht sehen kann. Wo man auch wieder gucken muss wie lang so ein Lebenszyklus einer Webseite ist und was bringt mir gute Code-Qualität eigentlich? Man kann auch mit schlechter Qualität eine TYPO3-Seite oder eine Webseite allgemein, je nachdem wie viel Weiterentwicklung da drauf ist 4,5 oder 6 Jahre betreiben. Und wenn dann dann die Policy eh ist, dass man dann einen Relaunch macht und alles neu macht, dann ist das vielleicht auch gar nicht so wichtig. Das ist eher der Anspruch der Entwickler, dass das wichtig genommen wird. Eher wenn ein IT-Verständnis da ist, das ist bei der Projektleiter-Stelle eher weniger ausgeprägt aber, wenn man irgendwie jemanden davor geschaltet hat oder auch der PO vielleicht ein technisches Verständnis hat, dann hat man das eher noch, dass das Verständnis für Code-Qualität da ist. Aber sonst ist es immer zu wenig. Das muss eigentlich immer von den Entwicklern kommen, die Entwickler verantworte ja den Code. Die Entwickler müssen sagen, dass Code-Qualität ihnen wichtig ist und müssen pochen und einfordern, dass entsprechend Zeit und Ressourcen dafür zur Verfügung gestellt bekommt, aus der Management-Ebene wird das niemals kommen.

5. **Welche Maßnahmen oder Praktiken können dabei helfen von Anfang an guten Code zu schreiben?**

Reden. Immer reden. Das kommt immer zu kurz, leider. Also ich glaube da muss man noch mal unterscheiden zwischen wenn man so automatische Tools hat, also die Code-Qualität auch automatisch bewerten. Wenn man solche Dinge automatisiert hat, also angefangen von Code-Formatting bis „ich habe schon ein Tool", das zeigt mir dann wo ich Mist gemacht habe. Klar, wenn man sowas schon von Anfang an am Start hat, ja. Aber solche Architektur-Entscheidungen oder auch das zum Beispiel bestimmte Sachen in einen Service gehören aber in einem Model stehen, das kannst du ja mit statischer Code-Analyse auch nicht herausfinden. Und da hilft dann tatsächlich nur reden. Das sind dann klassische Code-Reviews. Aber auch, dass alle Entwickler im Team das gleiche Verständnis von Code-Qualität haben. Zum einen, dass man Junioren damit ausbildet, wenn man so einen Code-Review macht und zeigt worauf man achten sollte. Aber auch unter seniorigen Entwicklern, dass man zum Beispiel über eine Lösungsskizze redet, im Vorfeld. Das ist aus meiner Erfahrung noch nie Umsonst gewesen über eine Lösungsskizze zu reden. Ich habe die Erfahrung gemacht, dass wenn zwei seniorige Entwickler zusammenkommen, beide haben eine andere Lösungsskizze, auch wenn es nur leicht ist und dann kann man immer noch mal diskutieren was besser ist und dann merkt man immer oder ich habe es ganz oft gemerkt, dass der andere an etwas gedacht hat, woran ich nicht gedacht habe und es ein bisschen schlechter gemacht hätte. Der Code wird immer besser dadurch. Also reden. Vorher, während und danach. Code-Reviews, Austausch, gemeinsame Plannings aber das ist alles Reden, miteinander reden, über Code.

Statische Software-Analyse-Tools und Code-Qualität

1. **Welche Aspekte von Code-Qualität können deiner Erfahrung nach statische Analyse-Tools in TYPO3 Extensions erfassen und bewerten?**

Ich weiß nicht ob das mit in statische Analyse-Tools mit rein spielt. Aber was mein wichtigstes Hilfsmittel ist um guten Code zu schreiben, ist tatsächlich die IDE und wie man sie sich einrichtet. Also wir nutzten hier meistens PHPStorm. Da gibt es halt so klassische TYPO3 oder PHP Extensions, die dir halt helfen automatisch oder mit Code Completetion oder der Hinweis,

dass du das Test-Framework richtig einbindest, dass du eine Datenbank gleich mit anbindest. Umso mehr die IDE konfiguriert wird und passend auch das Projekt auch mit Sprachversion und allem darum und daran. Oder dass bei TYPO3 die Pfade zu den View-Helpern erkannt werden oder dass man externe XML-Schema mit einbinden, also umso mehr man da tut, umso besser kann eine IDE auch einen unterstützen guten Code zu schreiben. Zum Beispiel „die Funktion ist unsicher, schreibe das anders" oder ich habe hier 3 Parameter und den kannst du weglassen, weil er default ist oder du verwendest hier irgendein Attribut, dass es nicht mehr gibt oder sonstiges. Das ist jetzt zwar kein statisches Analyse-Tool aber es ist für mich das wichtigste erst mal um sauberen Code zu schreiben.

   a. **Gibt es auch Aspekte der Code-Qualität, von denen du denkst, dass sie nicht durch solche Tools zu erfassen sind? Wenn ja, welche?**

Architektur. Weil es ja wirklich nur statisch ist, vielleicht hat sich die Welt da ja aber auch schon weitergedreht.

2. **Welche Vor- und Nachteile siehst du bei der Verwendung von statischen Analyse-Tools?**

Dass es frustriert, weil man nichts angehen kann. Oder dass man um die Ohren gehauen bekommt, dass der Code ganz schlecht ist und man ihn refactoren müsste und man hat nie Zeit dafür. Meine Erfahrung war immer, dass es das Problem ist, dass man sieht, dass es schlecht ist, man kann daran aber nichts ändern. Weil keine Zeit ist etwas zu refactoren oder weil bestimmte Dinge auch so sind, wie sie sind. Es gibt so Ansätze, dass wenn man sagt, der Code sei schlecht, das ist okay aber der neue Code muss die Code-Qualität im Gesamtprojekt steigern, dass könnte man sich als Regel setzten. Aber wenn man eh nichts daran ändert, kann das frustrieren.

3. **Welche Rolle spielen Analyse-Tools bei der der Code-Qualität in TYPO3-Extensions über die Zeit hinweg?**

Weiß ich nicht. Die erste Antwort ist ja vielleicht erst mal ja, weil man es sich wünschen würde. Ich wäre noch skeptisch dabei. Ich glaube das nicht. Es ist ganz nett aber ich glaube da fehlt mir vielleicht noch ein bisschen die Erfahrung, weil ich damit noch nicht so viel gearbeitet habe um den Nutzen wirklich abschätzen zu können. Bisher ist es für mich ein nettes Beiwerk, wo ich so ein bisschen gucken kann, wo meine Schwachstellen sind. Aber selbst wenn ich fremden Code analysieren und bewerten müsste, reicht eine statische Codeanalyse in meinen Augen nicht aus. Es ist für mich aktuell nur ein nettes Beiwerk umso ein bisschen den eigenen Code zu analysieren und Schwachstellen zu finden vielleicht.

Empfehlungen und Schlussfolgerungen

1. **Auf Basis deiner Erfahrungen mit TYPO3 Extension-Programmierung. Welche Empfehlungen würdest du Entwickler / Entwicklerin geben, um die Code-Qualität zu verbessern?**

Programmieren auf alle Fälle. Und austauschen, also Programmieren und Austauschen. Also mit erfahrenen Programmierer:innen klassische Software-Patterns durchgehen. Über Code reden.

2. **Fällt dir etwas für zukünftige Forschungen im Bereich der Code-Qualität ein? Gibt es Themen, zu denen du dir mehr Forschung wünschen würdest?**

Ich habe mich mit statischen Analyse-Tools viel zu wenig beschäftigt. Und ich glaube, dass uns diese ganze KI-Entwicklung sowieso überholen wird. Das sehen wir ja jetzt auch schon, also die KI schreibt jetzt schon besseren Code als wir. Das wird den Markt revolutionieren, also ich glaube nicht, dass wir überflüssig sein werden aber das wird uns überrollen glaube ich.

Anhang 2.4: Interview 4 – I

Hintergrundfragen

1. **Wie lange arbeitest du bereits mit TYPO3, insbesondere an der Entwicklung und Weiterentwicklung von Extensions?**

Ich habe angefangen mit TYPO3 2009. Da habe ich meine Diplomarbeit gemacht und habe eine Webseite umgesetzt und habe das mit TYPO3 gemacht. Und dann habe ich 2010 bei der MMS angefangen und bin dann gleich in dem Projektfeld eingestiegen und habe dort angefangen im TYPO3-Umfeld Sachen zu entwickeln, also Extension-Entwicklung. Also 2010 ungefähr.

2. **Als du mit der Entwicklung von TYPO3-Extensions begonnen hast, welche Bedeutung hatte die Code-Qualität für dich?**

Noch nicht so eine hohe, sage ich mal. Weil man ja auch erst angefangen hat damit zu entwickeln. Das hat sich erst über die Jahre entwickelt, dass man da mehr Wert drauflegt, dass das ein schöner Code ist und dass man ihn gut lesen kann. Das bringt die Erfahrung auch mit und da man am Anfang, bei meiner Diplomarbeit zum Beispiel habe ich auch alles alleine gemacht und mit keinem zusammengearbeitet, da ist das noch mal etwas Anderes als wenn man mit anderen zusammenarbeitet und anderen Input bekommt. So hat sich das über die Jahre entwickelt.

3. **Hat sich diese Bedeutung für dich im Laufe der Zeit verändert?**

Ja. Da ist jetzt auf jeden Fall auch ein Anspruch dazu gekommen. Ich glaube am Anfang war man froh, dass es funktioniert hat, wie man das hingeschrieben hat. Und mittlerweile legt man da eher einen Anspruch darauf, dass es auch andere lesen können und dass da auch ein paar Standards eingehalten werden, die man sich auch als Team gesetzt hat.

4. **Benutzt du in deinem Alltag als Entwickler:inn statische Analyse-Tools um die Code-Qualität zu überprüfen? Wenn ja, welche?**

Die PHPStorm bringt ja schon ein bisschen was mit. Und wir haben auch teilweise in den Projekten schon ein paar Sachen mit drin. Da wird ja quasi ein bisschen mit geguckt, das benutzte ich hauptsächlich. Es kommt auch immer ein bisschen darauf an, wie die Setups sind von dem Projekt. Unsere eigenen Setups sind da glaube ich ein bisschen besser. Ich habe jetzt viel mit Projekten zu tun gehabt, die wir nur übernommen haben von anderen Agenturen und da ist das teilweise auch noch nicht so gut umgesetzt.

Code-Qualität und TYPO3 Extensions

1. **Auf welche Bereiche im Softwareentwicklungsprozess (Beispiele: Testing, Einführung, Wartung/Pflege, Fehlerbehebung) wirkt sich deiner Erfahrung nach Software-Code-Qualität aus?**

An vielen Stellen. Allein in der Entwicklung. Wenn ich zum Beispiel etwas entwickelt habe und jemand anderes aus dem Team das weitermachen muss oder der Kunde irgendwelche Veränderung wünscht. Dann ist es für den anderen natürlich einfacher wenn irgendwelche Standards eingehalten werden oder wenn Klassen- und Methodennamen ordentlich benannt sind, damit der auch weiß, was diese Funktionalität macht. Also wenn zum Beispiel irgendwas

gelöscht werden soll und dann aber in der Funktion Sachen hinzugefügt werden, dann ist das natürlich unproduktiv und für die Qualitätssicherung auch nicht gut und der Kollege kommt damit eigentlich auch nicht klar und müsste eigentlich bei dir nachhaken. Also da wirkt sich das aus. Und für die Tests macht es das natürlich auch einfacher dezidierte Tests zu schreiben für die einzelnen Fälle. Also als wenn du eine riesige Funktion hast, die mit einem Mal alles abdeckt. Das wirkt sich dann schon aus.

2. **Welche Aspekte (Merkmale oder Kriterien) sind für dich besonders wichtig, um die Code-Qualität von TYPO3 Extensions zu bewerten?**

Die Benamung zum Beispiel. Und Dass man die Sachen an den richtigen Stellen ablegt. Dass die Views zum Beispiel wirklich nur zum Anzeigen benutzt werden und dort keine Logik drin ist. So etwas macht es dann einfacher. Und keine ellenlangen, verschachtelten Absätze oder if-whre-conditions, also ganz viele conditions ineinander gestapelt, dass man das schön, einzeln, simpel hält. Dass man es mit dem Auge schon erfassen kann, was da wirklich passiert.

