

Essay

version: 2.0
datum: 02.11.2026
lesezeit: 75 min

Specs beschleunigen, was schon da ist ■

Spec-Driven Development und der Reifegrad

Von

Nils Plagemann

Inhalt

00

Abstract

1 min

01

Das Versprechen

3 min

- 1.1 Eine alte Idee wird neu verkauft
- 1.2 Reibung, Drift und Überproduktion
- 1.3 Die eine Frage

02

Drei Dinge heißen Spec-Driven Development

8 min

- 2.1 Was eine Spezifikation ist, wenn mehr als ein Dokument gemeint ist
- 2.2 Der Vertrag: API-First, OpenAPI, Protobuf
- 2.3 Das Verhalten: BDD, Gherkin, Cucumber
- 2.4 Die Entscheidung: ADRs
- 2.5 Der Beweis: TLA+, Alloy, Dafny
- 2.6 Die Sprache: DDD und die Ubiquitous Language
- 2.7 Die Nachbarn: TDD und Design-First

03

Was der Markt verkauft

7 min

- 3.1 Wer was baut
- 3.2 Die Werkzeugkette
- 3.3 Die Spec wird zum Prompt
- 3.4 Die Spec wird zum Prüfstein
- 3.5 Die Spec wird zum Ereignis

04

Die Spec als Vertrag zwischen Teams

3 min

- 4.1 Was die Organisation in die Spec schreibt
- 4.2 Zentrale Regel, lokale Freiheit

05

Der Reifegrad

3 min

- 5.1 Was Reifegrad hier heißt
- 5.2 Automatisiertes Chaos
- 5.3 Die Illusion der Geschwindigkeit
- 5.4 Wer hält die Spec aktuell?

06

Was eine Spec verschwendet

6 min

- 6.1 Die sieben Verschwendungen, an der Spec gemessen
- 6.2 Wie lange dauert eine Spezifikation wirklich?
- 6.3 Verifikation und Validierung
- 6.4 Falsche Anforderung, sauber automatisiert
- 6.5 Klarheit schlägt Tooling
- 6.6 Ein Beispiel ist mehr wert als eine Regel

07

Was andere Branchen schon wissen

8 min

- 7.1 Bauwesen: Leistungsphasen und Nachtrag
- 7.2 Luftfahrt und Medizin: Nachweis statt Meinung
- 7.3 Recht: Bestimmtheit und Konfliktlösung
- 7.4 Produktdesign: Appetite statt Umfang
- 7.5 Fünf Bräuche, ein Muster

08

Ein SDD, das zum Reifegrad passt

12 min

- 8.1 Die Weiche: acht Fragen
- 8.2 Stufe 1 und 2: die Fachvision zuerst
- 8.3 Stufe 3 und darüber: die technische Spec
- 8.4 Wo Agenten schreiben: Evals vor der Spec
- 8.5 Welche Spec für welches Team
- 8.6 Ermöglichen statt erzwingen
- 8.7 Woran der Vorschlag scheitern würde

09

Preis und Ausblick

2 min

- 9.1 Overhead und Überkonstruktion
- 9.2 Wohin die Wertschöpfung wandert

10

Was bleibt

1 min

11

Quellen

21 min

Abbildungen

- Abb. 01 Backlog-Kopplung zwischen Teams
- Abb. 02 Drei Lebensdauern einer Spec
- Abb. 03 Specs nach Lebensdauer und Prüfbarkeit
- Abb. 04 Wo eine Spec im Ablauf hängen bleibt
- Tab. 01 Formen der Spezifikation
- Tab. 02 Spec-Werkzeuge am 14.10.2026
- Abb. 05 Der Vertrag als Published Language
- Tab. 03 Fünf Bräuche, ein Muster
- Abb. 06 Branchen-Bräuche und ihr Gegenstück in Software
- Tab. 04 Welche Spec für welches Team

Abstract

Der Markt verkauft Spec-Driven Development als Weg zu besserer Software. Belegt ist das nicht. Eine Spec koordiniert – prüfen kann sie nur, wo eine Maschine sie liest. Und sie verstärkt, was eine Organisation schon kann. Vor die Wahl des Werkzeugs gehört also die Frage nach dem Reifegrad.

Anm. 1

Böckeler 2025 [1].

Für einen kleinen Bug hat Kiro vier User Stories mit sechzehn Akzeptanzkriterien geschrieben. Birgitta Böckeler von Thoughtworks hat im Herbst 2025 drei Werkzeuge für Spec-Driven Development getestet und ihre Beobachtungen am 15. Oktober auf martinfowler.com veröffentlicht.¹ Ein Satz daraus:



„To be honest, I’d rather review code than all these markdown files.“

Code fällt durch, wenn er nicht läuft. Eine Spec fällt nie durch. Ihr Review kennt keinen Fehlerzustand.

Anm. 2

GitHub 2026 [2].

Anm. 3

Swaminathan, Ilinca 2025 [3];
Sharma 2026 [4].

Ein Jahr später ist aus der Idee ein Markt geworden. Das Repository von GitHub Spec Kit, im August 2025 angelegt, hatte am 14. Oktober 2026 **141.263 Sterne**.² Kiro, die Entwicklungsumgebung von AWS, ist seit dem 17. November 2025 allgemein verfügbar und der Nachfolger von Amazon Q Developer, das seit Mai 2026 keine Neuanmeldungen mehr annimmt.³ Tessl hat laut Presseberichten **125 Millionen Dollar** eingesammelt, um Software nur noch aus Specs zu erzeugen.⁴ Alle verkaufen dasselbe Versprechen: *Wer die Absicht vorher aufschreibt, bekommt bessere Software.*

Anm. 4

CTech 2024 [5].

Dieser Essay prüft das Versprechen. Das Ergebnis steht vorne:

Kernsatz



Eine Spec macht Software nicht von selbst besser. Sie macht schneller und sichtbarer, was eine Organisation ohnehin kann.

Wo die Fachlichkeit geklärt ist und jemand für Verträge zuständig ist, richtet sie Menschen und Agenten auf dasselbe Problem aus und spart Abstimmung. Wo beides fehlt, automatisiert sie das Fehlen. Prüfen kann eine Spec nur dort, wo eine Maschine sie liest. **Vor die Wahl des Werkzeugs gehört also die Frage, wie reif die Organisation ist, die es benutzen soll.**

Das Ergebnis ist begründet, nicht gemessen. Was es widerlegen würde, steht am Ende von Kapitel 8.

kapitel/01



01 / 11 · 3 min

01

Das Versprechen

1.1

Anm. 5
OMG 2003 [6].

Eine alte Idee wird neu verkauft

2003 veröffentlichte die Object Management Group den *MDA Guide 1.0.1*. Model Driven Architecture sollte Systeme plattformunabhängig spezifizieren – die Spezifikation würde dann maschinell in plattform-spezifischen Code übersetzt. Plattformunabhängiges Modell, plattform-spezifisches Modell, Transformation dazwischen. Der Code wäre ein Nebenprodukt gewesen.⁵

Ein Jahr später erklärte Dave Thomas in *IEEE Software*, warum das scheitern würde: latente Komplexität, fehlende interoperable Implementierungen, die zufällige Komplexität echter Plattformen und das Tempo der Branche. Sein Satz zum Kern der Idee blieb:

Anm. 6

Thomas 2004 [7].



„The term executable specification is an oxymoron – if the specification were truly executable, it would actually be ‘the thing.’“⁶

Anm. 7

Fowler 2004 [8].

Anm. 8

OMG 2014 [9].

Martin Fowler stellte sich im Februar 2004 auf die Seite derer, die in MDA die „*Night of the Living Case Tools*“ sahen – eine Wiederkehr der CASE-Werkzeuge aus den Achtzigern.⁷ Beide Kritiken kamen von Praktikern. Die überarbeitete Fassung, der *MDA Guide 2.0*, kam erst 2014.⁸

Böckeler zieht die Parallele zur modellgetriebenen Entwicklung selbst – und benennt den entscheidenden Unterschied: MDA brauchte eine maschinenlesbare Spezifikationssprache und Generatoren, die jemand bauen und pflegen musste. **Beides fällt mit Sprachmodellen weg**, weil das Modell Prosa liest und Code schreibt. Doch sie benennt auch den Verlust:

Anm. 9

Böckeler 2025 [1].



„The parseable structure also had upsides that we’re losing now.“⁹

Eine Sprache mit Grammatik konnte ein Werkzeug auf Vollständigkeit und Widersprüche prüfen. Ein Markdown-Dokument kann das nicht.

Anm. 10

Grove 2025 [10].

Sean Grove von OpenAI vertrat im Juni 2025 auf der *AI Engineer World’s Fair* die Gegenposition: Die Spec sei das Artefakt, das man behält, Code nur die Ausgabe. Wer den Prompt löscht und den Code versioniert, verhalte sich wie jemand, der den Quelltext schreddert und das Binary eincheckt.¹⁰ Das Bild ist treffend – und es setzt etwas voraus, das MDA auch schon voraussetzte: **dass die Spec stimmt.**

Neu an der Welle von 2025 ist also nicht die Idee, sondern der Leser. Früher hat ein Generator die Spec gelesen, heute ein Sprachmodell. Die Frage von 2004 bleibt:



Wer prüft, ob das Aufgeschriebene richtig ist, bevor es Code wird?

1.2

Reibung, Drift und Überproduktion

Die Probleme, die Spec-Driven Development lösen will, sind echt.

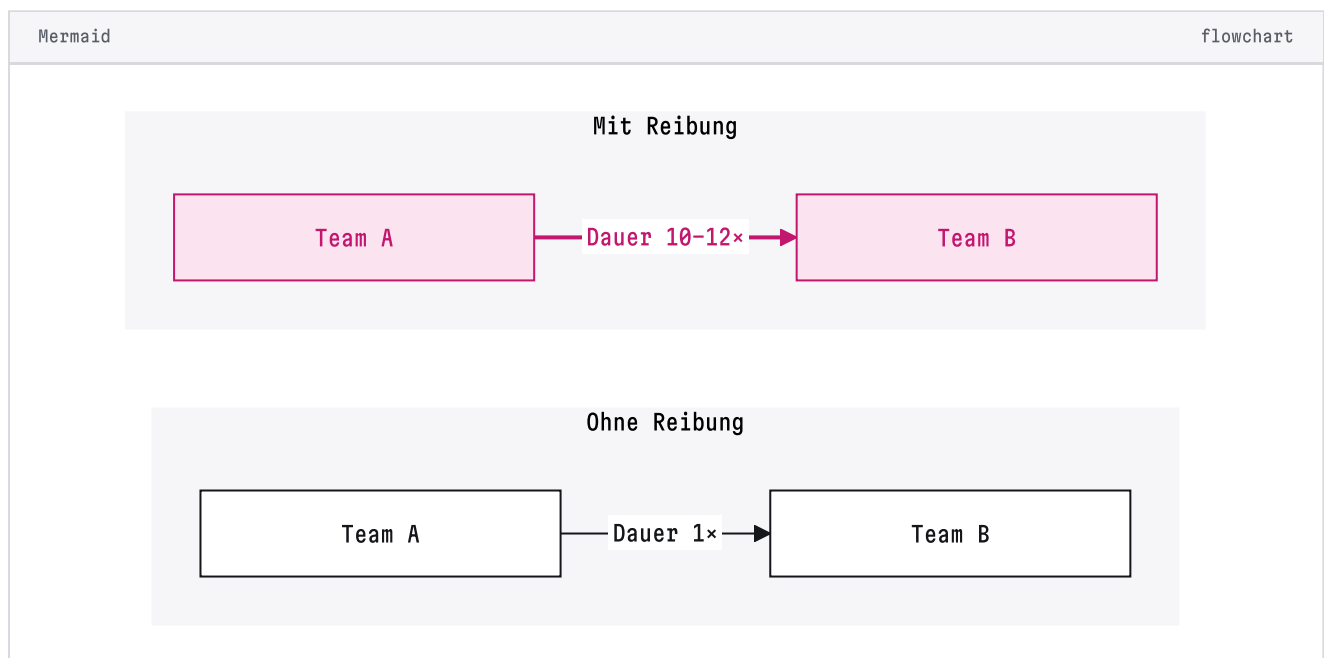
Anm. 11

Bottcher 2018 [11].