3. **Hast du schon mal mit fremdem Code gearbeitet, der in deinen Augen von schlechter Qualität war? Wenn ja, was genau hast du an diesem Code als schlecht empfunden?**

Mit fremdem Code arbeiten wir durch die Übernahme der Projekte immer. Da kräuseln sich schon manchmal die Zehennägel, wenn man das so liest was dort steht. Es wächst ja auch, wenn es ältere Projekte sind. Aber da sind dann zum Beispiel Funktionen drin, die heißen deleteAndAdd gleichzeitig. Das ist dann natürlich nicht so schön und nicht so nachvollziehbar. Aber das Problem ist ja immer Budget und Zeit. Meistens will der Kunde dann dafür nicht bezahlen um das zu verbessern. Der sieht ja das immer nicht im Hintergrund, wie das da aussieht, er sieht ja nur das Endergebnis. Und deshalb ist es schwierig das dem Kunden mit zu verkaufen. Die bezahlen nicht gerne dafür Geld. Teilweise macht es dann auch das Testen für die einfacher, wenn du das ordentlich oder auch automatisiert testest. Aber das dem Kunden zu verkaufen ist eher schwierig.

   a. **Würdest du sagen, dass dieser schlechte Code sich auf deine Motivation ausgewirkt hat, weiter an diesem Code zu arbeiten?**

Eigentlich nicht, weil man will ja den Kunden trotzdem zufrieden stellen. Also man will ja trotzdem das, was der Kunde neu angefordert hat trotzdem umsetzten. Und man versucht dann da mit den vorhandenen Möglichkeiten das beste rauszuholen. Und wenn man dann noch Zeit hat, kann man auch noch mal Sachen verbessern. Das kommt darauf an, wie man es abgeschätzt hat. Wenn man gut abgeschätzt hat und dann noch Kapazitäten hat, verbessert man natürlich noch Sachen aber wenn es sehr knapp abgeschätzt ist und kein Budget mehr da ist, dann bleibt es leider so wie es ist.

   b. **Gibt es bestimmte Merkmale von schlechtem Code, die dir immer wieder begegnet sind, auch in anderem Code?**

Die Benamung ist glaube ich ein großes Thema. Wie man Variablen und Konstanten benennt und was die dann im Endeffekt auch machen oder bewirken. Das ist ein schwieriges Thema und solche großen Verschachtelungen mit ganz vielen if-then-conditions, die ineinander verschachtelt sind. Das ist etwas, was einem immer wieder über den Weg läuft.

4. **Hast du bereits die Erfahrung gemacht, dass du in einem Projekt unter großem Zeitdruck TYPO3 Extensions entwickeln oder erweitern musstest?**

Ja, das gibt es auch bei den großen Projekten. Gerade arbeite ich auch in einem Projekt, das schnell fertig werden muss. Da ist aber auch noch die Problematik, dass wir das Front-End übernommen haben, das heißt, man muss den ganzen Overhead für die Views und so mitschleppen. Und wenn man dann Sachen gerne anders gemacht hätte oder Sachen vereinheitlicht hätte, weil die zum Beispiel für jede Seite eine eigene View gebaut haben. Und der hat dann noch spezielle Container-Klassen Drumherum. Das hätten wir anders gebaut. Das ist aber etwas, das der Zeit geschuldet ist und dem Kunden, dass das so bleibt. Der Zeitdruck wirkt sich schon darauf aus. Was beim Zeitdruck natürlich auch immer hinten runterfällt sind irgendwelche Tests, die lässt man dann meistens runterfallen, zum Beispiel Unit-Tests oder automatisierte Tests. Das ist immer ein Thema, das hinten runterfällt.

   a. **Wie würdest du die Wichtigkeit von Code-Qualität in Projekten einkategorisieren?**

Ich glaube heut zu Tage spielt es eine immer größere Rolle im ganzen Kontext. Weil die meisten Sachen komplexer werden und es nicht mehr nur solche Anzeigen-Sachen sind, sondern, dass man mehr Funktionalitäten und mehr Komplexität drin hast mit Schnittstellen und so weiter und deshalb gewinnt das Thema immer mehr an Bedeutung für uns Entwickler aber leider nicht für die Kunden. Die Kunden muss man dahingehend immer noch „erziehen" sage ich mal. Dass das ein Teil des Projektes ist und dass auch darauf geachtet werden muss, um später zum Beispiel die Wartungs-Sachen geringer zu halten. Der interessiert erst mal nur, was vorne rauspurzelt aber um das später zu warten, das weiß er dann in dem Moment noch nicht, dass das dann damit vielleicht auch aufwendiger wird. Um Sachen am Leben zu erhalten.

5. **Welche Maßnahmen oder Praktiken können dabei helfen von Anfang an guten Code zu schreiben?**

Dass man im Team die gleichen Standards etabliert, also, dass man Code-Guidelines und solche Sachen hat. Dass man die mitgelieferten Tools auch benutzt, also PHPStorm und so weiter, also, dass man die Sachen dort auch gleich einbindet. Also, dass man gleich ein Projekt-Setup mit diesen Dingen hat. Budget dafür einplanen. Dass man sich sagt, dass man eine bestimmte Prozentzahl davon nimmt um dann noch mal auf diese Sachen zu achten und dann auch wirklich Tests mitzuschreiben. Oder auch das dann Tests-Schreiben auch mit zu den Coding-Guidelines dazugehört. Kommt auch darauf an wie detailliert man das runterbricht, ob man dann zum Beispiel auch sagt, wie eine Extension auszusehen hat. Das ist dann Projektabhängig.

Statische Software-Analyse-Tools und Code-Qualität

1. **Welche Aspekte von Code-Qualität können deiner Erfahrung nach statische Analyse-Tools in TYPO3 Extensions erfassen und bewerten?**

Also die statischen Analyse-Tools, die gucken ja eher so nach der Zeilenlänge oder wo die Klammern sitzen. Ob Leerzeilen mit drin sind, solche Sachen. Das statische Analyse-Tool wird mir ja nichts dazu sagen, dass zum Beispiel eine Funktion total komplex innen drin ist und verschachtelt. Sie geben dann eher an, wenn man die Hilfen mitbenutzt, dass man dann schon ein bestimmtes Repository hat oder Funktionen, die schon da sind. Oder dass da doppelter Code ist. Nicht, dass man etwas einfacher machen könnte, wenn es sehr verschachtelt ist.

a. **Würdest du sagen, dass die genannten Aspekte, die für dich eine gute Code-Qualität ausmachen durch so ein statisches Analyse-Tool bewertet werden können?**

Ja, die werden ja auch immer ausgereifter. Also es entwickelt sich ja dort auch weiter. Da kommen ja auch ein paar Sachen dazu und solche Sachen, die früher nicht so im Fokus waren glaube ich. Das wird ja jetzt immer besser auch das TYPO3 hat sich ja in der Hinsicht auch ein bisschen weiterentwickelt, dass man Sachen besser findet und nicht immer nur sich durch den Core händisch durchsuchen muss.

b. **Gibt es auch Aspekte der Code-Qualität, von denen du denkst, dass sie nicht durch solche Tools zu erfassen sind? Wenn ja, welche?**

Die Verständlichkeit des Codes. Die Benamung, dass was das Ding dann eigentlich tut. Das kann er ja auch nicht. Wenn die irgendwie „ichMacheHeuteDenTagSchön" heißt und dann schlechtes Wetter bringt oder sowas. Das kriegt das ja auch nicht hin.

2. **Welche Vor- und Nachteile siehst du bei der Verwendung von statischen Analyse-Tools?**

Vorteil ist natürlich, dass man in den Teams besser arbeiten kann. Dass man das selbe Verständnis hat von Sachen und wie sie umgesetzt werden. Es hilft die Standards für das Team aufrecht zu halten, die man sich gesetzt hat. Wir haben jetzt zum Beispiel Fälle, bei denen wir statische Analyse-Tools eingebunden haben und da kann man dann die pre-hooks oder die git-hooks auskommentieren und trotzdem die Sachen pushen. Man kann es halt auch umgehen, wenn man es nicht richtig eingebunden hat. Es kommt aber darauf an, wie man die deplayment-pipeline macht. Dort hat man das ja eigentlich auch noch mit drin aber lokal kann man das schon umgehen. Sonst sehe ich keine weiteren Nachteile.

3. **Hast du schon einmal erlebt, dass ein statisches Analyse-Tool Verbesserungsvorschläge für deinen Code gemacht hat?**

Mit der Vervollständigung sieht man schon welche Klasse oder welche Funktionen schon da sind, die man dann benutzen kann. Das macht es jetzt schon ganz gut.

Empfehlungen und Schlussfolgerungen

1. **Auf Basis deiner Erfahrungen mit TYPO3 Extension-Programmierung. Welche Empfehlungen würdest du Entwickler / Entwicklerin geben, um die Code-Qualität zu verbessern?**

Dass er sich beim Team informiert, welche Standards und Guidelines es gibt und dass er sich das in seiner Entwicklungsumgebung einbindet und sich das auch mal anguckt, was da alles noch möglich ist. Dass er nicht mehr wie früher im Notepad programmiert zum Beispiel. Dass man eine ordentlich IDE benutzt und die Vorteile davon sieht.

2. **Welche Rolle spielen deiner Meinung nach statische Analyse-Tools bei der Verbesserung von Code-Qualität? Würdest du sagen, dass diese Tools in der Zukunft eine größere Rolle spielen werden?**

Ja, würde ich schon sagen. Weil sich da alles ja auch weiterentwickelt hat. Also TYPO3 hat sich ja auch stark verändert, im Aufbau und allem. Setzt das alles um, was quasi neu am Markt ist.

Anhang 2.5: Interview 5 – M

Hintergrundfragen

1. **Wie lange arbeitest du bereits mit TYPO3, insbesondere an der Entwicklung und Weiterentwicklung von Extensions?**

Seit 2006. Da habe ich angefangen.

2. **Als du mit der Entwicklung von TYPO3-Extensions begonnen hast, welche Bedeutung hatte die Code-Qualität für dich?**

Damals noch nicht wirklich, ich glaube damals war ich auch noch im Studium. Ich glaube dem habe ich am Anfang wenig Aufmerksamkeit geschenkt. Hauptsache es hat funktioniert. Das ist auch noch heute so, dass wenn man auf alten Quellcode guckt, ist das schon ein bisschen peinlich. Man lernt halt über die Jahre.

3. **Hat sich diese Bedeutung für dich im Laufe der Zeit verändert?**

Ja, die hat stark zugenommen, man hat auch mehr Ansprüche an den Quellcode, vor allem, wenn man auch fremden Quellcode liest. Das verändert sich einfach mit der Zeit. Man merkt auch, dass vieles besser funktioniert und man weniger Fehler macht. Das kommt dann einfach mit der Erfahrung.

4. **Benutzt du in deinem Alltag als Entwickler:inn statische Analyse-Tools um die Code-Qualität zu überprüfen? Wenn ja, welche?**

Ja, nutzte ich. Angefangen erst mal mit den Tools, die die IDE schon bietet, die auch schon in Richtung statische Code-Analyse gehen. Also gerade PHPStorm, da gibt es so einen PHP Extended Plug-In und es gibt auch schon viele Hinweise auch während der Entwicklung, was man besser machen könnte, z. B. hatte ich jetzt erst vor 5 Minuten den Fall, dass ein Hinweis kam, dass man diese if-condition auch anders schreiben kann oder der hintere Ausdruck ist schneller als der vordere Ausdruck, das solltest du umdrehen. Also solche Tipps gibt es schon von der Entwicklungsumgebung, ansonsten klar, die üblichen Tools, wie Psalm, PHPStan und weitere nutzen wir in unseren Projekten.

Code-Qualität und TYPO3 Extensions

1. **Auf welche Bereiche im Software-Lebenszyklus (Beispiele: Testing, Einführung, Wartung/Pflege, Fehlerbehebung) wirkt sich deiner Erfahrung nach Software-Code-Qualität aus?**

Auf jeden Fall bei der Entwicklung. Also auch während der initialen Entwicklung. Wenn quasi qualitativen Code schreibt, wenn man Qualität mit ins Projekt bringt, dann ist man meistens vielleicht noch ein bisschen langsamer unterwegs, weil man vielleicht noch Unit-Tests schreibt. Was sich aber vielleicht hinten raus bei der Wartung des Projektes, wenn es robuster ist oder Fehler durch verständlichen Code leichter gefunden werden, also eigentlich wirkt es sich an jeder Stelle aus, in der einen oder anderen Form.

2. **Welche Merkmale oder Kriterien sind aus deiner Erfahrung besonders wichtig, um die Code-Qualität von TYPO3 Extensions zu bewerten?**

Lesbarkeit, Verständlichkeit, das ist natürlich sehr wichtig, dass ein Code zuverlässig ist. Wiederverwendbarkeit, ist natürlich ein Thema. Ich denke auch das Thema Performance, Effizienz ist wichtig. Es muss auch gut testbar sein. Man sollte die Code-Guidelines einhalten. Die ganzen Prinzipien, wie „KISS“ und Single-Responsible-Prinzip und diesen ganzen Clean Code-Prinzipien. Und in Bezug auf TYPO3 Extensions, zu einer sauberen Programmierung gehört natürlich auch, dass Standards genutzt werden, die vom Core bereitgestellt werden, wie dependency injection oder irgendwelche Core-API's, wie die middleware-Konstrukte, dass man da nicht drin rumprogrammiert. Dann, dass eine Extension konfigurierbar, erweiterbar ist. Bevor ich eine Extension einsetzte, gucke ich mir immer den Quellcode an. Spagetti-Code, den installiere ich mir zum Beispiel nicht. Da muss man vorher mal reingucken.

3. **Hast du schon mal mit fremdem Code gearbeitet, der in deinen Augen von schlechter Qualität war? Wenn ja, was genau hast du an diesem Code als schlecht empfunden?**

Ja, klar. Ich kann mich jetzt an so ein Ding erinnern, das war total ineffizient geschriebener Code. Der war völlig unverständlich, sehr verschachtelte for- und if-Schleifen. Und inperformant, also das heißt er war richtig langsam. Den konnte man wirklich anders schreiben. Also generell vom Algorithmus her, dass das dann auch wesentlich schneller war. Und damit war es dann auch viel besser lesbar. Wenn ich z. B. Controller mit 5000 Zeilen sehe oder Controller Actions, dann sage ich schon, dass hier irgendwas schiefgelaufen ist.

   a. **Würdest du sagen, dass dieser schlechte Code sich auf deine Motivation ausgewirkt hat, weiter an diesem Code zu arbeiten?**

In bestimmten Situationen vielleicht. In anderen Situationen, wenn man zum Beispiel mit Legacy Projekten arbeitet ist es eher auch motivierend aus etwas Altem etwas tolles neues und Modernes zu machen. Also es macht schon Spaß auch etwas zu modernisieren. Es kann auch schon motivierend sein.

   b. **Hast du Zeit investiert, um diesen schlechten Code aufzuarbeiten und zu verbessern? Wenn ja, wie viel Zeit hat es dich gekostet?**

Ja, auf jeden Fall. Natürlich ist es aber auch immer eine Frage mit Zeit und Budget. Wenn ich es besser machen kann, dann investiere ich auch schon mal eine Zusatzstunde oder auch zwei, ohne dass der Kunde bezahlt. Das ist einfach so ein eigener Anspruch, wenn man solche Baustellen sieht.

   c. **Gibt es bestimmte Merkmale von schlechtem Code, die dir immer wieder begegnet sind, auch in anderem Code?**

Ja, zu lange Funktionen oder zum Beispiel, das habe ich jetzt erst wiedergehabt, so ein altes Projekt, wo zum Beispiel eine Funktion mehrere Funktionen abdeckt. Die Funktion heißt zum Beispiel „deleteAndUpdateUser“, die macht halt zwei Sachen in einer Methode und das ist irgendwie nicht gut. Oder gerade wenn man ganz viel Logik in einer Methode hat, wo sich eine Methode oder auch eine Klasse um viel zu viele Themen kümmert. Das begegnet einem relativ häufig würde ich sagen. Generell überladene Methoden oder Klassen, die man durch auf splitten vereinfachen könnte. Das wäre dann auch einfacher testbar. So etwas ist glaube ich

sehr häufig. Komplexe if-Abragen gibt es auch, wo dann 20 Bedingungen in einer Abfrage sind, dann muss man verstehen was das ist und das ist dann natürlich schwierig. Das kann man auch irgendwie anders lösen. Am Ende schreibt man aber auch selbst manchmal sowas, wo man dann weiß was dahintersteckt aber der andere Entwickler, der sich das anschaut, weiß es dann halt nicht. Entweder steht das dann im Kommentar drüber oder man löst es dann halt anders. So etwas gibt es häufig.

4. **Hast du bereits die Erfahrung gemacht, dass du in einem Projekt unter großem Zeitdruck TYPO3 Extensions entwickeln oder erweitern musstest?**

Ich persönlich, ja klar. Wir hatten erst letztes Jahr so einen Fall bei einem Projekt, wo wir wirklich in relativ kurzer Zeit etwas entwickeln mussten, was relativ komplex war. Und die Entwickler habe das dann auch gemacht. So entsprechend sieht das dann auch aus.

   a. **Wie groß ist die Motivation unter diesem Zeitdruck auf eine gute Code-Qualität zu achten?**

Die Motivation ist da hoch, also bei mir wäre sie hoch aber wenn es die Zeit halt nicht zulässt, ist das wirklich ein Problem. Also bei einfachen Sachen kann man schon darauf achten aber wenn es dann komplexer wird, wo man dann mal splitten muss oder Service-Klassen erstellen muss, das alles schönmachen muss, wenn dann die Zeit fehlt, dann ist das ärgerlich aber das ist dann halt so. Der Kunde möchte es so.

   b. **Wie würdest du die Wichtigkeit von Code-Qualität in Projekten einkategorisieren?**

Ich messe der Sache hohe Priorität bei, das ist wichtig, gerade bei langwierigen Projekten, wie wir sie haben. Weil der Quellcode muss halt verständlich sein, es kommen neue Entwickler ins Projekt oder es gehen auch mal Entwickler aus dem Projekt raus, also wir haben eine ständige Rotation und da ist es wichtig, dass die Code-Qualität stimmt. Schon hoch würde ich jetzt sagen.

5. **Welche Maßnahmen oder Praktiken können dabei helfen von Anfang an guten Code zu schreiben?**

Auf jeden Fall Code-Reviews. Ich glaube das ist eine der besten Maßnahmen, die man machen kann. Dass man seinen Code anderen zeigt. Gerade zum Thema Verständlichkeit ist das das erste, was man tun sollte. Natürlich auch statische Codeanalyse, Code-Guidelines, vielleicht auch Vorgaben machen, dass man sich zum Beispiel an irgendwelche TYPO3-Guidelines hält. Tests, Unit-Tests schreiben, nicht zu viel aber auch nicht zu wenig. Ich empfehle immer schon Unit-Tests zu schreiben, weil das hilft auch Code so zu schreiben, dass er leichter wartbar ist. Wenn man Code testen kann mit Unit-Tests und da nicht total komplizierte Unit-Tests schreiben muss, dann ist das ein gutes Zeichen. Und natürlich die statische Code-Analysen, um halt Fehler aufzudecken.

Statische Software-Analyse-Tools und Code-Qualität

1. **Welche Aspekte von Code-Qualität können deiner Erfahrung nach statische Analyse-Tools in TYPO3 Extensions erfassen und bewerten?**

Die Formatierung erst mal, dass man halt eine einheitliche Codeformatierung hat. Dann können potentielle Fehlerquellen, so etwas wie Return-Werte, wenn eine Funktion, die etwas returnen könnte auch Null sein könnte aber der Entwickler das nicht beachtet, also alles potentielle Fehler. Vielleicht auch Sicherheits-Themen könnten gefunden werden von der statischen Code-Analyse. Zum Beispiel unsichere Hash-Funktionen. So etwas bekommen wir auch häufig mit rein. Man könnte auch auf Design-Fehler hinweisen, auf Code-Design also auf Anwendungs-Design-Fehler. Und natürlich auch zu komplexen Quellcode, also Verständlichkeit. Da kommen ja auch Hinweise dazu.

   a. **Würdest du sagen, dass die genannten Aspekte, die für dich eine gute Code-Qualität ausmachen durch so ein statisches Analyse-Tool bewertet werden können?**

Nicht vollständig glaube ich. Es gibt natürlich viele Sachen, die findet so ein Tool vielleicht nicht. Also gerade semantische Sachen oder die Bedeutung hinter dem Code verstehen. Daher sind das vielleicht auch viele Fehler, die gar keine Fehler sind, also false-positives. Aber auch, dass man Entwurfsmuster anwendet oder dass man jetzt Quellcode besser mit diesem oder einem anderen Entwurfsmuster machen könnte. Dass das viel flexibler und verständlicher wäre, ich glaube das können solche Tools dann nicht, das kann nur jemand dann mit Code-Review herausfinden, wenn ein Senior-Entwickler dann sieht, wie es einfacher gehen würde. Ich sehe dann vielleicht auch im Code, dass etwas performance-Probleme machen würde, dass würde die statische Code-Analyse vielleicht gar nicht erkennen. Vielleicht weiß das nur ich, dass diese Methode sehr oft aufgerufen wird. Das kriegt die statische Analyse vielleicht gar nicht mit, weil es die Laufzeiten nicht untersuchen kann.

   b. **Gibt es auch Aspekte der Code-Qualität, von denen du denkst, dass sie nicht durch solche Tools zu erfassen sind? Wenn ja, welche?**

Semantische Sachen oder Sachen, die zur Laufzeit passieren, was das Tool nicht wissen kann, dass irgendwelche Daten dahinterstehen. Bedeutung von dem Code.

2. **Welche Vor- und Nachteile siehst du bei der Verwendung von statischen Analyse-Tools?**

Vorteil ist natürlich, das Tool findet viel. Der Code wird tatsächlich verbessert. Es kann halt potentielle Fehler finden, Sicherheitsrisiken und so weiter. Nachteil ist die Konfiguration und Wartung solcher Analyse-Tools. Könnte natürlich auch einen Aufwand bedeuten. Es ist einfach ein zeitlicher Aufwand auch wenn man jetzt etwas entwickelt und am Ende lässt man diese Codeanalyse durchlaufen und dann kommen da sehr viele Themen, das ist natürlich viel. Die muss man sich dann alle angucken. Das ist ein zeitlicher Aufwand auf jeden Fall. Man darf sich darauf nicht verlassen, dass nur mit den Tools alles funktioniert.

3. **Hast du schon einmal erlebt, dass ein statisches Analyse-Tool Verbesserungsvorschläge für deinen Code gemacht hat?**

Klar, Verbesserungsvorschläge auf jeden Fall. Also gerade bei solchen Sachen, wo Null zurückkommen könnte. Und wie gesagt in PHPStorm hatte ich jetzt erst den

Verbesserungsvorschlag, dass ich die Statements in meiner if-Bedingung umdrehen soll, weil das eine schneller als das andere ist. Oder Vorschläge, dass man etwas anders schreiben kann. So etwas kommt schon teilweise zurück. Aber jetzt nicht in dem extrem großen Umfang würde ich sagen.

Empfehlungen und Schlussfolgerungen

1. **Auf Basis deiner Erfahrungen mit TYPO3 Extension-Programmierung. Welche Empfehlungen würdest du Entwickler / Entwicklerin geben, um die Code-Qualität zu verbessern?**

Man sollte auf jeden Fall die Tools aus der IDE nutzten. Die von Anfang an verbesserungswürdige Geschichten unterstreichen. In Extensions sollte man nicht jeden Quellcode in den Controller packen, sondern Utility-Klassen und Services nutzten. Ein Hinweis vielleicht noch, man kann auch ganz gut, gerade in Extension-Programmierung sich den Quellcode von anderen Entwicklern anschauen, vor allen von Entwicklern, wo man weiß, dass es gut ist. Wie zum Beispiel die News-Extension von Georg Ringer, dass man sich solche Extensions oder von der TYPO3 GmbH selbst programmierte Extensions sich anschaut, sich Ideen holen, mal reinschaut und sich anguckt, wie die Sachen umgesetzt haben. Das empfehle ich auch generell also nicht bloß in Bezug auf TYPO3. Dass man da auch von anderen lernt. Dann noch vielleicht Unit-Tests, schreiben zu versuchen. Man soll ja nicht alles mit Unit-Tests versehen, ich glaube dadurch wird eine Anwendung viel zu wartungsintensiv, wenn man jetzt wirklich 100 % Testabdeckung mit Unit-Tests hat, das funktioniert nicht aber es gibt viele Teile, die man testen kann. Zum Beispiel irgendwelche Service-Klassen mit kleinen Methoden, die kann man relativ gut testen mit wenig Aufwand. Und die helfen tatsächlich so etwas wie lose Kopplungen einzuhalten und saubere Trennung von Verantwortlichkeiten, deswegen würde ich das noch empfehlen, obwohl man das meistens aus Zeitgründen hinten runterfallen lässt. Ich bin da auch leider manchmal davon betroffen. Statische Code-Analyse, so etwas wie CodeSniff sollte man einsetzten und einbinden. Und man sollte, wenn man mit TYPO3 arbeitet sich die TYPO3-Coding-Guidelines anschauen. Dann gibt es noch Test-Guidelines, Security-Guidelines, das ist wichtig, dass man sich das mit anschaut.