Das erste ist **Reibung zwischen Teams**. Evan Bottcher berichtete 2018 von einer Untersuchung bei einem australischen Telekommunikationsunternehmen: Aufgaben, die auf ein anderes Team warten mussten, brauchten zehn- bis zwölfmal so lange wie die übrigen.¹¹

Abb. 01

Backlog-Kopplung zwischen Teams



Anm. 12

Bottcher 2018 [11].

Das ist ein Beratungsfall ohne veröffentlichte Stichprobe – doch jede Organisation mit mehr als drei Teams kennt das Muster. Bottcher nennt es *Backlog-Kopplung*.¹²

Anm. 13

APIContext 2024 [12].

Das zweite ist **Drift** – der Abstand zwischen Dokumentation und Realität. APIContext, Anbieter von API-Monitoring, hat nach eigenen Angaben 650 Millionen API-Aufrufe gegen über 10.000 Endpunkte ausgewertet: **75 % der Produktions-APIs wichen von ihrer OpenAPI-Beschreibung ab**, 43 % hatten keine auffindbare öffentliche.¹³ Die Zahl stammt von einem Anbieter, der Überwachung verkauft. Dennoch zeigt eine unabhängige ältere Messung in dieselbe Richtung:

Anm. 14

Tan, Wagner, Treude 2024 [13].

Tan, Wagner und Treude untersuchten 2022 über 3.000 GitHub-Projekte – die meisten hatten irgendwann mindestens einen toten Codeverweis in der Dokumentation.¹⁴

Anm. 15

Perez 2026 [14].

Das dritte Problem erzeugt Spec-Driven Development selbst. Alfredo Perez zog am 31. August 2026 Bilanz: 94 Spec-Ordner, 238 Markdown-Dateien, 15.225 Zeilen – nach Abschluss der Änderung nie wieder gelesen. Was als Dokumentation gedacht war, wurde zur Ablagerung. Sein Ausweg: eine lebende Spec *je Fähigkeit* statt einer neuen *je Änderung*.¹⁵

Reibung und Drift soll die Spec beheben. Überproduktion ist das, was sie dabei anrichtet, wenn niemand sagt, wie viel Spec ein Vorhaben verträgt.

1.3

Die eine Frage

Um eine Frage kreist dieser Text:

Kernsatz



Unter welchen Bedingungen macht eine Spezifikation Software besser – und wann beschleunigt sie nur, was ohnehin schief läuft?

Drei Linien laufen durch die Antwort:

1. **Koordination vs. Qualität:** Eine Spec richtet Menschen und Agenten auf dasselbe Problem aus – das ist ihr belegter Nutzen. Fehler verhindert sie nur, wenn ein Algorithmus die Implementierung gegen sie prüft.
2. **Versprechen vs. Belege:** Der Markt behauptet Wirkung, gemessen ist bisher fast nur das Gegenteil oder nichts.

3. **Verstärkung vs. Veränderung:** Eine Spec verstärkt, was eine Organisation schon kann. Ob sie hilft, hängt also weniger am Werkzeug als am Reifegrad der Organisation, die es einsetzt.

02

Drei Dinge heißen Spec-Driven Development

2.1

Was eine Spezifikation ist, wenn mehr als ein Dokument gemeint ist

Anm. 16

ISO/IEC/IEEE 29148:2018 [15];
IEEE 830-1998 [16]; ISO/IEC/IEEE
DIS 29148 [17].

Die Norm hat eine klare Vorstellung davon, was eine Spezifikation ist. ISO/IEC/IEEE 29148 hat 2011 IEEE 830 von 1998 abgelöst. Die heutige Ausgabe von 2018 kennt vier Arten von Anforderungsdokumenten, von der Business Requirements Specification bis zur Software Requirements Specification. Eine einzelne Anforderung soll unter anderem notwendig, eindeutig, vollständig, machbar und verifizierbar sein. Eine dritte Ausgabe ist im Entwurf.¹⁶ Gemessen an diesem Maßstab sind die Prosa-Specs der neuen Werkzeuge fast nie Spezifikationen. Sie sind weder systematisch prüfbar noch rückverfolgbar.

Anm. 17

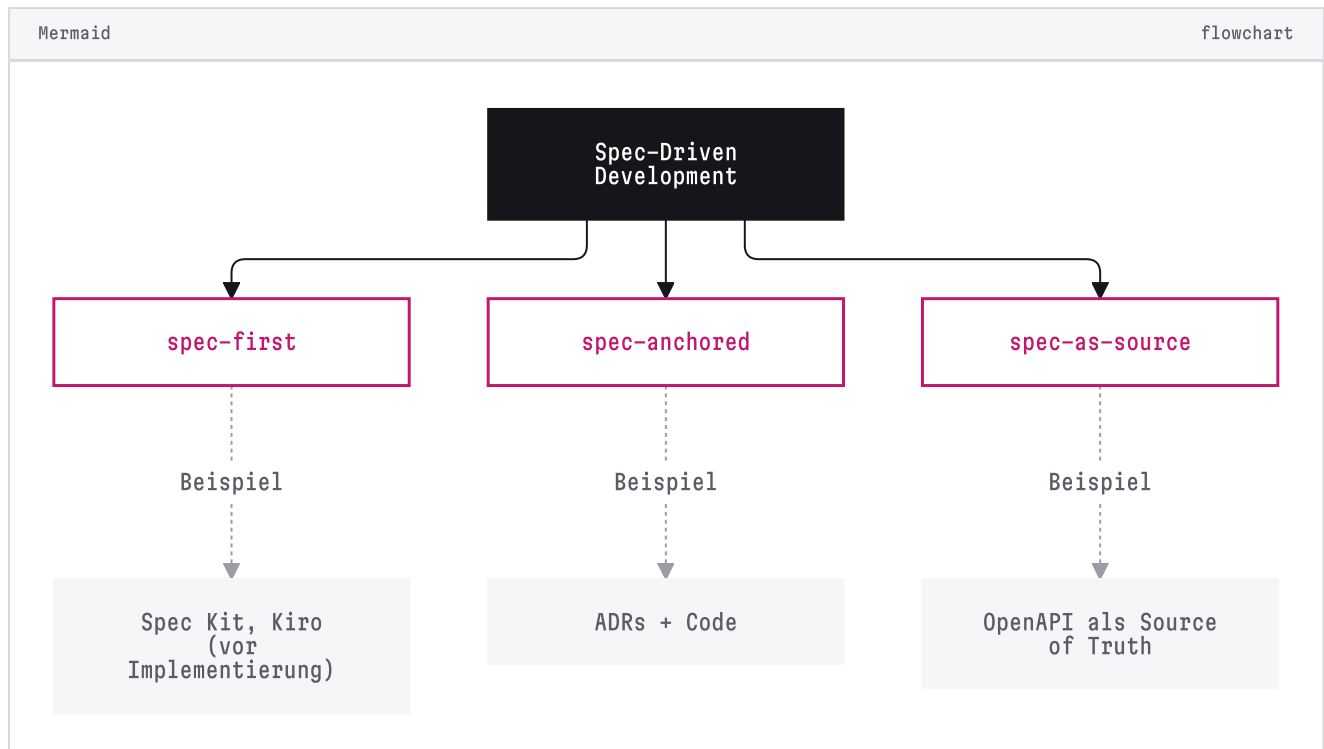
Böckeler 2025 [1].

Den Werkzeugen ist das kaum vorzuwerfen. Der Begriff hat sich gespalten. „Spezifikation“ meint inzwischen mindestens drei verschiedene Dinge, und sie unterscheiden sich darin, wie lange die Spec lebt. Böckeler hat das in ihrem Text vom Oktober 2025 sortiert:¹⁷

- **Spec-first** heißt, die Spec wird vor dem Code geschrieben, einmal benutzt und dann vergessen.
- **Spec-anchored** heißt, die Spec lebt mit dem Code weiter und wird mit ihm geändert.
- **Spec-as-source** heißt, nur noch die Spec wird gepflegt, und der Mensch fasst den Code nicht mehr an.

Abb. 02

Drei Lebensdauern einer Spec



Fast alles am Markt ist spec-first und wirbt mit dem Versprechen von spec-as-source.

Die zweite Achse ist, wer die Spec liest. Am einen Ende prüft eine Maschine: ein Model Checker, ein Compiler, ein Vergleichswerkzeug, ein Test. Am anderen Ende liest ein Mensch oder ein Sprachmodell Prosa und deutet sie. **Nur am ersten Ende verhindert eine Spec Fehler mechanisch.** Am zweiten Ende richtet sie aus, und ob das Ergebnis stimmt, entscheidet jemand anderes.

Mit diesen zwei Achsen – Lebensdauer und Prüfbarkeit – lassen sich die älteren Formen sortieren.

Abb. 03

Specs nach Lebensdauer und Prüfbarkeit

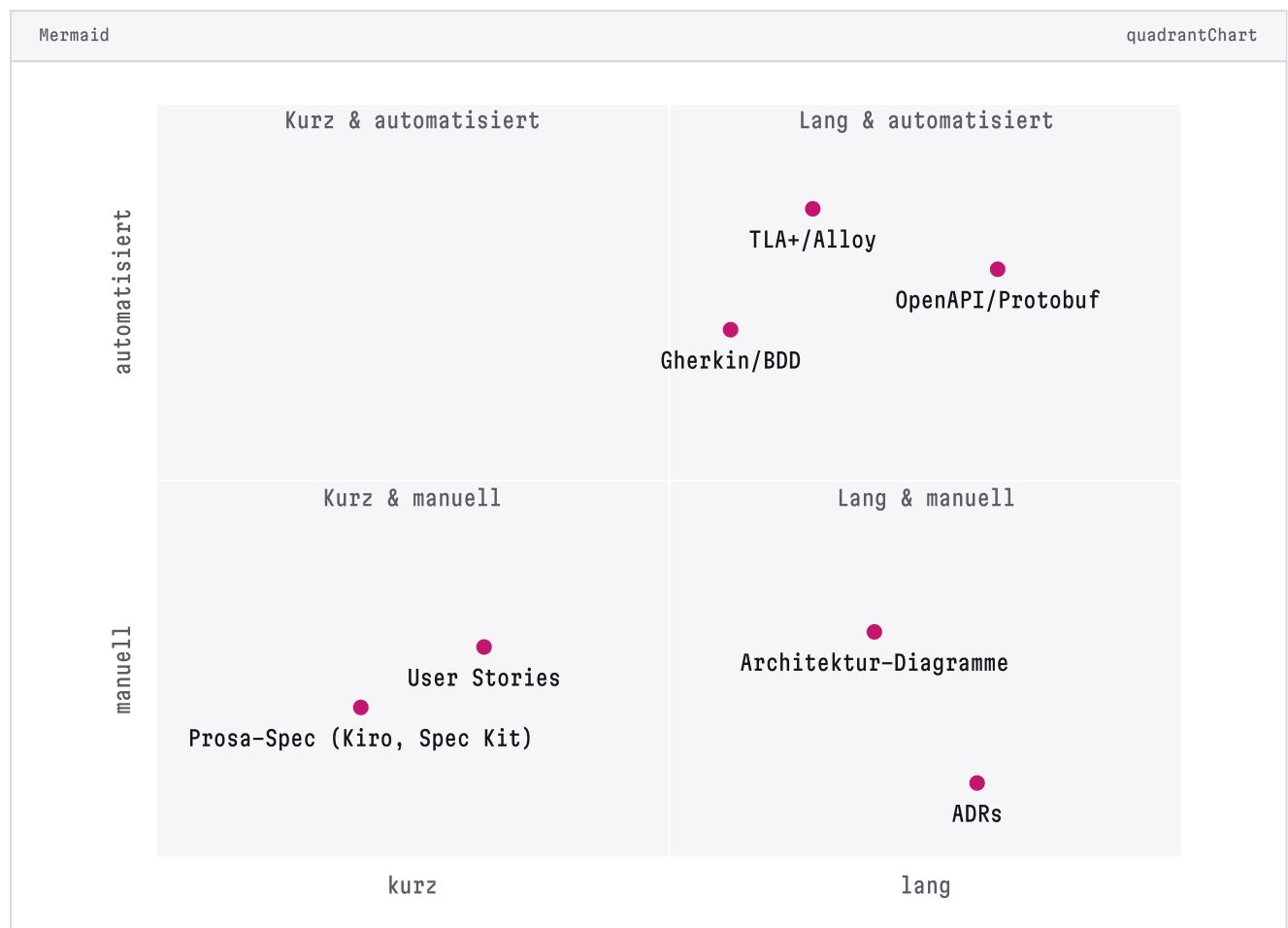
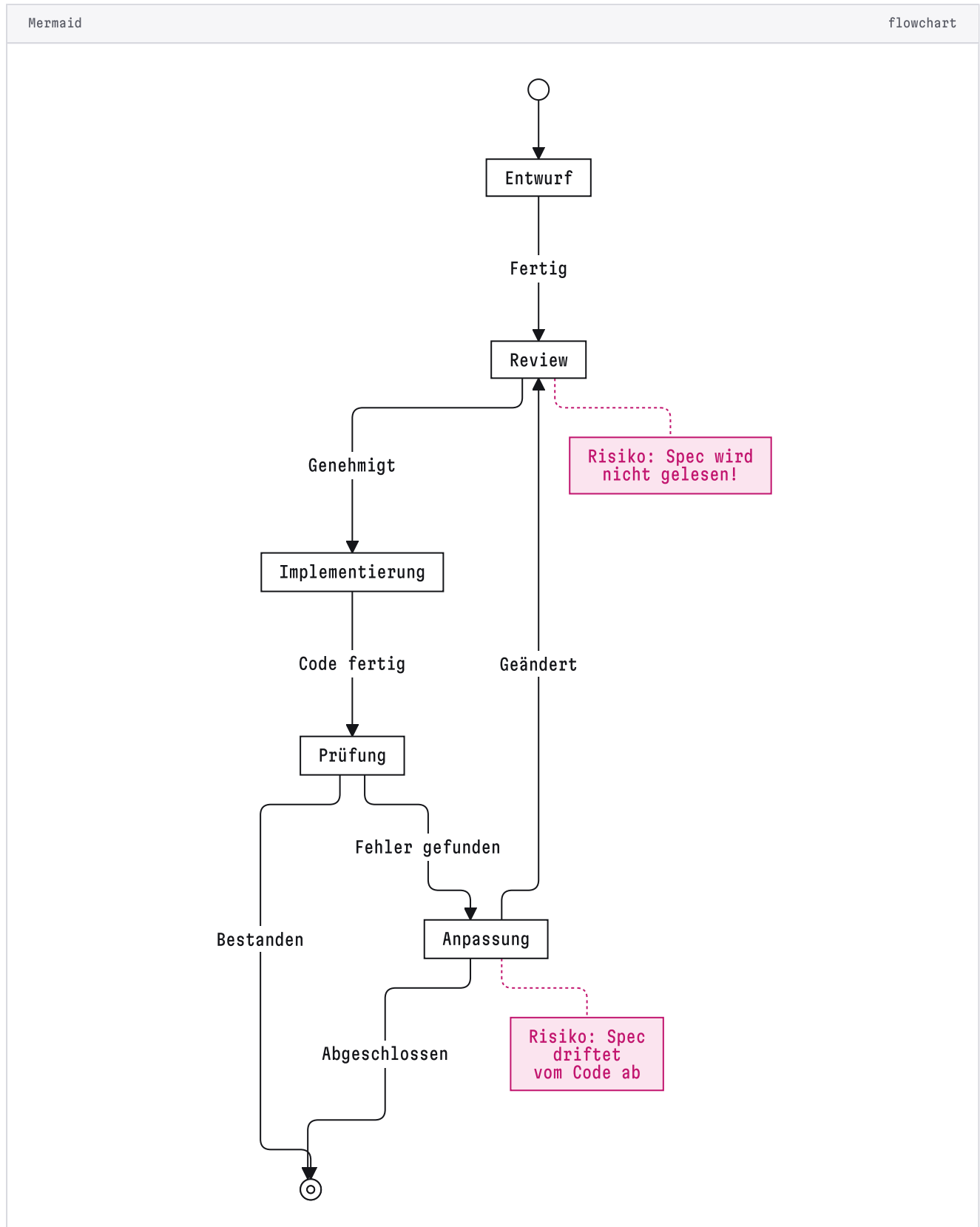


Abb. 04

Wo eine Spec im Ablauf hängen bleibt



2.2

Anm. 18

Amundsen 2015 [18]; OpenAPI Initiative [19].

Anm. 19

OpenAPI Initiative [19]; OAI Best Practices [20]; Arazzo 2024 [21].

Anm. 20

Protocol Buffers [22]; gRPC [23]; Buf [24].

Anm. 21

Lane 2014 [25].

Anm. 22

Zalando [26].

Anm. 23

Postman 2025 [27].

Anm. 24

Robinson 2006 [28].

Anm. 25

Fellows 2025 [29].

Der Vertrag: API-First, OpenAPI, Protobuf

Die älteste Form, die heute noch täglich im Repository liegt, ist der Schnittstellenvertrag. Swagger ist bei Wordnik entstanden, als generiertes JSON-Modell einer API. Version 1.0 erschien am 10. August 2011, 2.0 im September 2014. Im November 2015 wurde die OpenAPI Initiative gegründet, zum Jahresende hat SmartBear ihr die Spezifikation übergeben, seit Januar 2016 heißt sie OpenAPI Specification. Version 3.0 folgte im Juli 2017, 3.1 im Februar 2021, 3.2 am 19. September 2025. Die aktuelle Fassung ist 3.2.1 vom 10. September 2026.¹⁸

Das Selbstverständnis: eine „*interface description for HTTP APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to source code*“. Beschrieben werden Pfade, Operationen, Parameter, Datenformen und Antwortcodes. Die OpenAPI Initiative sagt selbst, das Format sei „*not capable of describing every possible HTTP API*“. Für die Reihenfolge von Aufrufen hat sie mit Arazzo eine zweite Spezifikation geschaffen. Und die Spezifikation 3.0 hält zu Links zwischen Operationen fest: „*The presence of a link does not guarantee the caller’s ability to successfully invoke it*“.¹⁹

Bei Google lief seit 2001 ein ähnliches Format für interne Dienste. Protocol Buffers wurden am 7. Juli 2008 Open Source, gRPC als Aufrufrahmen im Februar 2015, Version 1.0 im August 2016. Die Kernforderung war von Anfang an Evolution: Alte und neue Server sollen die Daten des jeweils anderen lesen können. Die Regeln dafür sind streng. Eine Feldnummer „*cannot be changed once your message type is in use*“, Nummern dürfen nie wiederverwendet werden, gelöschte müssen reserviert werden. Mit **buf breaking** gibt es ein Werkzeug, das jede Änderung gegen einen früheren Stand prüft.²⁰

API-First als Haltung hat Kin Lane 2014 beschrieben: Bevor die Anwendung gebaut wird, entsteht die API.²¹ Zalando hat daraus eine Regel gemacht: „*You must define APIs first, before coding its implementation, using OpenAPI as specification language*“.²² Wie verbreitet das ist, misst vor allem Postman. Im Bericht für 2025 mit über 5.700 Befragten verfolgen 82 % der Organisationen API-First in irgendeiner Form.²³

Auf den beiden Achsen steht der Vertrag weit vorn bei der Prüfbarkeit, aber nur für die Form. **Ob die Antwort richtig ist, kann ein Werkzeug nicht prüfen.** Ian Robinson hat das 2006 so formuliert: „*doc-*

ument schemas are only a portion of what a service provider has to offer consumers“.²⁴ Matt Fellows von PactFlow sagt es kürzer: „Schemas are abstract – contracts are concrete“.²⁵

Anm. 26

Wright [30].

Der grundsätzlichsste Einwand kommt von Hyrum Wright: „With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody.“ Ab einer gewissen Zahl von Konsumenten ist der wirkliche Vertrag das beobachtbare Verhalten. Wright: „the implementation has become the interface“.²⁶

2.3

Anm. 27

North 2006 [31]; Keogh 2014 [32].

Das Verhalten: BDD, Gherkin, Cucumber

Behavior-Driven Development hat Dan North 2006 in „Introducing BDD“ beschrieben. Given-When-Then hat North gemeinsam mit Chris Matts entwickelt. Der Anlass war ein Wort. „Test“ führte Entwickler beim TDD in die Irre, „Verhalten“ nicht. Mit Matts kam die Fachseite dazu: Auch Anforderungen sind Verhalten.²⁷ BDD ist weniger ein zweites Verfahren neben TDD als dessen Sprache für Leute, die keinen Code lesen.

Anm. 28

Adzic 2009 [33]; Adzic 2011 [34];
Adzic 2020 [35].

Gojko Adzic hat das 2009 und 2011 zu *Specification by Example* ausgebaut. Zehn Jahre später zog er Bilanz, in zwei Sätzen: „Given-When-Then won, by a huge margin“ und „Collaboration won over testing“.²⁸ Das Format hat sich durchgesetzt, der Grund ist die Zusammenarbeit, nicht die Automatisierung. Auf der Prüfbarkeitsachse steht ein automatisiertes Beispiel weit vorn. Auf der Lebensdauerachse hängt alles an der Pflege.

2.4

Anm. 29

Nygard 2011 [36].

Die Entscheidung: ADRs

Michael Nygard hat am 15. November 2011 vorgeschlagen, Architekturentscheidungen als kurze Textdateien im Repository festzuhalten: Kontext, Entscheidung, Status, Konsequenzen. Ersetzte Entscheidungen bleiben liegen und werden als „superseded“ markiert. Seine Begründung war agil: „Agile methods are not opposed to documentation, only to valueless documentation.“²⁹

Ein ADR ist eine Spezifikation besonderer Art. Es prüft nichts und beschreibt kein Verhalten. Es begründet. Auf der Prüfbarkeitsachse steht es ganz hinten, auf der Lebensdauerachse hat es eine eingebaute Regel: Eine Entscheidung wird nicht geändert, sondern durch eine neue ersetzt.

2.5

Anm. 30

Lamport 1994 [37]; Lamport 2002 [38]; Jackson 2002 [39]; Leino 2010 [40]; Harel 1987 [41].

Der Beweis: TLA+, Alloy, Dafny

Am äußersten Ende der Prüfbarkeitsachse stehen formale Spezifikationen. Leslie Lamport hat TLA um 1990 erfunden. TLA+ ist die Sprache, TLC der Model Checker. Alloy von Daniel Jackson sucht Gegenbeispiele. Dafny von Rustan Leino prüft Vor- und Nachbedingungen im Programm selbst. David Harels Statecharts von 1987 gehören in dieselbe Familie.³⁰

Anm. 31

Newcombe et al. 2015 [42].

Anm. 32

Bornholt et al. 2021 [43].

Der bekannteste Beleg kommt von Amazon. Chris Newcombe und Kollegen berichten 2015 in den *Communications of the ACM*, dass AWS TLA+ seit 2011 nutzt. Ein Modell von Teilen von DynamoDB fand einen Designfehler, dessen kürzester Fehlerpfad 35 Schritte hatte.³¹ Für ShardStore in S3 waren die ausführbaren Referenzmodelle 2021 rund ein Prozent so groß wie die Implementierung. Die Prüfungen gegen diese Modelle haben 16 Probleme vor der Produktion abgefangen.³²

Anm. 33

Newcombe et al. 2015 [42].

Derselbe Text von 2015 zieht die Grenze selbst. Auf die Frage, woher man wisse, dass der Code das verifizierte Design umsetzt, antworten die Autoren: „*The answer is we do not know.*“³³ **Auch die stärkste Spezifikation deckt den Entwurf, nicht die Umsetzung.**

2.6

Anm. 34

Evans 2003 [44]; Evans 2015 [45].

Die Sprache: DDD und die Ubiquitous Language

Eric Evans hat 2003 „*Domain-Driven Design*“ veröffentlicht. Die Ubiquitous Language ist darin keine Technik unter vielen, sondern die Voraussetzung. Evans definiert sie als „*a language structured around the domain model and used by all team members within a bounded context*“. Und: „*Translation blunts communication and makes knowledge crunching anemic.*“³⁴

Anm. 35

Evans 2003 [44]; Evans 2015 [45].

Anm. 36

Fowler 2014 [46].