2. **Welche Rolle spielen deiner Meinung nach statische Analyse-Tools bei der Verbesserung von Code-Qualität? Würdest du sagen, dass diese Tools in der Zukunft eine größere Rolle spielen werden?**

Ich kann mir schon gut vorstellen, dass das quasi immer schöner wird und besser anwendbar, das kann ich mir gut vorstellen. Dass man da auch mit KI hinter arbeitetet. Also, dass da dann noch tiefgreifender Vorschläge gemacht werden. Das kann ich mir echt super vorstellen. Das dauert bestimmt nicht mehr lange. Es wird auf jeden Fall nicht weniger, in der Vergangenheit hat man noch per FTP deployed und da wird einfach alles professioneller, das wird auch weiter so gehen. Ich glaube nicht, dass wir irgendwann einen Schritt rückwärtsgehen und wir in alte Muster verfallen.

3. **Fällt dir etwas für zukünftige Forschungen im Bereich der Code-Qualität ein? Gibt es Themen, zu denen du dir mehr Forschung wünschen würdest?**

Ja, vielleicht KI-Einbindung. Dass man noch mehr Vorschläge macht, wie zum Beispiel Quellcode, der über mehrere Seiten geht, also über mehrere Dateien, dass man da vielleicht

auch schon Vorschläge macht. Eventuell auch Hinweise auf Entwurfsmuster, das kann ich mir gut vorstellen. Habe ich auch schon darüber nachgedacht.

Anhang 2.6: Interview 6 – S

Hintergrundfragen

1. **Wie lange arbeitest du bereits mit TYPO3, insbesondere an der Entwicklung und Weiterentwicklung von Extensions?**

Ich glaube 13 Jahre.

2. **Als du mit der Entwicklung von TYPO3-Extensions begonnen hast, welche Bedeutung hatte die Code-Qualität für dich?**

Deutlich weniger als heute. Es war schon wichtig, den man gut versteht aber das hatte viel weniger mit Tools zu tun, also weniger Tools, mit denen man den Code analysiert hat. Es war eher so eine Gefühlssache, ob man den Code versteht oder nicht.

3. **Hat sich diese Bedeutung für dich im Laufe der Zeit verändert?**

Ja. Ist auf jeden Fall noch größer geworden. Auch dadurch, dass man mehr im Team arbeitet und da ist es noch wichtiger, dass alle ein einheitliches Verständnis von Code-Qualität haben. Und auch da unterschiedliche Levels von Leuten im Team sind, da hilft es, wenn man einen einheitlichen Rahmen hat.

4. **Benutzt du in deinem Alltag als Entwickler:inn statische Analyse-Tools um die Code-Qualität zu überprüfen? Wenn ja, welche?**

Ja. PHPStan ist gesetzt als statisches Analyse-Tool. Im Frontend Typescript. Ist ja eigentlich eine Sprache aber letztendlich macht sie ja das gleiche.

Code-Qualität und TYPO3 Extensions

1. **Auf welche Bereiche in der Softwareentwicklung (Beispiele: Testing, Einführung, Wartung/Pflege, Fehlerbehebung) wirkt sich deiner Erfahrung nach Software-Code-Qualität aus?**

Ich glaube ganz extrem auf den Wartungs-Bereich, weil Bugs damit reduziert werden. Es ist ein bisschen entspannter, wenn man solche Tools am Start hat. Der Testing-Bereich, wenn man Test-Driven arbeitet, dann fängt es relativ früh an. Wenn man das nicht macht und sich eher später überlegt wo man Tests drauf packt, dann ist es ein bisschen später im Zyklus. Aber theoretisch kann es auf den gesamten Zyklus Auswirkungen haben. Es fängt schon in der Konzeptions-Phase an, in der man sich überlegt welche Testfälle nachher erfolgreich sein müssen. Das hat jetzt weniger mit Codeanalyse zu tun aber mit Testing, Akzeptanz-Tests zum Beispiel.

2. **Welche Merkmale oder Kriterien sind aus deiner Erfahrung besonders wichtig, um die Code-Qualität von TYPO3 Extensions zu bewerten?**

Das ist ein großes Thema. Ich glaube das sind mehrere Layer übereinander. Einmal die Basics wie Einrückungen, Standard-PHP-Qualitätskriterien, weil das einfach gerade im Team Reibungsverluste verhindert, also, wenn sich alle auf eine Baseline einigen, dann kann man sich mit den wirklichen Problemen und den wirklichen Herausforderungen im Projekt beschäftigen. Also sich auf das inhaltliche konzentrieren ohne sich über Tabs oder Spaces zu

streiten. Und dann gibt es darüber auch Standards für TYPO3-Extensions, also zum Beispiel die Ordnerstruktur, wo liegt was im Projekt, wie ist etwas benannt, welcher Teil einer Extension greift auf einen anderen Teil einer Extension zu, ist die Konfiguration getrennt vom Code, solche Standards finde ich auch total wichtig.

3. **Hast du schon mal mit fremdem Code gearbeitet, der in deinen Augen von schlechter Qualität war? Wenn ja, was genau hast du an diesem Code als schlecht empfunden?**

Ja. Das sind ganz unterschiedliche Aspekte. Das fängt zum Beispiel an mit Namings, dass Variablen- oder Funktions-Namen nicht so benannt sind, dass man versteht was da eigentlich passiert und meistens sind dann auch keine Kommentare vorhanden, die das fixen könnten. Also im Idealfall würde ich sagen sind die Variablen so benannt, dass man gar keine Kommentare braucht aber wenn sie nicht so benannt sind, dann würde ich auch Kommentare nehmen aber die sind dann meistens auch nicht da. Das heißt es ist wirklich eine Verständnis-Frage, was auch häufig ist, ist, dass die Struktur entweder von Anfang an nicht ordentlich bedacht wurde, also welcher Teil einer Anwendung gehört in welches Modul oder es ist historisch gewachsen und niemand hat zwischendurch Zeit oder Budget gehabt um das einmal zu refactoren, so dass man wieder versteht, was der aktuelle Stand tut und nicht der Stand von vor 5 Jahren. Das sind so die größten Probleme. Für die Einrückungen und so gibt es ja Tools, die man drüber laufen lassen kann aber das große Problem ist, dass man nicht versteht, was damit erreicht werden sollte. Welche Features stecken in diesem Ding mit drin, was muss nachher funktionieren. Die Business-Logik, wenn man die nicht versteht, dann wird es sehr schwer.

   a. **Würdest du sagen, dass dieser schlechte Code sich auf deine Motivation ausgewirkt hat, weiter an diesem Code zu arbeiten?**

Ja, auf jeden Fall. Ich glaube es kann auch motivierend sein, wenn man die Zeit hat sowas dann mal neu zu machen. Aber was so ein bisschen problematisch daran ist, ist, wenn man nicht weiß, was man nachher testen muss, was nachher noch funktionieren muss. Wenn der Code getestet ist, dann kann der Code auch von schlechter Qualität sein, wenn ein gutes Set an Tests da ist. Dann kann man den Code ja relativ einfach refactoren und weiß danach ob man was Großes kaputt gemacht hat oder nicht. Wenn das aber nicht der Fall ist, stochert man im Dunkeln und man weiß nicht, ob das was man umgebaut hat das tut, was es vorher getan hat.

   b. **Hast du Zeit investiert, um diesen schlechten Code aufzuarbeiten und zu verbessern? Wenn ja, wie viel Zeit hat es dich gekostet? (Ungefähre Angabe in Relation zur eigentlichen Aufgabe)**

Auf jeden Fall. Teilweise Wochen.

   c. **Gibt es bestimmte Merkmale von schlechtem Code, die dir immer wieder begegnet sind, auch in anderem Code?**

Ja ich glaube es ist häufig wirklich nicht sprechender Code. Also, dass die Funktionen komisch benannt sind, Abkürzungen als Variablennamen, irgendwelche kryptischen Namen, dass man nicht weiß, was passiert. Was auch häufig passiert ist, dass irgendwelche Kommentare geschrieben wurden, die aber gar nicht mehr dem aktuellen Stand des Codes entsprechen und dann weiß man nicht, ob der Kommentar dem entspricht, was passieren soll oder der Code.

Also, wenn es eine Differenz zwischen Kommentar oder Dokumentation gibt. Das ist total ätzend. Diese ganzen Basics, dafür gibt es mittlerweile Tools. Ich glaube was vielleicht noch so ein Ding ist, sind Altlasten, wo man nicht weiß ob die da einfach nur seit 10 Jahren drinstehen und gar nicht mehr benutzt werden oder ob man das ein Bug ist und man das reparieren muss. Gerade bei TYPO3-Extensions gibt es relativ viel, was über die Jahre deprected wurde und dann einfach nicht mehr funktioniert hat und wenn man sowas auf den Tisch kriegt und das da drinsteht, weiß man nicht, ob es ein Bug ist und es wurde nur vergessen es neu zu machen oder wurde entschieden, dass es nicht mehr gebraucht wird und es wurde nicht rausgeworfen.

4. **Hast du bereits die Erfahrung gemacht, dass du in einem Projekt unter großem Zeitdruck TYPO3 Extensions entwickeln oder erweitern musstest?**

Ja, tatsächlich nicht häufig aber das gab es schon. Wir hatten meistens unsere Kunden und unsere Deadlines ganz gut unter Kontrolle. Aber es gab so ein paar einzelne Fälle, wo das tatsächlich der Fall war.

   a. **Wie groß ist die Motivation unter diesem Zeitdruck auf eine gute Code-Qualität zu achten?**

Sie sollte groß sein. Aber man ist dann schon versucht shortcuts zu nehmen. Ich glaube auch in den Workflows, wenn man zum Beispiel einen hat, wo ab einem gewissen Punkt die Code-Qualität automatisiert kontrolliert wird oder irgendwelche Tests abgefahren werden und wenn dann ein Test nicht zuverlässig funktioniert, schaltet man den dann eher mal ab als sich anzuschauen, warum er nicht zuverlässig funktioniert. In der Situation passiert es schon mal, dass man Abstriche in der Code-Qualität macht, um eben die Deadline zu halten. Und noch schlimmer ist es danach, dass man es zum Beispiel vergisst oder wieder keine Zeit dafür ist das zu reparieren, was man sich als „Technische Schuld" in sein Projekt reingeholt hat.

   b. **Wie würdest du die Wichtigkeit von Code-Qualität in Projekten ein kategorisieren?**

Generell ist es natürlich total wichtig, weil es in Zukunft Bugs vermeidet. Dazu führt, dass die Extension Upgrade-fähiger wird. Also eigentlich ist es eine Investition in die Zukunft, wenn man das durchzieht. Und wenn man da auch immer am Ball bleibt und die Standards immer wieder erhöht mit dem Standard, den die Community vorgibt. In der Praxis funktioniert es gerade bei Kundenprojekten meistens nicht so gut. Was mir immer ganz wichtig war, sobald etwas open-source, ich habe das meiste Augenmerk auf Code-Qualität bei open-source-Extensions gelegt, weil das eben auch andere nutzten sollen. Weil es eine generische Lösung sein soll und nicht ein spezifischer Fall für den Kunden. Da habe ich mir dann auch die Zeit genommen wirklich viel Energie reinzustecken, ganz viel auf Code-Qualität zu achten, ganz viele automatisierte Prozesse einzuführen. Damit die CI mir auf die Finger haut, wenn ich irgendetwas mache, was ich nicht machen sollte. Und eben auch alle anderen, die daran beteiligt sind.

5. **Welche Maßnahmen oder Praktiken können dabei helfen von Anfang an guten Code zu schreiben?**

Ganz früh im Projekt schon Standards definieren. Das fängt an mit Editor-Konfig, also in welcher Datei wird was, wie eingerückt, das sollte man ganz am Anfang definieren, weil sonst gibt es relativ schnell Chaos. Und eben auch Linting-Tools einbauen, statische Code-Analyse und so weiter. Und das eben nicht nur optional machen, sondern verpflichtend. Das heißt,

wenn man relativ früh anfängt mit Merge Requests zu arbeiten im Projekt und da die CI sagt, es sollen alle Regeln geprüft werden, wenn nicht, dann darf da nicht rein gemerged werden. Das führt dazu, dass nie etwas rein gemerged wird, was schlechte Code-Qualität hat. Zumindest von diesem automatisiert prüfbaren Standard. Und es führt auch gleichzeitig dazu, dass man beim Code-Review dann auf andere Dinge achtet, weil der Code schon in einer höheren Qualität ankommt und dann kann man sich auf die Sachen konzentrieren, die eine Maschine nicht prüfen kann. Und dadurch werden die „higher-level" Sachen besser, also Code-Architektur, Naming-Things, das wird alles dadurch besser, dass man sich nicht mit den Kleinigkeiten beschäftigen muss.

Statische Software-Analyse-Tools und Code-Qualität

1. **Welche Aspekte von Code-Qualität können deiner Erfahrung nach statische Analyse-Tools in TYPO3 Extensions erfassen und bewerten?**

Der Klassiker sind Typen. Wenn man zum Beispiel nicht merkt, dass man an einer bestimmten Stelle einen Integer rein gibt aber eigentlich ein String erwartet wird und so etwas. Wobei das mittlerweile auch in die Sprache reingewandert ist, die Typensicherheit. Aber speziell als es vor ein paar Jahren noch nicht so war, war das enorm hilfreich solche Tools zu haben, die einem sagen, obwohl die Sprache das einen tun lässt. Da gibt es bestimmt noch viel mehr, was mir gerade nicht einfällt aber das sind so die klassischen Fehler, die einem halt häufig auf die Füße fallen, wenn man PHPStan oder so verwendet, dass es auf die Typen hinweist oder nach Typen fragen, Typescript ist da auch ganz penetrant, wenn man da keine Typen angibt, dann haut er einem auf die Finger. Das ist auf jeden Fall hilfreich.

   a. **Würdest du sagen, dass die genannten Aspekte, die für dich eine gute Code-Qualität ausmachen durch so ein statisches Analyse-Tool bewertet werden können?**

Teile davon. Gerade so eine Baseline, um eine Baseline zu schaffen. Automatisierte Tools können halt nicht entscheiden, ob zum Beispiel die Variable gut benannt ist. Oder ob der Code gut strukturiert ist auf einem höheren Level. Ob die Business Logik getrennt ist von der Ausgabe. Das ist etwas, was man besser prüfen kann als Mensch, weil man sich nicht mehr mit dem ganzen Kleinkram beschäftigen muss. Als Mensch hat man halt auch nur eine gewisse Anzahl von Dingen, die man gleichzeitig beachten kann und wenn man den Wald vor lauter Bäumen nicht sieht, weil da irgendwelche Einrückungsfehler oder so sind, dann übersieht man auch die Sachen, die wirklich relevant sind. Und deswegen sind die Tools auf jeden Fall ein großer Bestandteil.

   b. **Gibt es auch Aspekte der Code-Qualität, von denen du denkst, dass sie nicht durch solche Tools zu erfassen sind? Wenn ja, welche?**

Naming z. B. und was ich vorher schon gesagt hatte, ob die Konfiguration getrennt ist von der Business-Logik. Also, dass nicht mitten im Quellcode eine Variable steht, die eigentlich Konfiguration sein müsste. Die den Code dann nicht so wiederverwendbar machen lässt. Wenn statische Werte an verschiedenen Stellen im Code verwendet werden, zum Beispiel ein definierter String, der überall im Code verwendet wird, den dann in eine Konstante auszulagern, um dann Flüchtigkeitsfehler zu vermeiden, solche Sachen sind glaube ich im Moment Sachen, die ein Mensch machen muss. Auch die Frage ob es eine Extension ist oder ob es lieber zwei Extensions sein sollten, die getrennt werden müssten. Wie viel oder wie wenig

gehört in eine Klasse rein. Was soll abstrahiert werden, was vielleicht auch nicht. Es gibt zum Beispiel mal Code, den man an 5 Stellen verwendet und den will man dann vielleicht an einer Stelle zusammenfassen, um den nicht zu wiederholen, es gibt aber auch Stellen, an denen das keinen Sinn macht, an denen es besser wäre das an 5 Stellen zu lassen. Das sind glaube ich Entscheidungen, die so ein Tool nicht treffen kann, weil man da Programmier-Erfahrung braucht. Vielleicht geht das in Zukunft irgendwann mit KI aber im Moment habe ich das Gefühl, dass es da einen Menschen braucht. Und ich glaube auch die Kommunikation dazu, also sich mit einer anderen Person darüber austauschen. Was ich in Code-Reviews auch häufig gemacht habe ist Fragen zu stellen. Rückfragen stellen, fragen ob etwas so richtig ist und dann denkt die Person noch mal darüber nach und dann entscheidet man entweder, dass es richtig ist und manchmal entscheidet man sich für eine andere Lösung. Da gehört auch eine gewisse Kommunikation dazu. Also nicht nur Sachen anzukreiden, wie das so ein Tool eben macht, sondern auch zu entscheiden, ob es wirklich relevant ist in dem Fall oder nicht.

2. **Welche Vor- und Nachteile siehst du bei der Verwendung von statischen Analyse-Tools?**

Das spart quasi Brainpower für die wirklich relevanten Sachen, die eben ein Tool nicht machen kann. Das heißt man hat dann schon eine Baseline im Projekt, über die man einfach nicht mehr diskutieren muss, die einfach gesetzt ist. Nachteil ist so ein bisschen, dass man manchmal gegen diese Tools kämpft. Das ist zumindest so meine Erfahrung, also einerseits, deswegen, weil man sich in einem Framework bewegt, was es schwierig macht bestimmte Regeln durchzusetzen, wenn zum Beispiel das Framework bestimmte Regeln nicht erfüllt, wird es schwierig im eigenen Code die Regeln zu erfüllen, wenn man mit diesem Framework interagiert. Manchmal wirken die Regeln auf mich auch ein bisschen sehr verkopft und führen aus meiner Sicht dazu, darüber könnte man aber auch wahrscheinlich diskutieren, dass der Code weniger lesbarer wird für einen Menschen. Und ich denke mir immer, dass ich diesen Code ja nicht für eine Maschine, sondern für einen Menschen schreibe. Und wenn es z. B. eine Regel gibt, dass ich viele Kommentare schreiben muss, damit der Computer zufrieden ist wie der Code aussieht und nachher ein Mensch den Code weniger versteht, habe ich auch nichts gewonnen. Es ist immer eine Abwägungssache. Deswegen bin ich auch bei den Tools nicht am Ende von der Skala, man kann die ja unterschiedlich konfigurieren. Ich bin eher so im Mittelfeld bei der Konfiguration, weil es mir irgendwann zu viel Overhead wird im Quellcode tausend Sachen hinzuzufügen, die für Menschen zwar auch interessant sind aber es halt komplizierter machen insgesamt.

3. **Hast du schon einmal erlebt, dass ein statisches Analyse-Tool Verbesserungsvorschläge für deinen Code gemacht hat?**

Regelmäßig, wahrscheinlich täglich.

4. **Welche Rolle spielen Analyse-Tools bei der der Code-Qualität in TYPO3-Extensions über die Zeit hinweg?**

Auf jeden Fall. Man hat es schon in den letzten Jahren gemerkt, dass der Trend ganz klar in die Richtung geht. Und es wird wahrscheinlich noch viel mehr werden, wenn man jetzt so an KI denkt, wenn man das jetzt noch da drauf wirft. Es ist jetzt die Frage, ob sie am Anfang zuverlässig wären aber ich glaube so mit der Zeit werden die noch ganz andere Probleme finden im Vergleich zu jetzt. Weil jetzt sind sie halt wirklich sehr deterministisch, also man weiß genau jedes Mal was das Ding ausspuckt, weil halt irgendjemand sich genau diese Regeln

ausgedacht hat. Aber wenn man irgendwann mal der KI sagt sie kann sich ihre Regeln selbst ausdenken und ihr ganz viel Code gibt, guten und schlechten Code und die KI soll dann entscheiden wie gut der Code ist, den sie bekommt, das ist glaube ich noch mal ein viel größeres Potential. Da muss man aber auch mehr Erfahrung mitbringen als jetzt, ich glaube im Moment kann man die Regeln in vielen Fällen stumpf abarbeiten oder sogar den Computer umsetzten lassen, es gibt ja häufig auch so ein fix-Kommando. Da wäre ich in Zukunft skeptischer, wenn das Ding sich seine Regeln selber ausdenkt, da muss man glaube ich noch mit ein bisschen mehr Fingerspitzengefühl rangehen. Auf jeden Fall wird es zunehmen.

Empfehlungen und Schlussfolgerungen

1. **Auf Basis deiner Erfahrungen mit TYPO3 Extension-Programmierung. Welche Empfehlungen würdest du Entwickler / Entwicklerin geben, um die Code-Qualität zu verbessern?**

Ich würde mir ein paar von den großen und bekannten Extensions anschauen. Der Klassiker war immer Georg Ringer „news“, wo man mal reinschauen kann aber auch vielleicht so ein paar kleinere aber welche, die vielleicht eher für die neueren Versionen gebaut wurden, also die frisch gebaut wurden. Da habe ich es selbst gemerkt, als ich meine Extension gebaut habe, dass ich da ganz viele von den Altlasten gar nicht mehr drin hatte. Das heißt, du hast dann eine Extension, die nur mit 12 kompatibel ist. Und damit hat man nicht die tausend alten Sachen drin, um mit 11 oder 10 kompatibel zu sein, das kann auch interessant sein. Wenn man jetzt frisch reingeht und vielleicht sogar das Glück hat nicht mit alten Projekten zu tun zu haben. Das ist vielleicht auch mal ganz hilfreich sich nur die neuen Sachen anzuschauen. Ansonsten die Dokumentation lesen, wenn man etwas nicht versteht, die ist mittlerweile ja auch deutlich besser. Ich glaube vieles sollte man sich aber auch bei anderen abschauen und verbessern nach und nach.

2. **Fällt dir etwas für zukünftige Forschungen im Bereich der Code-Qualität ein? Gibt es Themen, zu denen du dir mehr Forschung wünschen würdest?**

Vielleicht das Setup von solchen Tools, also, dass das ein bisschen vereinheitlicht wird und einfacher wird. Ich hatte jetzt neulich mal so ein bisschen experimentiert mit so einem Tool, bei dem man ein Github-Repo übergeben konnte und es sollte alles eingerichtet werden. Weil das war zumindest bei mir immer noch ein manueller Prozess, sich zu überlegen was man checken möchte, was die Coding-Standards sind, es gibt von TYPO3 mittlerweile auch schon ein Repo, dass man einbinden kann aber das ist noch relativ minimalistisch, was da so mitkommt aber quasi so ein One-Click-Install „Ich möchte jetzt die Code-Qualität prüfen für meinen Code“. Ich glaube das könnte noch ein bisschen besser werden, da hat jeder noch sein eigenes Setup. Speziell jetzt auch im TYPO3-Bereich. Aber wahrscheinlich ist das große Thema an dieser Stelle, das ist relativ naheliegend, zu schauen, wie die KI uns da helfen kann mit den ganz eigenen Problemen, die mit sich kommen, dass das Ding sich Sachen ausdenkt, die nicht so schlau sind. Das wird man dann sehen.