Ein Bounded Context ist bei Evans „*a description of a boundary [...] within which a particular model is defined and applicable*“.³⁵ Er verlangt, diese Grenze ausdrücklich zu ziehen. Martin Fowler hat 2014 das Beispiel eines Energieversorgers beschrieben, bei dem „*meter*“ in jedem Bereich etwas anderes bedeutete.³⁶

Anm. 37

Evans 2003 [44]; Evans 2015 [45].

Für Specs ist das die erste Ebene. Eine Ubiquitous Language ist nicht maschinell prüfbar, sie lebt so lange wie das Team, das sie spricht. Ein OpenAPI-Dokument mit einem Feld `customer` ist eindeutig für den Parser und mehrdeutig für drei Teams, die unter Kunde drei vers-

chiedene Dinge verstehen. Evans hat für die Stelle, an der zwei Kontexte miteinander reden, die *Published Language* beschrieben: eine gut dokumentierte gemeinsame Sprache als „*common medium of communication*“.³⁷ Ein API-Vertrag zwischen Teams ist dann gut, wenn er eine *Published Language* ist und nicht das interne Modell eines der beiden.

2.7

Die Nachbarn: TDD und Design-First

Anm. 38

Beck 2002 [47].

Anm. 39

Rafique, Mišić 2013 [48].

Test-Driven Development hat Kent Beck 2002 beschrieben.³⁸ Ein Test ist eine Spezifikation im Zuschnitt einer Prüfung. Auf der Prüfbarkeitsachse steht TDD ganz vorn, auf der Lebensdauerachse verschwindet die Absicht im Testcode. Eine Meta-Analyse von Yahya Rafique und Vojislav Mišić über 27 Studien fand 2013 „*a small positive effect on quality but little to no discernible effect on productivity*“.³⁹

„*Design-First*“ ist ein Homonym. Für Schnittstellen heißt es, die Beschreibung vor dem Code zu schreiben. In der Debatte über Softwareentwurf heißt Design-First geplanter Entwurf vor dem Bauen. Wer beides gleichsetzt, kauft mit einem Contract-First-Werkzeug einen Wasserfall, den er nicht bestellt hat.

Tab. 01

Formen der Spezifikation

Form	Beispiel	Wer liest	Prüft Fehler mechanisch	Lebensdauer meist
Sprache	Ubiquitous Language, Glossar	Menschen	nein	so lange wie das Team
Entscheidung	ADR, RFC	Menschen	nein	dauerhaft, ersetzt statt geändert
Prosa-Spec für Agenten	Spec Kit, Kiro, OpenSpec	Sprachmodell	nein	spec-first
Verhalten	Gherkin, ausführbare Beispiele	Menschen und Testlauf	ja, wenn automatisiert	spec-anchored
Vertrag	OpenAPI, Protobuf, AsyncAPI	Werkzeuge und Menschen	die Form, nicht das Verhalten	spec-anchored
Test	TDD	Testlauf	ja	an den Code gebunden
Beweis	TLA+, Alloy, Dafny	Model Checker, Verifizierer	ja, für den Entwurf	spec-anchored

Tab. 02 Spec-Werkzeuge am 14.10.2026

Werkzeug	Träger	Ablauf	Stand 14.10.2026	Lebensdauer
GitHub Spec Kit	GitHub, MIT	constitution, specify, plan, tasks, implement, converge	v1.2.0, 141.263 Sterne	spec-first, spec-anchored nur per Konvention
Kiro	AWS, proprietär	IDE und CLI, requirements, design, tasks, EARS-Notation	GA seit 17.11.2025, Workflows seit 30.09.2026	spec-first
OpenSpec	Fission AI, MIT	explore, propose, apply, archive, verify im erweiterten Profil	v1.14.1, 71.932 Sterne	spec-first
BMAD-METHOD	BMad Code LLC, MIT	rollenbasierte Agenten	v6.12.2, 54.135 Sterne	spec-first
Tessl	Tessl, kommerziell	war spec-as-source	seit 29.01.2026 „Skills on Tessl“	Fokus jetzt auf Skills
Cosmos	Augment Code, seit 08.10.2026 Harness	gemeinsamer Kontext für Teams	Public Preview seit 04.05.2026	offen
Plan-Modi	Claude Code, Cursor, GitHub Copilot, JetBrains Junie, Cline	Plan vor dem Code	nicht als SDD vermarktet	spec-first
AGENTS.md, Cursor Rules	Konvention	Regeln und Kontext im Repo	60.000+ Repos	dauerhaft

Drei Beobachtungen:

Anm. 41
Podjarny 2026 [52].

Anm. 42
Breslav 2026 [55].

Erstens: Fast alles ist spec-first und wirbt mit spec-as-source. Tessl war der einzige ernsthafte Anwarter auf die letzte Stufe und stellt seit Januar 2026 ein anderes Produkt in den Vordergrund: einen Paketmanager für Agent Skills.⁴¹ Andrey Breslav begründete seinen Bruch mit dem eigenen Format bei CodeSpeak: „people don’t want to write specs“.⁴²

Anm. 43
GitHub 2026 [2].

Zweitens: Spec Kit dokumentiert selbst, dass der Schritt `converge` „append-only“ arbeitet. Welches Modell gilt, sei „a team convention, not a CLI setting“.⁴³ Ob eine Spec veraltet oder mit dem Code lebt, entscheidet die Organisation.

Anm. 44

Thoughtworks 2025 [56].

Anm. 45

Thoughtworks 2026 [57].

Anm. 46

Swaminathan, Ilinca 2025 [3].

Anm. 47

Kiro 2026 [49].

Drittens: Unter den großen Anbietern verkaufen nur GitHub und AWS Spec-Driven Development. Thoughtworks hat die Technik im November 2025 in den Ring „Assess“ seines Technology Radar gesetzt und gewarnt, man lerne womöglich gerade neu, dass handgeschriebene Detailregeln für KI nicht skalieren.⁴⁴ In der Ausgabe vom April 2026 steht die Technik nicht mehr als eigener Eintrag, dafür Spec Kit und OpenSpec als Werkzeuge, beide in „Assess“.⁴⁵

Wovon die Anbieter leben, ist Nutzung: Credits, Tokens, Plätze. Kiro hat mit der GA property-basiertes Testen zur Prüfung der Spec eingeführt.⁴⁶ In der Ankündigung der Workflows kommt das Wort Spec nicht vor.⁴⁷ **Der Markt verkauft Specs und baut daneben die Werkzeuge, die tatsächlich prüfen.**

3.2

Die Werkzeugkette

Neben den neuen Agenten-Werkzeugen gibt es eine ältere Kette, die aus einem Vertrag Code, Mocks, Tests und Dokumentation macht. An ihr lässt sich zeigen, was eine Spec prüfen kann.

Anm. 48

OpenAPI Tools [58].

Der OpenAPI Generator erzeugt aus einem Vertrag Client-SDKs, Server-Stubs und Referenzdokumentation. Was er erzeugt, sind Typen, Signaturen und Aufrufcode, keine Logik. Die Qualität hängt vollständig an der Spec.⁴⁸

Anm. 49

Stoplight [59]; WireMock [60].

Mock-Server wie Prism oder WireMock Cloud erzeugen aus der Spec Beispiellantworten.⁴⁹ Das ist reine Koordination. **Logikfehler findet ein Mock nicht.**

Anm. 50

Pact Foundation [61].

Contract Testing prüft eine Ebene tiefer. Bei Pact schreibt der Konsument einen Test gegen einen Mock, das Framework hält die Interaktionen fest, und der Anbieter spielt sie gegen den laufenden Dienst ab. Geprüft werden die minimalen Erwartungen des Konsumenten, nicht die vollständige Antwort.⁵⁰ PactFlow hat die Grenze des eigenen Modells veröffentlicht: Der Nutzen stellt sich nicht sofort ein, und der konsumentengetriebene Ansatz skaliert nicht auf viele Konsumenten.⁵¹ **Der Ausweg ist ein statischer Vergleich:** Passt der Vertrag des Konsumenten in die Beschreibung des Anbieters?

Anm. 52

Hatfield-Dodds, Dygalo 2022 [63].

Das belastbarste Glied der Kette ist Schemathesis. Es erzeugt Testfälle aus dem Schema, leitet Zustandsfolgen ab und minimiert fehlschlagende Fälle. Ein begutachteter Posterbeitrag auf der ICSE 2022 zeigte an 16 echten Webdiensten, dass es 1,4- bis 4,5-mal mehr eindeutige Defekte fand als das jeweils zweitbeste Werkzeug.⁵²

Anm. 53

oasdiff [64].

Lebende Dokumentation prüft gar nichts. Sie wird bei jeder Änderung neu erzeugt. Ist die Spec selbst gedriftet, veröffentlicht das Portal die Drift treu mit. `oasdiff` sagt den wichtigen Satz dazu: Es urteilt gegen den erklärten Vertrag, nicht dagegen, was ein Server tatsächlich akzeptiert.⁵³

Der einzige gemessene Nutzen entsteht dort, wo ein Algorithmus die Implementierung gegen eine maschinenlesbare Zusicherung prüft. Ein Markdown-Dokument, das ein Modell liest, ist ein anderer Gegenstand.

3.3

Die Spec wird zum Prompt

Die neue Welle verspricht mehr als die alte Kette. Die Spec soll den Agenten so führen, dass er das Richtige baut. Ob das stimmt, lässt sich an Messungen prüfen.

Anm. 54

Hill 2026 [65].

Die größte Messung stammt von Hill (ESEM 2026): 88.052 Pull Requests aus 119 Open-Source-Repositories. Es ist eine Beobachtungsstudie, und gemessen wurden Spec-Artefakte, fast immer Verweise auf Tickets und Issues. Die Ausgabe von SDD-Werkzeugen hat keines der Repositories eingeecheckt. Keine der fünf Hypothesen wurde gestützt. Pull Requests mit Spec führten nicht seltener zu Fehlern, und Nacharbeit fiel bei ihnen etwas häufiger an, je nach Modell um 0,5 bis 1,2 Prozentpunkte. Hill erklärt das mit der Schwierigkeit der Aufgaben: Wer eine schwierige Änderung vorhat, schreibt eher eine Spec. Wo keine KI-Nutzung erkennbar war, ging eine Spec mit weniger Nacharbeit einher. Hill wertet das selbst nur als Hinweis, weil es die Korrektur für mehrfaches Testen nicht übersteht.⁵⁴

Anm. 55

Gloaguen et al. 2026 [66].

Anm. 56

Dente, Satriani, Papotti 2026 [67].

Anm. 57

Orlanski et al. 2026 [68].

Die übrigen Messungen stammen aus Konferenz- und Workshopbeiträgen von 2026. Gloaguen et al. (ETH Zürich) fanden, dass Kontextdateien wie AGENTS.md die Erfolgsquote von Agenten nicht generell verbessern und die Ausführung um über 20 % verteuern.⁵⁵ Dente et al. (EURECOM) zeigten, dass festgeschriebene Architektur, Datenbank und ORM die Quote bestandener Prüfungen im Mittel um 27,28 Punkte senken.⁵⁶ SlopCodeBench: Kein Agent löste ein Problem durchgehend, der beste Wert lag bei 14,8 %.⁵⁷

Anm. 58

Tufano et al. 2026 [69].

Der einzige positive Befund kommt aus einem Industrielabor. Tufano et al. (Google) ließen einen Agenten Vorbedingungen, Nachbedingungen und undefiniertes Verhalten aufschreiben. Die Fehlererk-

ennung stieg um 9,8 Prozentpunkte.⁵⁸ Das Papier misst einen Vertrag mit Prüfwert, nicht das Markdown der Spec-Werkzeuge.

Anm. 59

Gonzalez 2026 [70].

Gabriella Gonzalez scheiterte im März 2026 daran, aus der `SPEC.md` von OpenAIs Symphony eine Implementierung zu bauen. **Eine Spec, die genau genug ist, um verlässlich Code zu erzeugen, sei selbst Code.**⁵⁹ Und die Anbieter veröffentlichen keine Zahlen mit Methode. Die Zahl einer drei- bis zehnfach besseren Erfolgsquote im ersten Durchlauf hat keine Quelle mit Rohdaten.

Was bleibt, ist bescheiden und nützlich: Eine Prosa-Spec richtet einen Agenten aus. Sie sagt ihm, was gewollt ist, und er befolgt explizite Anweisungen. Dass das Ergebnis stimmt, sagt sie nicht.

3.4

Die Spec wird zum Prüfstein

Ein zweites Versprechen: Aus der Spec ließen sich automatisch Sicherheitsprüfungen ableiten. Dafür gibt es einen guten Beleg und eine harte Grenze.

Anm. 60

Atlidakis et al. 2019 [71].

Vaggelis Atlidakis, Patrice Godefroid und Marina Polishchuk (Columbia University, Microsoft Research) haben 2019 RESTler vorgestellt, einen Fuzzer, der aus der OpenAPI-Beschreibung Abhängigkeiten zwischen Aufrufen ableitet. Er fand 28 neue Fehler in GitLab und mehrere in Diensten von Azure und Office 365. Ein Fehler ist dort definiert als HTTP-Statuscode 500.⁶⁰ **Die Suche findet Abstürze, keine Rechteverstöße.**

Anm. 61

OWASP 2023 [72].

Anm. 62

Martin-Lopez et al. 2019 [73].

Genau die Rechteverstöße führen die OWASP API Security Top 10 von 2023 an. Auf Platz eins steht *Broken Object Level Authorization*.⁶¹ Ob ein Zugriff erlaubt ist, entscheidet die Fachlogik. Im Vertrag steht davon nichts. Martin-Lopez et al. fanden 2019 in 40 realen APIs, dass 85 % Abhängigkeiten zwischen Eingabeparametern haben. OpenAPI bietet dafür keine Unterstützung.⁶²

Anm. 63

Oluwabunmi, Kim 2026 [74].

Die Prüfung erbt die Fehler der Spec. Für automatisch erzeugte OpenAPI-Beschreibungen von 20 WordPress-Plugins fanden Oluwabunmi und Kim 2026 nur 25 % korrekte Angaben. OWASP ZAP, von diesen Beschreibungen geführt, übersah Schwachstellen. Nach der Korrektur der Spec stieg die Trefferquote um 44 %.⁶³

3.5

Anm. 64

AsyncAPI [75].

Anm. 65

Confluent [76].

Die Spec wird zum Ereignis

Die dritte Tendenz verschiebt die Spec von der Schnittstelle in die Zeit. AsyncAPI beschreibt den Vertrag für Ereignisse. Version 2.0 erschien am 11. September 2019, Version 3.0 am 5. Dezember 2023.⁶⁴

Interessanter als das Format ist, was bei Ereignissen „kompatibel“ heißt. Die Schema Registry von Confluent kennt sieben Kompatibilitätsmodi. Der Standard ist `BACKWARD`, nicht transitiv.⁶⁵ Abwärtskompatibilität bei Ereignissen ist keine Eigenschaft des Schemas allein. Sie ist eine Aussage darüber, welche alten Nachrichten noch jemand lesen muss.

Ein Vertrag über Ereignisse ist immer auch ein Vertrag über die Reihenfolge, in der Teams Software ausliefern.

Mit diesem Punkt verlässt die Frage endgültig das Werkzeug. Der Markt verkauft Formate, Generatoren und Agenten. Ob ein Vertrag hält, entscheidet sich zwischen den Teams, die ihn benutzen.

04

Die Spec als Vertrag zwischen Teams

4.1

Anm. 66

Conway 1968 [77].

Anm. 67

Brooks 1975 [78]; Conway 1968 [77].

Anm. 68

Conway 1968 [77].

Anm. 69

Conway 1968 [77].

Was die Organisation in die Spec schreibt

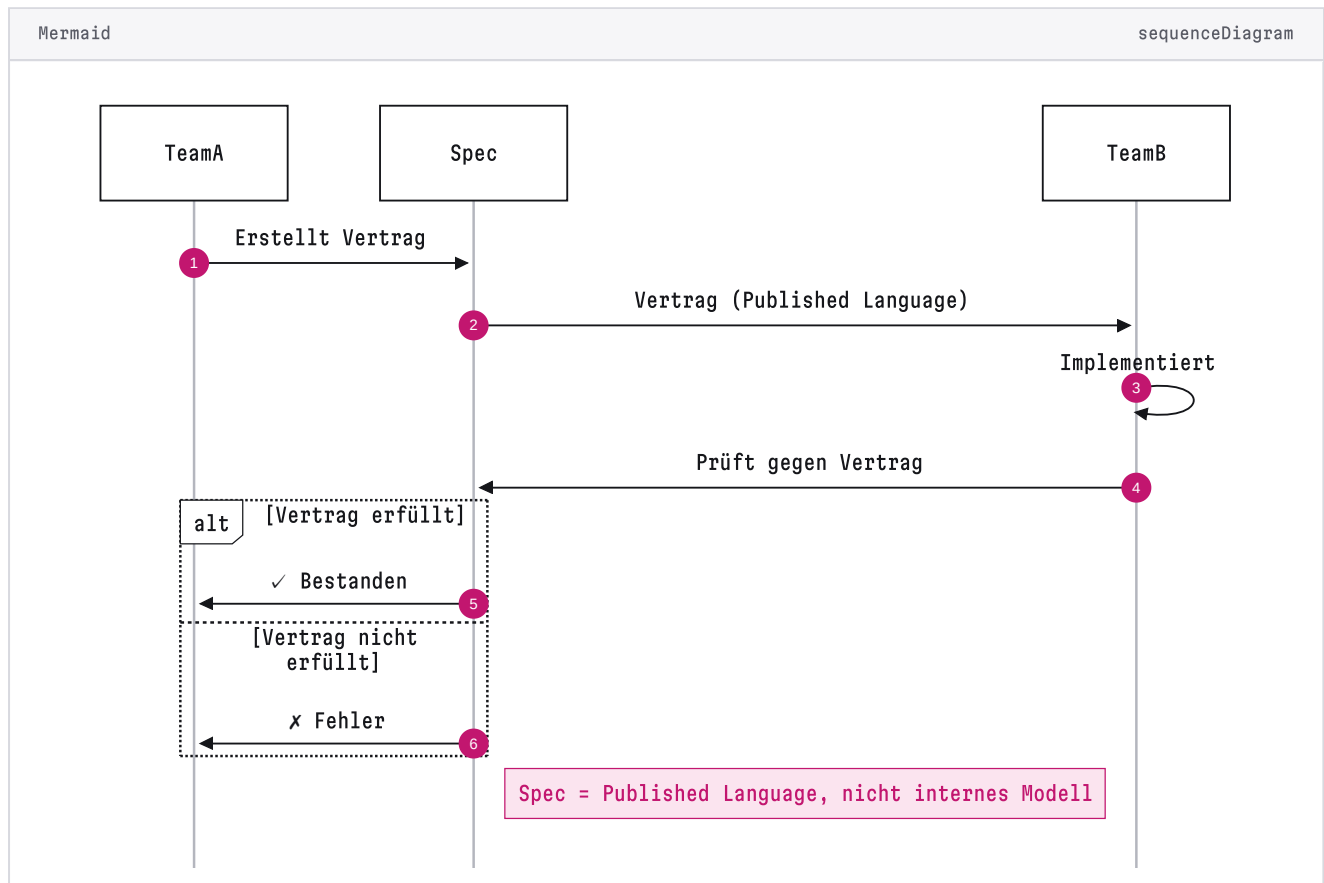
Melvin Conway hat sein bekanntestes Papier 1967 bei der Harvard Business Review eingereicht. Sie lehnte ab, weil er seine These nicht bewiesen habe. Erschienen ist „*How Do Committees Invent?*“ im April 1968 in *Datamation*.⁶⁶ Fred Brooks gab ihm 1975 den Namen „Conway’s Law“. ⁶⁷ Der berühmte Satz: Organisationen, die Systeme entwerfen, „are constrained to produce designs which are copies of the communication structures of these organizations“. ⁶⁸

Conway spricht von einem Homomorphismus, nicht von Ursache und Wirkung. Und zwei Sätze aus demselben Text sind für Specs wichtiger. „*There is no such thing as a design group which is both organized and unbiased.*“ Und: „*flexibility of organization is important to effective design.*“ ⁶⁹

Für eine Spec zwischen Teams folgt daraus: Ein API-Vertrag zwischen zwei Teams beschreibt nicht nur eine Schnittstelle. Er beschreibt, wie gut die beiden Teams miteinander reden. Wo sie sich regelmäßig abstimmen, wird der Vertrag schmal und stabil. Wo sie es nicht tun, wird er breit, vorsichtig und voller optionaler Felder.

Abb. 05

Der Vertrag als Published Language



Anm. 70

Colfer, Baldwin 2016 [79].

Lyra Colfer und Carliss Baldwin haben 2016 für die Mirroring-Hypothese 142 empirische Studien ausgewertet. Von den beschreibenden Studien zu Firmen zeigen 70 % starke Belege. Bei offenen, meist softwarebezogenen Projekten stützt die Mehrheit die Hypothese nicht: 56 % finden keine Spiegelung.⁷⁰ **Digitale Werkzeuge ermöglichen Koordination ohne gemeinsamen Ort.**

Anm. 71

Nagappan, Murphy, Basili 2008 [80].

Nachiappan Nagappan et al. haben 2008 für Windows Vista gezeigt, dass Metriken der Organisationsstruktur die Fehleranfälligkeit von Binärdateien besser vorhersagten als Code Churn oder Komplexität.⁷¹ **Die Organisation schlägt in die Software durch, ob eine Spec dazwischen steht oder nicht.**

Anm. 72

Skelton, Pais 2019 [81].

Matthew Skelton und Manuel Pais haben 2019 mit *Team Topologies* das Werkzeug geliefert: vier Teamtypen, drei Interaktionsmodi, kognitive Last als Maß.⁷² **Ein Interaktionsmodus wie X-as-a-Service setzt einen Vertrag voraus, den der Konsument ohne Rückfrage benutzen kann.**

4.2

Zentrale Regel, lokale Freiheit

Wer Specs zwischen Teams einführt, landet bei der Frage, wer die Regeln setzt. Die Forschung ist sich in einem Punkt einig: **Es gibt keine allgemeine Antwort.**

Anm. 73

Moe et al. 2021 [82].

Nils Brede Moe et al. haben 2021 zwei große Projekte bei Ericsson untersucht. In einem der beiden war die Regel einfach: Ein Team durfte seine Arbeitsweise selbst festlegen, solange keine anderen Teams betroffen waren. Was die Gemeinschaft beschlossen hatte, konnte ein einzelnes Team nicht ändern.⁷³

Anm. 74

Iyer, Lyytinen, Robinson 2026 [83].

Deepa Iyer et al. haben 2026 für 255 Open-Source-Projekte gemessen, wann zentrale Entscheidungen über Anforderungen helfen. Zentralisierung erhöhte die Erledigungsquote. Bei mittlerem bis hohem Volumen und Tempo half dagegen Dezentralisierung. Vereinfacht: zentralisieren, wo wenig und langsam entschieden wird, dezentralisieren, wo viel und schnell entschieden werden muss.⁷⁴

Anm. 75

Datta 2019 [84].

Subhajit Datta hat 2019 an über hundert Projekten getrennt: Mehr zentralisierte Information ging mit mehr Fehlern einher, mehr zentralisierter Einfluss mit weniger.⁷⁵ **Wissen soll breit fließen, Entscheidungsrechte dürfen gebündelt sein.**

Anm. 76

DORA 2024 [85].

Der DORA-Bericht 2024 fand bei Nutzern einer internen Entwicklerplattform höhere Produktivität, aber 8 % weniger Durchsatz und 14 % weniger Stabilität bei Änderungen.⁷⁶ **Eine zentrale Vorschrift hilft dort, wo Unsicherheit gering und Abstimmung über Teams nötig ist. Sonst kostet sie.**

Für Specs heißt das: **Der Vertrag an einer Teamgrenze braucht eine gemeinsame Regel. Wie ein Team innen spezifiziert, braucht keine.**

05

Der Reifegrad

5.1

Was Reifegrad hier heißt

Wenn Specs verstärken, was eine Organisation kann, braucht der Essay ein Wort dafür, was sie kann. Hier wird CMMI verwendet, mit dem Bewusstsein, dass das die angreifbarste Entscheidung ist.