Abschluss

1. **Fehlt deiner Meinung nach ein Themenbereich oder Aspekte, die ich fragen sollte?**

Ja, also über Testing haben wir jetzt nicht gesprochen. Da gibt es ja tausend verschiedene Aspekte, da ist aus meiner Sicht auch so der Setup-Aspekt das relevante. Also für die Unit-Tests

geht es mittlerweile ja ganz gut, das war vor ein paar Jahren deutlich anstrengender, so Unit-Tests oder Functional-Tests im TYPO3-Universum zu machen. Das ist mittlerweile deutlich besser. Ein großer Brocken ist noch das Akzeptanztest-Thema. Das wird besser aber das ist noch ein ordentlicher Aufwand und da gibt es bisher noch wenig Best-Practices, wie man das macht. Da kann die Hürde noch etwas niedriger werden. Das ist auf jeden Fall noch ein wichtiger Faktor was die Code-Qualität angeht, speziell wenn es um Refactoring geht.

Anhang 3: Kategorie-Leitfaden – Ebene 1

Code	Kategorie	Definition	Ankerbeispiel
k1	Code-Qualität	Was sagen die Interview-Personen über den allgemeinen Begriff der Code-Qualität aus? Welche Begriffe erklären sie? Wie beschreiben sie Code-Qualität? Welche Bedeutung hat die Code-Qualität für sie persönlich? Welche Gründe werden für schlechten Software-Code genannt?	„Also auf der Hand liegt natürlich, dass es sich vor allem in der Wartung auswirkt und in der Erweiterbarkeit, also wenn der Kunde neue Wünsche hat oder es gibt von außen irgendwelche Änderungen, also Schnittstellenänderungen, an die wir noch mal ranmüssen. Das ist wahrscheinlich mit das wichtigste.“ - A
k2	Merkmale und Kriterien von Code-Qualität	Welche Merkmale werden genannt, die Software-Code-Qualität beschreiben? Sowohl positive, als auch negative Aspekte. Werden Design-Patterns oder Prinzipien des Clean Code Development genannt?	„Unklare Rückgabewerte von Methoden und Argumente von Methoden sind auch ein häufiges Problem. Das ist etwas, das PHP sehr lange erlaubt hat, dass man alles Mögliche zurückgeben kann, das ist nicht streng typisiert.“ - D
k3	Erfahrungen und Herausforderungen	Mit welchen Herausforderungen und Schwierigkeiten haben die Befragten Personen bei der Entwicklung von TYPO3-Extensions zu kämpfen? Sowohl den Software-Code betreffend, als auch den Bereich von Analyse-Tools.	„In bestimmten Situationen vielleicht. In anderen Situationen, wenn man zum Beispiel mit Legacy Projekten arbeitet ist es eher auch motivierend aus etwas Altem etwas tolles neues und Modernes zu machen. Also es macht schon Spaß auch etwas zu modernisieren. Es kann auch schon motivierend sein.“ - M
k4	Qualitätssicherung	Wie kann Software-Code-Qualität sichergestellt werden? Welche Maßnahmen können ergriffen werden? Sowohl vorbeugende, als auch analytische und auch organisatorische Maßnahmen sollen erfasst werden. Welche Empfehlungen werden gegeben?	„Ganz früh im Projekt schon Standards definieren. Das fängt an mit Editor-Konfig, also in welcher Datei wird was, wie eingerückt, das sollte man ganz am Anfang definieren, weil sonst gibt es relativ schnell Chaos. Und eben auch Linting-Tools einbauen, statische Code-Analyse und so weiter. Und das eben nicht nur optional machen, sondern verpflichtend.“ - S
k5	Analyse-Tools	Wie schätzen die befragten Personen die Bedeutung von Analyse-Tools ein? Welche Merkmale können damit erfasst werden, welche nicht? Welche Tools werden genutzt? Wie schätzen die Interviewten die Zukunft von Analyse-Tools ein?	„Also Vorteil definitiv, dass man eine externe Referenz bekommt, dass man nicht auf persönliche Einschätzung angewiesen ist, sondern eine Basis bekommt, um gemeinsam über Code-Qualität zu sprechen.“ - D

Quelle: Eigene Darstellung

Anhang 4: Kategorie-Leitfaden – Ebene 2: Kategorie 1

HK	Code	Kategorie	Definition	Ankerbeispiel
k1	k11	Auswirkungen von Code-Qualität	Welche Auswirkungen von Code-Qualität werden von den Interview-Personen genannt? Das können sowohl positive, als auch negative sein.	„Also auf der Hand liegt natürlich, dass es sich vor allem in der Wartung auswirkt und in der Erweiterbarkeit, also wenn der Kunde neue Wünsche hat oder es gibt von außen irgendwelche Änderungen, also Schnittstellenänderungen, an die wir noch mal ranmüssen. Das ist wahrscheinlich mit das wichtigste.“ - A
k1	k12	Bedeutung Software-Code-Qualität	Wie wird die Bedeutung der Code-Qualität von den befragten Personen zum Ausdruck gebracht? Wie hat sich die Bedeutung für die Befragten im Laufe der Zeit entwickelt?	„Ich messe der Sache hohe Priorität bei, das ist wichtig, gerade bei langwierigen Projekten, wie wir sie haben. Weil der Quellcode muss halt verständlich sein, es kommen neue Entwickler ins Projekt oder es gehen auch mal Entwickler aus dem Projekt raus, also wir haben eine ständige Rotation und da ist es wichtig, dass die Code-Qualität stimmt. Schon hoch würde ich jetzt sagen.“ - M
k1	k13	Gefühlte Code-Qualität	Welche Aussagen treffen auf eine subjektive Wahrnehmung von Code-Qualität zu? Erwähnen die befragten Personen eine gefühlte Wahrnehmung? Ist diese Wahrnehmung messbar?	„Also für mich hat sich im Laufe der Jahre herausgestellt, dass wenn man eine Extension aufmacht, dass man da sofort ein Gefühl für bekommt, ob die Extension modern ist und sich an Paradigmen und Konventionen hält oder eben nicht.“ - A
k1	k14	Gründe für schlechte Code-Qualität	Welche Gründe werden für eine bestehende schlechte Code-Qualität genannt?	„Oft ist es so, dass Aufgaben, die man mit einem gut getesteten third-class-pakage lösen könnte, aus Unwissenheit oder Bequemlichkeit, mal eben schnell gemacht werden und dann die Lösung quasi einerseits eine niedrige Umsetzungsqualität hat und andererseits der Komplexität des Problems nicht gerecht wird, weil es nur einen bestimmten Aspekt abdeckt und dann wird es sehr wackelig.“ - D

Quelle: Eigene Darstellung

Anhang 5: Kategorie-Leitfaden – Ebene 2: Kategorie 2

HK	Code	Kategorie	Definition	Ankerbeispiel
k2	k21	Merkmale guter Software-Code-Qualität	Welche Merkmale und Kriterien werden von den Interview-Personen genannt, die einen guten Software-Code beschreiben?	„Lesbarkeit, Verständlichkeit, das ist natürlich sehr wichtig, dass ein Code zuverlässig ist. Wiederverwendbarkeit, ist natürlich ein Thema. Ich denke auch das Thema Performance, Effizienz ist wichtig.“ - M
k2	k22	Merkmale schlechter Software-Code-Qualität	Welche Merkmale und Kriterien werden von den Interview-Personen genannt, die einen schlechten Software-Code beschreiben?	„Und grundsätzlich, was ich auch im TYPO3-Kontext sehr oft erlebe und was ich für eine schwache Code-Qualität halte, ist, wenn Klassen sehr tiefe Vererbungen haben und sehr viele Abhängigkeiten.“ - D
k2	k23	Clean Code	Wird der Begriff Clean Code erwähnt? Werden die Prinzipien des Clean Code Development genannt?	„Die ganzen Prinzipien, wie "KISS" und Single-Responsible-Prinzip und diesen ganzen Clean Code-Prinzipien.“ - M
k2	k24	neutrale Aussagen über Code-Qualität	Welche Aussagen zu Code-Qualität können nicht eindeutig guter oder schlechter Merkmale zugeordnet werden? Aussagen, die neutral klingen oder hinterfragend sind, ob das die Code-Qualität wirklich verbessert.	„Da hat jeder so seine eigene Art. Also es gibt eine Kollegin von mir da weiß ich, ihr ist besonders wichtig, das automatisiertes Testen am Start ist. Was aber auch nur bedingt aussagt, wie gut der Code ist, teilweise schon, aber schwierig.“ - A

Quelle: Eigene Darstellung

Anhang 6: Kategorie-Leitfaden – Ebene 2: Kategorie 3

HK	Code	Kategorie	Definition	Ankerbeispiel
k3	k31	Erfahrungen mit Zeitdruck	Welche Erfahrungen haben die befragten Personen bezüglich Zeitdruck in Projekten gemacht im Kontext mit TYPO3-Extension-Entwicklung?	„Ich persönlich, ja klar. Wir hatten erst letztes Jahr so einen Fall bei einem Projekt, wo wir wirklich in relativ kurzer Zeit etwas entwickeln mussten, was relativ komplex war. Und die Entwickler habe das dann auch gemacht.“ - M
k3	k32	Schwierigkeiten Code-Qualität	Auf welche Schwierigkeiten sind die befragten Personen beim Einhalten von Code-Qualität (oder dem Versuch) gestoßen?	„Also, wenn zum Beispiel irgendwas gelöscht werden soll und dann aber in der Funktion Sachen hinzugefügt werden, dann ist das natürlich unproduktiv und für die Qualitätssicherung auch nicht gut und der Kollege kommt damit eigentlich auch nicht klar und müsste eigentlich bei dir nachhaken. […]“ - I
k3	k34	Erfahrung mit schlechtem Code	Haben die befragten Personen bereits Erfahrung mit schlechtem Code gemacht. Das kann sowohl fremden Code, als auch eigenen betreffen.	„Natürlich ist es aber auch immer eine Frage mit Zeit und Budget. Wenn ich es besser machen kann, dann investiere ich auch schon mal eine Zusatzstunde oder auch zwei, ohne dass der Kunde bezahlt. Das ist einfach so ein eigener Anspruch, wenn man solche Baustellen sieht.“ - M
k3	k35	Erfahrung mit TYPO3 in Jahren	Wie viele Jahre Erfahrung haben die befragten Personen mit TYPO3-Extensions?	„Lass mich mal kurz überlegen. Ich meine seit 2009. Das wären jetzt irgendwie 14 Jahre.“ - T

Quelle: Eigene Darstellung

Anhang 7: Kategorie-Leitfaden – Ebene 2: Kategorie 4

HK	Code	Kategorie	Definition	Ankerbeispiel
k4	k41	Konstruktive Qualitätssicherung	Welche Maßnahmen werden genannt, um die Code-Qualität in konstruktivem Sinne, also a priori zu schaffen?	„Ganz früh im Projekt schon Standards definieren. Das fängt an mit Editor-Konfig, also in welcher Datei wird was, wie eingerückt, das sollte man ganz am Anfang definieren, weil sonst gibt es relativ schnell Chaos.“ - S
k4	k42	Analytische Qualitätssicherung	Welche Maßnahmen werden genannt, um die Code-Qualität in analytischem Sinne, also a posteriori zu schaffen?	„Test-Driven-Development ist natürlich eine gute Sache. Wenn man statische Code Analysen in seine Projekte und in den Workflow einbaut, wenn ich schon beim Commiten die Rückmeldung bekomme, dass da ein Fehler drin ist, das hilft mir natürlich enorm.“ - A
k4	k43	Organisatorische Qualitätssicherung	Werden Maßnahmen genannt, die Auswirkungen auf die Code-Qualität haben und organisatorischer Natur sind?	„Budget dafür einplanen. Dass man sich sagt, dass man eine bestimmte Prozentzahl davon nimmt um dann noch mal auf diese Sachen zu achten und dann auch wirklich Tests mitzuschreiben.“ - I
k4	k45	Empfehlungen	Welche Empfehlungen geben die Interview-Personen um die Software-Code-Qualität zu verbessern oder im Vorhinein sicherzustellen? Empfehlungen zur Qualitätssicherung.	„Also ich finde das sehr schwierig, weil jeder Programmierer, dem man sagt, dein Code ist Mist, sich beleidigt fühlt, zu Recht. Man muss lernen, dass in Begriffe zu packen oder das auch in ein Kategorien-System zu bringen, Bewertungsmaßstäbe, die man selbst akzeptieren kann und die anderen auch akzeptieren können.“ - D