CMMI 3.0 liegt seit dem 6. April 2023 vor.⁷⁷ Die Stufen sind seit 1991 im Kern dieselben: Stufe 1 ad hoc, Stufe 2 geplant und gesteuert, Stufe 3 definierte Prozesse, Stufe 4 und 5 messen und verbessern diese Prozesse quantitativ.⁷⁸

Gegen Reifegradmodelle gibt es lange Kritik. Jens Pöppelbuß und Maximilian Röglinger halten 2011 fest, das Konzept sei „*frequently subject to criticism*“.⁷⁹ **Das Modell wird hier nicht als Leiter verwendet, sondern als Aussage über eine Fähigkeit.** Eine Organisation kann für ihre Schnittstellen auf Stufe 3 arbeiten und für ihre Fachlichkeit auf Stufe 1.

Marilyn Strathern gab Goodharts Beobachtung 1997 ihre bekannte Form: Wenn ein Maß zum Ziel wird, hört es auf, ein gutes Maß zu sein.⁸⁰

5.2

Automatisiertes Chaos

Auf Stufe 1 hängt Erfolg an Personen. Es gibt keinen Ort, an dem Anforderungen geklärt werden, keine Rolle für Verträge, keinen Ablauf für Änderungen.

Führt so eine Organisation ein Spec-Werkzeug ein, erzeugt das Werkzeug Dokumente in einer Menge, die vorher niemand von Hand geschrieben hätte. 238 Dateien, die niemand wieder liest.⁸¹ Ein Review dieser Dokumente hat keinen Fehlerzustand. **Die Dokumente sehen aus wie das Ergebnis eines definierten Prozesses. Den Prozess gibt es nicht.**

Anm. 77
CMMI Institute 2023 [86].

Anm. 78
Paulk et al. 1991 [87].

Anm. 79
Pöppelbuß, Röglinger 2011 [88].

Anm. 80
Strathern 1997 [89]; Goodhart
1976 [90].

Anm. 81

Perez 2026 [14].

Das ist das automatisierte Chaos. **Die Spec ordnet nichts, was vorher ungeordnet war. Sie gibt dem Ungeordneten eine Form, die nach Ordnung aussieht.**

Anm. 82

Kamsties et al. 2001 [91].

Erik Kamsties et al. (Fraunhofer IESE, TU München) fanden in einem Seminarexperiment, dass Studierendenteams 57 % der bekannten Mehrdeutigkeiten selbst auflösten, ohne nachzufragen.⁸² **Auf Stufe 1 ist das der Normalfall.** Ein Agent, der eine Prosa-Spec liest, tut das-selbe, nur schneller.

5.3

Die Illusion der Geschwindigkeit

Spec-Werkzeuge werden mit Geschwindigkeit verkauft, und Agenten liefern sie auch: Code entsteht schneller, als ein Team ihn lesen kann. Ob das Team damit schneller ist, ist eine andere Frage.

Anm. 83

Becker et al. 2025 [92].

Anm. 84

DORA 2025 [93].

METR hat im Juli 2025 eine randomisierte Studie mit 16 erfahrenen Open-Source-Entwicklern veröffentlicht. Mit KI-Werkzeugen brauchen sie 19 % länger. Erwartet hatten sie 24 % Beschleunigung, und noch hinterher glaubten sie an 20 %.⁸³ **Der Abstand zwischen gefühlter und gemessener Geschwindigkeit ist der Befund, der hängen bleibt.** Der DORA-Bericht 2025 sieht eine verwandte Spannung: KI geht mit mehr Durchsatz einher, aber mit weniger Stabilität.⁸⁴

Anm. 85

Ohno 1988 [94]; LEI [95].

Anm. 86

Sedano, Ralph, Péraire 2017 [96].

Im Toyota-Produktionssystem gilt Überproduktion als die schlimmste der sieben Verschwendungen.⁸⁵ **Wer schneller das Falsche baut, hat mehr Falsches.** Todd Sedano et al. haben 2017 bei Pivotal neun Arten von Verschwendung beschrieben. Die erste auf ihrer Liste ist „*building the wrong feature or product*“.⁸⁶ **Kein Spec-Werkzeug fragt, ob das Feature das richtige ist.**

Anm. 87

Boehm, Basili 2001 [97].

Barry Boehm und Victor Basili haben 2001 geschätzt, dass 40 bis 50 % des Aufwands in Softwareprojekten in vermeidbare Nacharbeit fließen, eine der zwei Hauptquellen: „*hastily specified requirements*“.⁸⁷ **Eine hastig geschriebene Spec ist auch dann hastig, wenn ein Modell sie in Sekunden geschrieben hat.**

5.4

Wer hält die Spec aktuell?

Specs zwischen Teams altern. Die Branche hat zwei Antworten: eine Konvention und ein Verfahren.

Anm. 88

Preston-Werner 2013 [98].

Die Konvention ist *Semantic Versioning 2.0.0* (18. Juni 2013). Major für inkompatible Änderungen, Minor für kompatible Erweiterungen, Patch für Fehlerbehebungen.⁸⁸ **Semver ist ein Versprechen, kein Mechanismus.**

Anm. 89

Raemaekers, van Deursen, Visser 2014 [99].

Anm. 90

Ochoa et al. 2022 [100].

Steven Raemaekers et al. haben 2014 in Maven Central gefunden, dass rund ein Drittel aller Releases mindestens einen Breaking Change enthielt.⁸⁹ Lina Ochoa et al. fanden 2022 mit strengeren Filtern, dass 83,4 % der Upgrades Semver einhielten.⁹⁰ **Den Unterschied machen das Vergleichswerkzeug und die Auswahl der Artefakte.**

Anm. 91

Yasmin, Tian, Yang 2020 [101].

Jerin Yasmin et al. haben 2020 2.224 OpenAPI-Spezifikationen untersucht. Von 251 API-Versionen mit Breaking Change hatten 87,3 % in der Vorgängerversion keine Operation als veraltet markiert. Nur 3 von 219 APIs informierten Entwickler, wenn Client-Code eine veraltete Operation aufrief.⁹¹

Anm. 92

Sato 2014 [102].

Das Verfahren heißt *Parallel Change* oder *Expand and Contract* (Danilo Sato, 13. Mai 2014; Joshua Kerievsky, 2006). Erst wird die Schnittstelle erweitert, dann ziehen alle Aufrufer um, dann wird die alte Form entfernt.⁹² **Beides funktioniert nur, wenn jemand dafür zuständig ist.** Auf Stufe 1 fehlt diese Eigenschaft per Definition.

06

Was eine Spec verschwendet

6.1

Die sieben Verschwendungen, an der Spec gemessen

Anm. 93

LEI [95]; Brunt 2021 [103]; Ohno 1988 [94].

Lean liefert die naheliegende Linse. Schon der Kanon ist weniger fest, als er klingt. Das Lean Enterprise Institute listet Ohnos sieben Verschwendungen. Die Lean Enterprise Academy zitiert Ohno mit „*time on hand*“, „*stock on hand*“ und „*making defective products*“. Eine achte Verschwendung kam später dazu.⁹³

Vier der Kategorien treffen die Spec:

Anm. 94

Hill 2026 [65].

- **Überproduktion:** die Spec, die niemand liest
- **Inventar:** die Spec, die liegt
- **Wartezeit:** die Backlog-Kopplung zwischen Teams
- **Nacharbeit:** in Hills Daten mit Spec etwas häufiger, was Hill mit schwierigeren Aufgaben erklärt⁹⁴

Anm. 95

Poppendieck, Poppendieck 2006 [104].

Mary und Tom Poppendieck rechnen 2006 Dokumentation, die noch nicht in Code umgesetzt ist, zur halb erledigten Arbeit. Je länger sie liege, desto eher müsse sie geändert werden. Wer ständig Anforderungen ändern müsse, habe sie zu früh geschrieben. Und bei jeder Übergabe bleibe ein großer Teil des impliziten Wissens zurück.⁹⁵

Anm. 96

Sedano, Ralph, Péraire 2017 [96].

Sedano et al. haben bei Pivotal Übergaben und Phasendokumente als Verschwendung gar nicht beobachtet.⁹⁶ **Ob Spec-Dokumente Verschwendung sind, ist damit nicht belegt. Belegt ist, dass eine schlanke Organisation sie nicht braucht, um zu liefern.**

6.2

Wie lange dauert eine Spezifikation wirklich?

Für die Durchlaufzeit von Spezifikationsarbeit gibt es keine veröffentlichte Zahl. Was es gibt, ist ein Werkzeug, um es zu messen.

Anm. 97

Little 1961 [105].

Little's Law (1961 von John Little bewiesen) verbindet die Zahl der Einheiten im System mit Ankunftsrate und Verweilzeit.⁹⁷ In der Kanban-Praxis: **Durchlaufzeit = Arbeit in Bearbeitung / Durchsatz**. Eine Spec, die geschrieben, aber nicht umgesetzt ist, zählt als Arbeit in Bearbeitung. Wer zwanzig Specs gleichzeitig offen hat, verlängert die Durchlaufzeit jeder einzelnen.

Anm. 98

Keyte, Locher 2004 [106].

Beau Keyte und Drew Locher führten 2004 „percent complete and accurate“ (%C&A) ein. Sie misst den Anteil der Arbeit, den der nachgelagerte Schritt **ohne Korrektur, Ergänzung oder Nachfrage** verwenden kann.⁹⁸ **Für eine Spec ist das die entscheidende Frage.**

Anm. 99

AWS 2023 [107].

Die Erhebung ist weich. Die AWS Prescriptive Guidance zum Value Stream Mapping schreibt, die Daten sollten aus „*anecdotal average scenarios*“ kommen.⁹⁹ Die Zahlen, mit denen Lean im Softwarekontext sonst begründet wird, sind oft noch weicher. **Eine Messung der Spec-Arbeit wäre möglich, %C&A beim Empfänger und Little's Law für die offenen Specs. Dass sie fehlt, sagt etwas über den Markt. Wer Wirkung verkauft, misst sie nicht.**

6.3

Verifikation und Validierung

Anm. 100

Boehm 1984 [108].

Die Anforderungstechnik unterscheidet seit Jahrzehnten Verifikation und Validierung. Verifikation fragt, ob wir es richtig bauen. Validierung fragt, ob wir das Richtige bauen.¹⁰⁰ **Fast alles, was Spec-Werkzeuge anbieten, ist Verifikation.** Ob die Anforderungen stimmen, prüft keines.

Anm. 101

Brooker 2026 [109].

Der stärkste Einwand gegen die Werkzeuge betrifft die Chargengröße. Ein Werkzeug, das vor dem ersten Code eine vollständige Spec verlangt, verschiebt die Entscheidung nach vorn, an einen Punkt, an dem sie noch niemand treffen kann. Marc Brooker (AWS) hat am 9. April 2026 das Gegenteil empfohlen: *„Specifications aren’t up-front designs because you don’t need to, and probably shouldn’t, develop the entire specification upfront.“*¹⁰¹ **Das ist die brauchbarste Verteidigung des Ansatzes.**

6.4

Falsche Anforderung, sauber automatisiert

Anm. 102

Lions 1996 [110].

Am 4. Juni 1996 ist die erste Ariane 5 rund 40 Sekunden nach dem Start zerstört worden. Der Untersuchungsbericht (Jacques-Louis Lions, 19. Juli 1996) beschreibt eine Kette, die man als Lehrstück über Spezifikation lesen kann.¹⁰²

Anm. 103

Lions 1996 [110].

Eine Konvertierung von 64-Bit-Gleitkomma auf 16-Bit-Ganzzahl in der Inertialplattform lief über. Sie war nicht geschützt, weil man für den Rechner eine Auslastung von höchstens 80 % erreichen wollte. Die Begründung stand nicht im Code und war *„essentially obscured, though not deliberately, from any external review“*. Die Funktion hatte auf der Ariane 5 keinen Zweck mehr und lief nur weiter, weil sie aus der Ariane 4 übernommen war. Der entscheidende Satz: ***„it was jointly agreed not to include the Ariane 5 trajectory data in the SRI requirements and specification“***. Die Annahme, die verletzt wurde, stand nie in der Spezifikation.¹⁰³

Anm. 104

Lions 1996 [110].

Die Spezifikation der Fehlerbehandlung verlangte, den Prozessor bei jeder erkannten Ausnahme abzuschalten. *„The supplier of the SRI was only following the specification given to it“*. Weil in beiden Rechnern dieselbe Software lief, schalteten sich zwei gesunde Einheiten kurz nacheinander ab.¹⁰⁴ **Die Spec wurde korrekt umgesetzt. Sie war falsch.**

Anm. 105

MCO MIB 1999 [111].

Drei Jahre später ging der Mars Climate Orbiter verloren, weil ein Modul der Bodensoftware Impulse in Pound-force-Sekunden schrieb, während die Schnittstellenspezifikation Newton-Sekunden verlangte.¹⁰⁵ **Zwei Programme, die jedes für sich funktionierten, eine Datei ohne Einheitenfeld und eine Schnittstelle, die niemandem gehörte.**

Garbage in, garbage out ist ein alter Satz. Neu ist, wie sauber der Ausgang inzwischen aussieht. Ein Agent, der eine falsche Anforderung umsetzt, schreibt dazu Tests, die bestehen, und eine Zusammenfassung, die überzeugt. Je besser die Automatisierung, desto glaubwürdiger der Fehler.

6.5

Anm. 106

ISO/IEC/IEEE 29148:2018 [15].

Anm. 107

Berry 2007 [112].

Anm. 108

de Bruijn, Dekkers 2010 [113].

Anm. 109

Kamsties, Berry, Paech 2001 [114];
Kamsties et al. 2001 [91].

Anm. 110

Chantree et al. 2006 [115].

Anm. 111

Femmer et al. 2017 [116].

Klarheit schlägt Tooling

ISO/IEC/IEEE 29148:2018 verlangt für eine einzelne Anforderung neun Merkmale, darunter „unambiguous“ und „verifiable“. ¹⁰⁶ Daniel Berry hält dagegen: „*there is no such thing as an unambiguous specification. There are useful specifications!*“¹⁰⁷ **Die Norm setzt einen Grenzwert, keinen erreichbaren Zustand.**

Fabian de Bruijn und Hans Dekkers haben 2010 ein abgebrochenes Projekt mit rund 21 Personenjahren untersucht. Je nach Prüfverfahren waren zwischen 66 und 94 von 102 Anforderungen mehrdeutig. Von 40 Hauptproblemen ging genau eines auf Mehrdeutigkeit zurück, es erklärte 2 % der Budgetüberschreitung.¹⁰⁸ **Die Annahme, mehrdeutige Anforderungen ließen Projekte scheitern, hat kaum empirische Daten.**

Kamsties, Berry und Paech haben 2001 in Experimenten mit Studierenden gezeigt, dass Inspektionen nur 18 bis 25 % der bekannten Mehrdeutigkeiten finden. Ein anderes Experiment, das sie zitieren, fand, dass 72 % der Mehrdeutigkeiten trotzdem richtig gelesen werden.¹⁰⁹ Francis Chantree et al. haben 2006 vorgeschlagen, Mehrdeutigkeit als Eigenschaft eines Textes gegenüber seinen Lesern zu verstehen.¹¹⁰ **Lesen alle dasselbe, ist sie harmlos. Gefährlich wird sie, wenn Leser verschieden lesen und es keiner merkt.**

Werkzeuge zur automatischen Erkennung finden viel und liegen oft daneben. Henning Femmer et al. kamen 2017 bei der Erkennung von Requirements Smells auf eine mittlere Precision von 59 %.¹¹¹

Der Engpass liegt bei den Lesern. Mit besserer Erkennung ist wenig gewonnen. Entscheidend ist nicht, ob ein Satz mehrdeutig ist, sondern ob verschiedene Leser verschieden lesen und ob es einen Ort gibt, an dem das auffällt. In einem Team, das seine Fachsprache teilt, schadet eine mehrdeutige Spec wenig. In einem Team ohne gemeinsame Sprache hilft auch eine eindeutige wenig.

Anm. 112

Lenberg, Feldt 2018 [117].

Per Lenberg und Robert Feldt haben 2018 bei 217 Beschäftigten aus 38 Teams gefunden, dass Normklarheit die selbst eingeschätzte Teamleistung stärker vorhersagt als psychologische Sicherheit.¹¹² „Klarheit schlägt Tooling“ ist ein begründeter Satz, kein gemessener.

6.6

Anm. 113

Adzic 2011 [34].

Ein Beispiel ist mehr wert als eine Regel

Zwischen Prosa und Formalismus liegt eine Form, die beides verbindet. Gojko Adzic hat sie 2011 in „*Specification by Example*“ beschrieben – basierend auf Fallstudien zu über 50 Projekten.¹¹³ Anforderungen werden an konkreten Beispielen geklärt, gemeinsam mit der Fachseite, und diese Beispiele bleiben lesbar, *auch wenn sie ausführbar sind*.

Anm. 114

Adzic 2020 [35].

Zehn Jahre später befragte Adzic 514 Personen. Wer Beispiele als Akzeptanzkriterien nutzt, bewertet die eigene Produktqualität zu 22 % als „great“ – ohne Beispiele sind es nur 8 %. Adzic relativiert: Das sei Korrelation, die Stichprobe stamme aus seinen eigenen Kanälen.¹¹⁴ **Doch zwei Randbefunde sind aufschlussreicher:**

- 71 % nutzen Given-When-Then, doch rund ein Drittel automatisiert die Beispiele *gar nicht*.
- Als wichtigste „*source of truth*“ nannten 57 % ein Ticket-Tool wie Jira – nur 12 % Textdateien im Repository.

Anm. 115

Adzic 2020 [35].

Adzics Fazit: „*Collaboration won over testing*“.¹¹⁵

Das ist die entscheidende Beobachtung für den Reifegrad. Ein Beispiel verlangt keinen definierten Prozess. Es verlangt ein Gespräch, in dem jemand von der Fachseite sagt, was in einem konkreten Fall passieren soll. Der Test entsteht als Nebenprodukt. Eine Organisation auf Stufe 1 kann morgen damit anfangen. **Und sie bekommt das, was ihr fehlt: einen Ort, an dem Mehrdeutigkeit auffällt.**

07

Was andere Branchen schon wissen

SPEZIFIKATION IST keine Erfindung der Softwareentwicklung. Branchen, in denen ein Fehler ein Gebäude, ein Flugzeug oder einen Prozess kostet, schreiben seit Jahrzehnten auf, was gebaut werden soll. Der Blick dorthin lohnt sich nicht, weil diese Branchen es richtig machen. Er lohnt sich, weil sie wissen, was es kostet.

7.1

Anm. 116

HOAI 2021 [118]; DIN 276:2018 [119].

Anm. 117

ISO 19650-1:2018 [120].

Anm. 118

BMI, BMVg 2021 [121].

Anm. 119

§§ 650b, 650c BGB [122].

Anm. 120

Flyvbjerg 2014 [123].

Anm. 121

Flyvbjerg 2014 [123].

Bauwesen: Leistungsphasen und Nachtrag

Das deutsche Bauwesen hat alles, was Spec-Driven Development verspricht. Die HOAI 2021 teilt die Planung in neun Leistungsphasen mit festen Prozentsätzen je Phase. DIN 276 gliedert die Kosten in acht Gruppen.¹¹⁶ ISO 19650 beschreibt seit Dezember 2018 ein gemeinsames Bauwerksmodell mit benannten Informationsanforderungen und einer gemeinsamen Datenumgebung als „*agreed source of information*“.¹¹⁷ Für Bundesbauten ist BIM seit Ende 2022 in einer ersten Stufe verbindlich.¹¹⁸ Das BGB regelt seit 2018, was bei einer Änderung passiert.¹¹⁹

Geliefert wird trotzdem nicht. Bent Flyvbjerg hat 2014 festgehalten: „*Nine out of ten such projects have cost overruns*“, und das habe sich über 70 Jahre nicht verbessert.¹²⁰ Er erklärt das mit Optimismus, Planungsfehlschluss und bewusstem Kleinrechnen. **In seiner Liste der größten Überschreitungen stehen gleich drei Softwareprojekte:** das kanadische Waffenregister (590 %), das Medicare Transaction System (560 %) und das IT-System des britischen Gesundheitsdienstes (550 %).¹²¹ **Die Branche mit den formalisiertesten Specs beweist nicht, dass Specs nichts nützen. Sie beweist, dass Spec-Disziplin allein nichts erzwingt.**

Drei Dinge aus dem Bauwesen tragen trotzdem:

Erstens: die Verteilung des Aufwands. Bei Gebäuden trägt in der HOAI nicht die Ausführungsplanung (25 %) den größten Honoraranteil, sondern die Objektüberwachung (32 %).¹²² **Die Branche bezahlt das Hinschauen, ob gebaut wird, was geplant war, höher als den Plan.**

Anm. 123

§§ 650b, 650c BGB [122].

Zweitens: das Änderungsverfahren. Nach § 650b BGB kann der Besteller eine Änderung verlangen, der Unternehmer legt ein Angebot vor. Einigen sie sich nicht binnen 30 Tagen, kann der Besteller in Textform anordnen. Nach § 650c darf der Unternehmer dann bei Abschlagszahlungen 80 % der Mehrvergütung aus seinem Angebot ansetzen, bevor über den Preis entschieden ist.¹²³ **Eine Spec-Änderung hat dort einen Preis, eine Frist, eine Form und einen Zuständigen.** In Software ist der Normalfall, dass das Team die Änderung schluckt.

Anm. 124

BMI, BMVg 2021 [121].

Anm. 125

ISO 19650-1:2018 [120].

Drittens: der Bestand als eigenes Dokument. Die Branche führt neben dem Plan eine As-built-Dokumentation dessen, was tatsächlich gebaut wurde, mit Soll-Ist-Vergleich.¹²⁴ Drift ist dort kein Versagen, sondern ein benannter Zustand mit eigener Dokumentgattung. ISO 19650 begründet ihren „level of information need“ mit: „One purpose of defining the level of information need is to prevent delivery of too much information.“¹²⁵ **Der Schutz gegen Überproduktion steht in der Bauwerksnorm.**

Anm. 126

Gallaher et al. 2004 [124].

Wo die Spezifikation im Bauwesen Geld kostet, zeigt eine NIST-Studie von 2004. Michael Gallaher et al. haben die Kosten unzureichender Interoperabilität für 2002 auf 15,8 Mrd. Dollar hochgerechnet. Zwei Drittel tragen Eigentümer und Betreiber, 9 Mrd. allein im Betrieb.¹²⁶ **Die teuerste Spec-Lücke entsteht nicht im Entwurf, sondern bei der Übergabe an die, die das Bauwerk danach betreiben.**

Anm. 127

Reeves 1992 [125].

Gegen die Analogie steht ein Text von 1992. Jack Reeves argumentierte im *C++ Journal*, dass Programmieren nicht Bauen ist, sondern Entwerfen: „Software is cheap to build.“¹²⁷ **Die Analogie scheitert genau dort, wo Spec-Driven Development sie braucht: an der Annahme, es gebe ein Dokument, das den Bau vollständig beschreibt, bevor er beginnt.**

7.2

Luftfahrt und Medizin: Nachweis statt Meinung

Anm. 128

FAA 2017 [126]; RTCA 2011a [127].

Anm. 129

RTCA 2011a [127]; RTCA 2011b [128].

Die Luftfahrt verlangt keine bestimmte Methode, sie verlangt einen Nachweis. Das FAA Advisory Circular AC 20-115D (21. Juli 2017) erkennt DO-178C (13. Dezember 2011) an und beschreibt sich selbst als „an acceptable means, but not the only means“.¹²⁸ **Freiwillig in der Form, verbindlich in der Wirkung.** Fünf Software-Level von A bis E nach der Schwere der Fehlerfolge. Je höher der Level, desto mehr Ziele müssen unabhängig verifiziert werden. DO-333 erlaubt, einen Teil dieser Ziele durch Beweis statt durch Test zu erfüllen.¹²⁹

Anm. 130

IEC 62304:2006 [129].

Die Medizin arbeitet ähnlich. IEC 62304 (2006) teilt Software in Sicherheitsklassen A, B und C.¹³⁰ Die FDA hat am 14. Juni 2023 ihre Vorgaben auf zwei Dokumentationsstufen umgestellt: Basic und Enhanced.¹³¹ **Die Methode bleibt frei.**

Wie wenig ein vollständiger Nachweis allein bedeutet, zeigen zwei Fälle:

Anm. 132

Leveson, Turner 1993 [131].