Quelle: Eigene Darstellung

Anhang 8: Kategorie-Leitfaden – Ebene 2: Kategorie 5

HK	Code	Kategorie	Definition	Ankerbeispiel
k5	k51	erfassbare Merkmale von Analyse-Tools	Welche Merkmale sind laut der Befragten durch statische Analyse-Tools erfassbar?	„So eine statische Analyse hilft einem schon bei generellen Dingen. Es zeigt dir teilweise die Komplexität von Klassen und Funktionen. Allerdings auch nur bedingt.“ - A
k5	k52	nicht erfassbare Merkmale von Analyse-Tools	Welche Merkmale sind laut der Befragten nicht durch statische Analyse-Tools erfassbar?	„Verständlichkeit. Da gibt es zwar auch Metriken, mit denen man Verständlichkeit einschätzen kann. Aber ob das tatsächlich dann so ist, also ob die Metrik dann stimmt oder nicht hängt extrem davon ab in welchem Umfeld der Code gerade ist.“ - D
k5	k53	genutzte Analyse-Tools	Welche statischen Analyse-Tools werden von den Interview-Personen genutzt? Werden überhaupt Tools genutzt?	„Also ich verwende „PHP Mess Detector“, dann gibt es im „PHPUnit“ die Komplexitätsanalyse, das verwende ich in der Regel und ich habe verschiedene Tools von unterschiedlichen Anwendern ausprobiert, „codecov“, z. B. war eins womit ich längere Zeit gearbeitet habe. Und am glücklichsten war ich mit „SonarQube“, was als Dienst für Open Source frei ist, also man es solange benutzten, solange das Repository public ist.“ - D
k5	k54	Bedeutung von Analyse-Tools	Welche Bedeutung haben Analyse-Tools für die Befragten? Wie schätzen sie die Bedeutung ein?	„Ich kann ja noch so viel statische Code-Analyse draufschmeißen, das hilft mir zum Teil, dass ich es besser verstehe was da passiert ist aber es kann natürlich trotzdem Müll sein am Schluss.“ - A
k5	k55	Zukunft Analyse-Tools	Wie schätzen die befragten Personen die Zukunft von statischen Analyse-Tools ein? Welche Ideen oder Vorstellungen haben sie von zukünftiger statischer Analyse?	„Ja, vielleicht KI-Einbindung. Dass man noch mehr Vorschläge macht, wie zum Beispiel Quellcode, der über mehrere Seiten geht, also über mehrere Dateien, dass man da vielleicht auch schon Vorschläge macht. Eventuell auch Hinweise auf Entwurfsmuster, das kann ich mir gut vorstellen. Habe ich auch schon darüber nachgedacht.“ - M
k5	k56	Schwierigkeiten	Welche Schwierigkeiten gibt es bei der Nutzung oder Einrichtung von statischen Analyse-Tools? Oder andere Schwierigkeiten in diesem Kontext?	„Dass es frustriert, weil man nichts angehen kann. Oder dass man um die Ohren gehauen bekommt, dass der Code ganz schlecht ist und man ihn refactoren müsste und man hat nie Zeit dafür.“ - T

Quelle: Eigene Darstellung

Anhang 9: PHPStan PHP-Analyseregeln

erfassbare Merkmale EN	**erfassbare Merkmale DE**	**Bereich**
unknown classes, unknown functions, unknown methods called on $this	Unbekannte Klassen, Funktionen und Methoden	Unbekannter Code
wrong number of arguments passed to those methods and functions	falsche Anzahl von Argumenten, die an Funktionen/Methoden übergeben werden	Ein- und Ausstiege
always undefined variables	immer undefinierte Variablen	Verwendung von Variablen
possibly undefined variables	möglicherweise undefinierte Variablen	Verwendung von Variablen
unknown magic methods and properties on classes with __call and __get	unbekannte magische Methoden und Eigenschaften in Klassen, die __call und __get verwenden	„Magische“ Methoden
unknown methods checked on all expressions (not just $this)	Überprüft unbekannte Methoden in allen Ausdrücken, nicht nur $this	Unbekannter Code
validating PHPDocs	validiert PHPDocs	PHPDocs
return types, types assigned to properties	Validiert Rückgabetypen und den Typ von Eigenschaften	Typisierung
basic dead code checking - always false instanceof and other type checks, dead else branches, unreachable code after return; etc.	Erkennt grundlegende toten Code, wie immer falsche instanceof-Überprüfungen, toten else-Zweig und unerreichbaren Code nach einem Return	Dead Code
checking types of arguments passed to methods and functions	Überprüft die Typen der an Methoden und Funktionen übergebenen Argumente	Typisierung
report missing typehints	Meldet fehlende Typhints im Code	Typisierung
report partially wrong union types - if you call a method that only exists on some types in a union type, level 7 starts to report that; other possibly incorrect situations	Meldung, wenn eine Methode nur für einige Typen in einem Vereinigungstyp aufgerufen wird	Typisierung
report calling methods and accessing properties on nullable types	Meldet das Aufrufen von Methoden und den Zugriff auf Eigenschaften auf Nullable-Typen	Typisierung
be strict about the mixed type - the only allowed operation you can do with it is to pass it to another mixed	Erzwingt eine Strenge auf den Mixed-Typ, erlaubt nur eingeschränkte Operationen damit	Typisierung

Quelle: (Rule levels, o. D.)

Anhang 10: SonarQube PHP-Analyseregeln

Regel EN	Regel DE	Kategorie
Classes should not be coupled to too many other classes (Single Responsibility Principle)	Klassen sollten nicht mit zu vielen anderen Klassen gekoppelt sein (Single Responsibility Principle).	Abhängigkeiten
PHPUnit assertTrue/assertFalse should be simplified to the corresponding dedicated assertion	PHPUnit assertTrue/assertFalse sollten auf die entsprechenden dedizierten Assertions vereinfacht werden.	Best Practise
Character classes should be preferred over reluctant quantifiers in regular expressions	Zeichenklassen sollten in regulären Ausdrücken gegenüber zurückhaltenden Quantifizierern bevorzugt werden.	Best Practise
Jump statements should not be redundant	Sprunganweisungen sollten nicht überflüssig sein.	Best Practise
"catch" clauses should do more than rethrow	„catch“-Klauseln sollten mehr tun als nur erneut werfen.	Best Practise
"&&" and "||" should be used	„&&“ und „||“ sollten verwendet werden.	Best Practise
Boolean checks should not be inverted	Boolesche Überprüfungen sollten nicht invertiert werden.	Best Practise
"elseif" keyword should be used in place of "else if" keywords	Das Schlüsselwort „elseif“ sollte anstelle von „else if“ verwendet werden.	Best Practise
PHP keywords and constants "true", "false", "null" should be lower case	PHP-Schlüsselwörter und Konstanten „true“, „false“, „null“ sollten in Kleinbuchstaben sein.	Best Practise
Closing tag "?>" should be omitted on files containing only PHP	Der Schluss-Tag „?>“ sollte in Dateien, die nur PHP enthalten, weggelassen werden.	Best Practise
Only LF character (Unix-like) should be used to end lines	Nur das LF-Zeichen (Unix-ähnlich) sollte zum Beenden von Zeilen verwendet werden.	Best Practise
More than one property should not be declared per statement	Mehr als eine Eigenschaft sollte nicht pro Anweisung deklariert werden.	Best Practise
Local variables should not be declared and then immediately returned or thrown	Lokale Variablen sollten nicht deklariert und dann sofort zurückgegeben oder geworfen werden.	Best Practise
A "while" loop should be used instead of a "for" loop	Eine „while“-Schleife sollte anstelle einer „for“-Schleife verwendet werden.	Best Practise
Overriding methods should do more than simply call the same method in the super class	Überschreibende Methoden sollten mehr tun als nur dieselbe Methode in der Oberklasse aufzurufen.	Best Practise
"empty()" should be used to test for emptiness	„empty()“ sollte verwendet werden, um auf Leere zu prüfen.	Best Practise
Return of boolean expressions should not be wrapped into an "if-then-else" statement	Die Rückgabe von booleschen Ausdrücken sollte nicht in eine „if-then-else“-Anweisung eingewickelt werden.	Best Practise
Tabulation characters should not be used	Tabulationszeichen sollten nicht verwendet werden.	Best Practise

Class names should comply with a naming convention	Klassennamen sollten einer Namenskonvention entsprechen.	Best Practise
Track lack of copyright and license headers	Verfolgen des Mangels an Urheberrechts- und Lizenzkopfzeilen.	Best Practise
Octal values should not be used	Oktalwerte sollten nicht verwendet werden.	Best Practise
Literal boolean values and nulls should not be used in equality assertions	Wörtliche boolesche Werte und Nullen sollten nicht in Gleichheits-Assertions verwendet werden.	Best Practise
Parentheses should not be used for calls to "echo"	Klammern sollten nicht für „echo“-Aufrufe verwendet werden.	Best Practise
else if" constructs should end with "else" clauses	„else if“-Konstrukte sollten mit „else“-Klauseln enden.	Best Practise
Assignments should not be made from within sub-expressions	Zuweisungen sollten nicht innerhalb von Teil-Ausdrücken gemacht werden.	Best Practise
Framework-provided functions should be used to test exceptions	Framework-bereitgestellte Funktionen sollten zur Testung von Ausnahmen verwendet werden.	Best Practise
Unicode-aware versions of character classes should be preferred	Unicode-fähige Versionen von Zeichenklassen sollten bevorzugt werden.	Best Practise
Superglobals should not be accessed directly	Superglobals sollten nicht direkt zugegriffen werden.	Best Practise
Colors should be defined in upper case	Farben sollten in Großbuchstaben definiert werden.	Best Practise
String literals should not be concatenated	Zeichenkettenliterale sollten nicht verkettet werden.	Best Practise
"final" should not be used redundantly	„final“ sollte nicht überflüssig verwendet werden.	Best Practise
String literals should not be duplicated	Zeichenkettenliterale sollten nicht dupliziert werden.	Copy & Paste
Methods should not have identical implementations	Methoden sollten nicht identische Implementierungen haben.	Copy & Paste
Two branches in a conditional structure should not have exactly the same implementation	Zwei Zweige in einer bedingten Struktur sollten nicht genau die gleiche Implementierung haben.	Copy & Paste
Class constructors should not create other objects	Klassenkonstruktoren sollten keine anderen Objekte erstellen.	Dependency Injection
Methods should not be empty	Methoden sollten nicht leer sein.	Dokumentation
Perl-style comments should not be used	Perl-Style-Kommentare sollten nicht verwendet werden.	Dokumentation
Comments should not be located at the end of lines of code	Kommentare sollten nicht am Ende von Codezeilen stehen.	Dokumentation
Switch cases should end with an unconditional "break" statement	Switch-Fälle sollten mit einer bedingungslosen „break“-Anweisung enden.	Fehlervermeidung
Character classes in regular expressions should not contain only one character	Zeichenklassen in regulären Ausdrücken sollten nicht nur ein Zeichen enthalten.	Fehlervermeidung

Reluctant quantifiers in regular expressions should be followed by an expression that can't match the empty string	Zurückhaltende Quantifizierer in regulären Ausdrücken sollten von einem Ausdruck gefolgt werden, der nicht auf den leeren String passt.	Fehlervermeidung
Character classes in regular expressions should not contain the same character twice	Zeichenklassen in regulären Ausdrücken sollten nicht das gleiche Zeichen zweimal enthalten.	Fehlervermeidung
Assertion arguments should be passed in the correct order	Argumente von Assertions sollten in der richtigen Reihenfolge übergeben werden.	Fehlervermeidung
Reflection should not be used to increase accessibility of classes, methods, or fields	Reflection sollte nicht verwendet werden, um die Zugänglichkeit von Klassen, Methoden oder Feldern zu erhöhen.	Fehlervermeidung
Generic exceptions ErrorException, RuntimeException and Exception should not be thrown	Generische Ausnahmen ErrorException, RuntimeException und Exception sollten nicht geworfen werden.	Fehlervermeidung
A subclass should not be in the same "catch" clause as a parent class	Eine Unterklasse sollte nicht in derselben „catch“-Klausel wie eine Elternklasse sein.	Fehlervermeidung
Unnecessary parentheses should not be used for constructs	Unnötige Klammern sollten nicht für Konstrukte verwendet werden.	Formatierung
A conditionally executed single line should be denoted by indentation	Eine bedingt ausgeführte einzelne Zeile sollte durch Einrückung gekennzeichnet sein.	Formatierung
Conditionals should start on new lines	Bedingungen sollten auf neuen Zeilen beginnen.	Formatierung
Control structures should use curly braces	Kontrollstrukturen sollten geschweifte Klammern verwenden.	Formatierung
Superfluous curly brace quantifiers should be avoided	Überflüssige geschweifte Klammer Quantifizierer sollten vermieden werden.	Formatierung
Non-capturing groups without quantifier should not be used	Nicht erfassende Gruppen ohne Quantifizierer sollten nicht verwendet werden.	Formatierung
Multiline blocks should be enclosed in curly braces	Mehrzeilige Codeblöcke sollten in geschweiften Klammern eingeschlossen sein.	Formatierung
Method arguments with default values should be last	Methodenargumente mit Standardwerten sollten zuletzt stehen.	Formatierung
Redundant pairs of parentheses should be removed	Redundante Paare von Klammern sollten entfernt werden.	Formatierung
Collapsible "if" statements should be merged	Zusammenlegbare „if“-Anweisungen sollten fusioniert werden.	Formatierung
Source code should comply with formatting standards	Der Quellcode sollte den Formatierungsstandards entsprechen.	Formatierung
Lines should not end with trailing whitespaces	Zeilen sollten nicht mit abschließenden Leerzeichen enden.	Formatierung
Files should contain an empty newline at the end	Dateien sollten am Ende eine leere Zeile enthalten.	Formatierung
A close curly brace should be located at the beginning of a line	Eine schließende geschweifte Klammer sollte am Anfang einer Zeile stehen.	Formatierung

URIs should not be hardcoded	URIs sollten nicht fest codiert sein.	Formatierung
Statements should be on separate lines	Anweisungen sollten auf separaten Zeilen stehen.	Formatierung
An open curly brace should be located at the beginning of a line	Eine öffnende geschweifte Klammer sollte am Anfang einer Zeile stehen.	Formatierung
Classes should not have too many methods	Klassen sollten nicht zu viele Methoden haben.	großer Programmumfang
Functions should not have too many lines of code	Funktionen sollten nicht zu viele Codezeilen haben.	großer Programmumfang
Classes should not have too many lines of code	Klassen sollten nicht zu viele Codezeilen haben.	großer Programmumfang
"switch case" clauses should not have too many lines of code	„switch case“-Klauseln sollten nicht zu viele Codezeilen haben.	großer Programmumfang
Files should not have too many lines of code	Dateien sollten nicht zu viele Codezeilen haben.	großer Programmumfang
Files should contain only one top-level class or interface each	Dateien sollten jeweils nur eine Top-Level-Klasse oder -Schnittstelle enthalten.	keine klar abgegrenzten Aufgaben
Classes should not have too many fields	Klassen sollten nicht zu viele Felder haben.	keine klar abgegrenzten Aufgaben
Lines should not be too long	Zeilen sollten nicht zu lang sein.	Lesbarkeit
Regular expression quantifiers and character classes should be used concisely	Quantifizierer und Zeichenklassen in regulären Ausdrücken sollten prägnant verwendet werden.	Lesbarkeit
Constant names should comply with a naming convention	Konstantennamen sollten einer Namenskonvention entsprechen.	Naming
Local variable and function parameter names should comply with a naming convention	Namen von lokalen Variablen und Funktionsparametern sollten einer Namenskonvention entsprechen.	Naming
Field names should comply with a naming convention	Feldnamen sollten einer Namenskonvention entsprechen.	Naming
Interface names should comply with a naming convention	Interface-Namen sollten einer Namenskonvention entsprechen.	Naming
Method and function names should comply with a naming convention	Methoden- und Funktionsnamen sollten einer Namenskonvention entsprechen.	Naming
The names of methods with boolean return values should start with "is" or "has"	Die Namen von Methoden mit booleschen Rückgabewerten sollten mit „is“ oder „has“ beginnen.	Naming
Local variables should not have the same name as class fields	Lokale Variablennamen sollten nicht denselben Namen wie Klassenfelder haben.	Naming
File names should comply with a naming convention	Dateinamen sollten einer Namenskonvention entsprechen.	Naming
Track uses of "NOSONAR" comments	Verfolgen von „NOSONAR“-Kommentaren.	NOSONAR
PHP parser failure	PHP-Parserfehler.	Parse Fehler

`str_replace` should be preferred to `preg_replace`	str_replace sollte gegenüber preg_replace bevorzugt werden.	Performanz
Single-character alternations in regular expressions should be replaced with character classes	Einzeichen-Alternativen in regulären Ausdrücken sollten durch Zeichenklassen ersetzt werden.	Performanz
"php_sapi_name()" should not be used	„php_sapi_name()“ sollte nicht verwendet werden.	Performanz
Modifiers should be declared in the correct order	Modifier sollten in der richtigen Reihenfolge deklariert werden.	Richtlinie
"goto" statement should not be used	Die „goto“-Anweisung sollte nicht verwendet werden.	Struktur
"switch" statements should have at least 3 "case" clauses	„switch“-Anweisungen sollten mindestens 3 „case“-Klauseln haben.	Struktur
Files should not contain inline HTML	Dateien sollten keinen Inline-HTML-Code enthalten.	Struktur
Test class names should end with "Test"	Testklassennamen sollten mit „Test“ enden.	Testbarkeit
Tests should include assertions	Tests sollten Assertions enthalten.	Testbarkeit
TestCases should contain tests	Testfälle sollten Tests enthalten.	Testbarkeit
A reason should be provided when skipping a test	Ein Grund sollte angegeben werden, wenn ein Test übersprungen wird.	Testbarkeit
Test methods should be discoverable	Testmethoden sollten auffindbar sein.	Testbarkeit
Inheritance tree of classes should not be too deep	Die Vererbungshierarchie von Klassen sollte nicht zu tief sein.	tiefe Vererbung
"switch" statements should have "default" clauses	„switch“-Anweisungen sollten „default“-Klauseln haben.	Unklare Rückgabewerte
Functions should use "return" consistently	Funktionen sollten konsistent „return“ verwenden.	Unklare Rückgabewerte
Unused assignments should be removed	Nicht verwendete Zuweisungen sollten entfernt werden.	veralteter Code
"__construct" functions should not make PHP 4-style calls to parent constructors	Die Funktionen „__construct“ sollten keine Aufrufe im Stil von PHP 4 an übergeordnete Konstruktoren tätigen.	veralteter Code
PHP 4 constructor declarations should not be used	Konstruktor-Deklarationen von PHP 4 sollten nicht verwendet werden.	veralteter Code
Deprecated predefined variables should not be used	Veraltete vordefinierte Variablen sollten nicht verwendet werden.	veralteter Code
Sections of code should not be commented out	Codeabschnitte sollten nicht auskommentiert sein.	veralteter Code
Unused function parameters should be removed	Nicht verwendete Funktionparameter sollten entfernt werden.	veralteter Code
Unused "private" methods should be removed	Nicht verwendete „private“ Methoden sollten entfernt werden.	veralteter Code
Track uses of "FIXME" tags	Verfolgen von „FIXME“-Tags.	veralteter Code
Nested blocks of code should not be left empty	Verschachtelte Codeblöcke sollten nicht leer gelassen werden.	veralteter Code
Unused "private" fields should be removed	Nicht verwendete „private“ Felder sollten entfernt werden.	veralteter Code

The "var" keyword should not be used	Das Schlüsselwort „var“ sollte nicht verwendet werden.	veralteter Code
Unused local variables should be removed	Nicht verwendete lokale Variablen sollten entfernt werden.	veralteter Code
Empty statements should be removed	Leere Anweisungen sollten entfernt werden.	veralteter Code
Track uses of "TODO" tags	Verfolgen von „TODO“-Tags.	veralteter Code
Deprecated features should not be used	Veraltete Funktionen sollten nicht verwendet werden.	veralteter Code
Alias functions should not be used	Alias-Funktionen sollten nicht verwendet werden.	veralteter Code
"default" clauses should be first or last	„default“-Klauseln sollten entweder zuerst oder zuletzt stehen.	Verständlichkeit
Cognitive Complexity of functions should not be too high	Die kognitive Komplexität von Funktionen sollte nicht zu hoch sein.	Verständlichkeit
Functions should not be nested too deeply	Funktionen sollten nicht zu tief verschachtelt sein.	Verständlichkeit
Regular expressions should not contain empty groups	Reguläre Ausdrücke sollten keine leeren Gruppen enthalten.	Verständlichkeit
Regular expressions should not contain multiple spaces	Reguläre Ausdrücke sollten keine mehrfachen Leerzeichen enthalten.	Verständlichkeit
Regular expressions should not be too complicated	Reguläre Ausdrücke sollten nicht zu kompliziert sein.	Verständlichkeit
Use of namespaces should be preferred to "include" or "require" functions	Die Verwendung von Namespaces sollte gegenüber den Funktionen "include" oder "require" bevorzugt werden.	Verständlichkeit
Ternary operators should not be nested	Verschachtelte Ternäroperatoren sollten vermieden werden.	Verständlichkeit
Classes named like "Exception" should extend "Exception" or a subclass	Klassen mit Namen wie „Exception“ sollten „Exception2 oder eine Unterklasse erweitern.	Verständlichkeit
"switch" statements should not have too many "case" clauses	„switch“-Anweisungen sollten nicht zu viele „case“-Klauseln haben.	Verständlichkeit
"for" loop stop conditions should be invariant	Die Stoppbedingungen der „for“-Schleife sollten invariant sein.	Verständlichkeit
Functions should not contain too many return statements	Funktionen sollten nicht zu viele „return“-Anweisungen enthalten.	Verständlichkeit
Functions should not have too many parameters	Funktionen sollten nicht zu viele Parameter haben.	Verständlichkeit
"<?php" and "<?=" tags should be used	„<?php“ und „<?=“ Tags sollten verwendet werden.	Verständlichkeit
Boolean literals should not be redundant	Boolesche Literalwerte sollten nicht redundant sein.	Verständlichkeit
"switch" statements should not be nested	„switch“-Anweisungen sollten nicht verschachtelt sein.	Verständlichkeit
Cyclomatic Complexity of functions should not be too high	Zyklomatische Komplexität von Funktionen sollte nicht zu hoch sein.	Verständlichkeit

Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	Kontrollflussanweisungen „if“, „for“, „while“, „switch“ und „try“ sollten nicht zu tief verschachtelt sein.	Verständlichkeit
Cyclomatic Complexity of classes should not be too high	Zyklomatische Komplexität von Klassen sollte nicht zu hoch sein.	Verständlichkeit
Expressions should not be too complex	Ausdrücke sollten nicht zu komplex sein.	Verständlichkeit
Increment (++) and decrement (--) operators should not be used in a method call or mixed with other operators in an expression	Inkrement (++) und Dekrement (--) Operatoren sollten nicht in einem Methodenaufruf verwendet oder mit anderen Operatoren in einem Ausdruck vermischt werden.	Verständlichkeit
Configuration should not be changed dynamically	Die Konfiguration sollte nicht dynamisch geändert werden.	Verständlichkeit
Variable variables should not be used	Variable Variablen sollten nicht verwendet werden.	Verwendung von Variablen
References should not be passed to function calls	Referenzen sollten nicht an Funktionsaufrufe übergeben werden.	Verwendung von Variablen
Constants should not be redefined	Konstanten sollten nicht erneut definiert werden.	Verwendung von Variablen
Parameters should be passed in the correct order	Parameter sollten in der richtigen Reihenfolge übergeben werden.	Verwendung von Variablen
Functions and variables should not be defined outside of classes	Funktionen und Variablen sollten nicht außerhalb von Klassen definiert werden.	Verwendung von Variablen
"global" should not be used	„global“ sollte nicht verwendet werden.	Verwendung von Variablen
Duplicate values should not be passed as arguments	Duplizierte Werte sollten nicht als Argumente übergeben werden.	Verwendung von Variablen
WordPress option names should not be misspelled	WordPress-Optionnamen sollten nicht falsch geschrieben sein.	WordPress
WordPress options should not be defined at the end of "wp-config.php"	WordPress-Optionen sollten nicht am Ende von „wp-config.php“ definiert werden.	WordPress

Quelle: (PHP Static Code Analysis, o. D.)