Für die Therac-25 (Juni 1985 – Januar 1987) lag eine Sicherheitsanalyse vor. Nancy Leveson und Clark Turner zitieren 1993: „*Any residual software errors are not included in the analysis.*“ **Die Analyse schloss die Software aus, auf der fast die ganze Sicherheitsverantwortung lag.** Der selbe Fehler steckte in der Therac-20 und richtete dort keinen Schaden an, weil die Hardware ihn abfing.¹³²

Anm. 133

JATR 2019 [132].

Anm. 134

DOJ 2021 [133].

Beim 737 MAX war die Zertifizierungsdokumentation vorhanden. Der Joint Authorities Technical Review (Christopher Hart, Oktober 2019) fand, MCAS sei „*not evaluated as a complete and integrated function*“, und die „*extensive and fragmented documentation*“ habe es erschwert zu beurteilen, ob die Konformität vollständig nachgewiesen war. Die FAA hatte die Prüfung zu großen Teilen an den Hersteller delegiert.¹³³ Am 7. Januar 2021 akzeptierte Boeing eine Vereinbarung mit dem US-Justizministerium über mehr als 2,5 Mrd. Dollar.¹³⁴ **346 Menschen waren tot.**

Anm. 135

Lions 1996 [110].

Das Ariane-Gremium fasste das 1996 in eine Empfehlung: Reviews sollten „*the substance of arguments, rather than check that verifications have been made*“ prüfen. **Und es drehte die Beweislast um:** Software solle nicht als korrekt gelten, bis ein Fehler gezeigt ist, sondern als fehlerhaft, bis anerkannte Methoden ihre Korrektheit zeigen.¹³⁵

Anm. 136

Haynes et al. 2009 [134].

Anm. 137

Urbach et al. 2014 [135].

Anm. 138

Leape 2014 [136].

Dass dieselbe Form zwei Wirkungen haben kann, zeigt die Chirurgie. Alex Haynes, Atul Gawande et al. haben 2009 eine Checkliste mit 19 Punkten in acht Krankenhäusern eingeführt, die Sterblichkeit fiel von 1,5 auf 0,8 %.¹³⁶ Ontario hat die Checkliste 2010 flächendeckend in allen Krankenhäusern eingeführt, fast überall unverändert. David Urbach et al. haben 2014 über 200.000 Eingriffe in 101 Krankenhäusern ausgewertet und keine signifikante Veränderung der Sterblichkeit gefunden.¹³⁷ **Dieselbe Liste, zwei Ergebnisse. Den Unterschied machte, wie die Liste zustande kam.**¹³⁸

7.3

Recht: Bestimmtheit und Konfliktlösung

Das Recht kennt das Problem mehrdeutiger Texte am längsten – und löst es anders, als man erwartet. **Nicht mehr Text schafft Klarheit, sondern klare Zuständigkeit.**

Anm. 139
§ 133 BGB [137].

Anm. 140
CISG 1980 [138].

Ein Vertrag muss nicht eindeutig sein, um gültig zu sein. Nach § 133 BGB „ist der wirkliche Wille zu erforschen und nicht an dem buchstäblichen Sinne des Ausdrucks zu haften“. ¹³⁹ Das UN-Kaufrecht (1980, Artikel 8) zieht das „spätere Verhalten“ der Parteien als Auslegungsmittel heran. ¹⁴⁰ **In dieser Perspektive ist Drift nicht Abweichung, sondern eine mögliche Lesart der Spec.**

Anm. 141
BGH 2021 [139].

Anm. 142
Hart, Moore 1990 [140].

Mehr Bestimmtheit ist nicht automatisch besser. Der BGH urteilte am 8. September 2021: Eine weitere Präzisierung sei „weder möglich noch zumutbar“, weil sie „die Gefahr der Unvollständigkeit in sich bergen würde“. ¹⁴¹ **Nachschärfen macht einen Text, der auf Unvorhersehbares reagieren soll, nicht robuster – sondern lückenhaft.** Oliver Hart und John Moore (1990): Was ein Vertrag nicht festlegt, entscheidet der Eigentümer. Er hat die Restverfügung. ¹⁴² **Unvollständige Verträge sind kein Mangel, sondern der Normalfall.**

Anm. 143
§ 305c BGB [141].

Anm. 144
RL 93/13/EWG [142].

Für die Auslegung hat das Recht klare Regeln. § 305c Abs. 2 BGB: Zweifel gehen zu Lasten dessen, der die AGB verwendet. ¹⁴³ Die EU-Richtlinie 93/13/EWG führt Klauseln als missbräuchlich, durch die dem Gewerbetreibenden „das ausschließliche Recht zugestanden wird, die Auslegung einer Vertragsklausel vorzunehmen“. ¹⁴⁴ **Wer den Text schreibt, trägt die Unklarheit und darf nicht ihr letzter Ausleger sein.**

Anm. 145
BMJ 2024 [143].

Auch für Versionen hat die Gesetzgebung eine Technik. Das Handbuch der Rechtsförmlichkeit (4. Auflage, 2024) unterscheidet statische und dynamische Verweisung. Eine dynamische Verweisung sei „der Entwicklung ihrer Bezugsquelle ausgeliefert“. ¹⁴⁵ **Eine Spec, die immer gilt und sich ständig ändert, wäre kein taugliches Bezugsobjekt.**

Und wenn es Streit gibt, gibt es ein Forum. Ein Vertrag ohne Forum ist eine Absichtserklärung. Eine Spec zwischen Teams hat fast nie eines.

7.4

Produktdesign: Appetite statt Umfang

Anm. 146
Singer 2019 [144].

Ryan Singer hat im Juli 2019 „*Shape Up*“ veröffentlicht, die Arbeitsweise von Basecamp. Ein geformter Entwurf beschreibt Grenzen und Regeln, ausdrücklich keine Lösung im Detail: „*This isn’t a spec. It’s more like the boundaries and rules of a game.*“¹⁴⁶

Anm. 147
Singer 2019 [144].

Statt einer Schätzung steht am Anfang ein *Appetit*: wie viel Zeit das Vorhaben wert ist. „*Estimates start with a design and end with a number. Appetites start with a number and end with a design.*“ Ein bis zwei Wochen oder sechs, feste Zeit, variabler Umfang. Wer in einem Zyklus nicht fertig wird, bekommt im Normalfall keine Verlängerung. Singer nennt das einen *Circuit Breaker*.¹⁴⁷

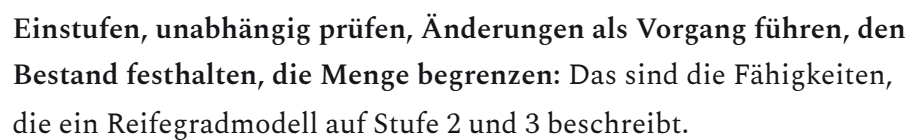
Shape Up ist eine Praktikererzählung ohne veröffentlichte Messung. Für den Essay zählt ein Gedanke: *Shape Up* begrenzt Überproduktion nicht durch eine bessere Spec, sondern durch ein Budget, das vor der Spec feststeht. Das ist dieselbe Idee wie der „*level of information need*“ im Bauwesen.

7.5 ————— **Fünf Bräuche, ein Muster**

Tab. 03 Fünf Bräuche, ein Muster

Brauch	Branche	Was er regelt	Gegenstück in Software	Wo es meist fehlt
Einstufung nach Folge	Luftfahrt, Medizin, Automotive	wie viel Nachweis ein Teil braucht	Risikoklassen für Änderungen	Specs haben eine Tiefe für alles
Unabhängige Prüfung	Luftfahrt, Bauüberwachung	wer prüft, ob gebaut wurde, was beschrieben ist	Review, Gates im Build	Prüfung durch den, der die Spec geschrieben hat
Änderungsverfahren	Bauwesen, Recht	Preis, Frist, Form und Zuständigkeit	Deprecation, Expand and Contract	Änderung als Gespräch ohne Zuständigen
Bestand als eigenes Dokument	Bauwesen, Logistik	was tatsächlich gebaut ist, mit fester Fassung	gepinnte Vertragsversion, Diff gegen Produktion	„lebende“ Spec als Plan und Bestand
Budget vor dem Text	Bauwesen, Shape Up	wie viel Spec ein Vorhaben verträgt	Appetit, Informationsbedarf	Werkzeuge erzeugen so viel, wie das Modell schreibt

Branchen-Bräuche und ihr Gegenstück in Software



08

Ein SDD, das zum Reifegrad passt

BIS HIERHER wurde gesammelt und eingeordnet. Ab hier wird vorgeschlagen. *Der Vorschlag ist argumentiert, nicht erprobt* – und am Ende steht, woran er scheitern würde.

8.1

Die Weiche: acht Fragen

Der Ansatz ist einfach: **Bevor ein Team ein Spec-Werkzeug einführt, klärt es zwei Dinge:** Was soll die Spec leisten? Und kann die Organisation das tragen? Je nach Antwort braucht es eine andere Art von Spec. Ich nenne das **Routing** – weniger eine Methode, mehr eine Weiche.

Für diese Weiche braucht es kein formelles Appraisal. Es reichen **acht Fragen**, die ehrlich beantwortet werden müssen. Sie prüfen die Fähigkeiten ab, die das vorherige Kapitel als Voraussetzung aus anderen Branchen abgeleitet hat:

1. **Kann das Team in zwei Sätzen sagen, für wen es baut und welches Problem es löst** – und stimmt die Fachseite dem zu?
2. **Gibt es für jeden Vertrag an einer Teamgrenze jemanden, der entscheidet**, was er bedeutet, wenn zwei Teams ihn unterschiedlich interpretieren?
3. **Läuft eine Änderung an einem solchen Vertrag als strukturierter Vorgang** mit Ankündigung, Frist und Zuständigem – oder als Nachricht im Chat?
4. **Gibt es im Build eine Prüfung, die fehlschlägt**, wenn ein Vertrag bricht?
5. **Gelten Format, Prüfung und Änderungsablauf an allen Teamgrenzen gleich** – oder handelt jedes Teampaar sie selbst aus?
6. **Misst jemand, wie viel Nacharbeit nach der Umsetzung einer Spec anfällt?**
7. **Folgt aus dieser Messung etwas** – etwa eine andere Vorlage oder ein strengeres Gate?
8. **Wird eine Spec nach Abschluss der Änderung noch einmal gelesen** – und von wem?

Die erste Frage ist eine Vorfrage. Wer sie nicht beantworten kann, hat ein fachliches Problem, kein Spezifikationsproblem. Die Fragen 2 bis 7 markieren die Übergänge zwischen den Stufen. Wer die zweite und dritte mit Nein beantwortet, arbeitet an seinen Grenzen auf Stufe 1, wer beide mit Ja beantwortet, auf Stufe 2. Wer bei der vierten Nein sagt, hat Verträge, aber keine Prüfung. Die vierte und fünfte trennen Stufe 2 von Stufe 3: Auf Stufe 2 hat jede Grenze ihre eigene Regel, auf

Stufe 3 gilt eine für alle, und der Build setzt sie durch. Die sechste trennt Stufe 3 von Stufe 4. **Wer sie mit Nein beantwortet, kann nicht feststellen, ob eine Spec etwas gebracht hat.** Die siebte trennt Stufe 4 von Stufe 5: Gemessen wird, damit sich etwas ändert.

Die achte Frage entscheidet keine Stufe, sondern die Lebensdauer. Liest niemand eine Spec nach der Änderung, bleibt sie spec-first und darf weggeworfen werden. Spec-anchored lohnt sich nur, wenn jemand sie liest und mit dem Code ändert.

Die Stufen sind dabei Kürzel für diese Antworten. Ein Zertifikat braucht dafür niemand. Ein Team kann für seine Schnittstellen auf Stufe 3 sein und für seine Fachlichkeit auf Stufe 1. Dann bekommt es für beides etwas anderes.

Das Modell trägt trotzdem etwas bei, das die Fragen allein nicht hergeben: **eine Reihenfolge.** Ein Breaking-Change-Gate hilft erst, wenn jemand entscheidet, was nach einem roten Build passiert. Messen lohnt sich erst, wenn ein Ablauf wiederholbar ist. **Daran lässt sich ablesen, wann ein Werkzeug zu früh kommt.**

8.2

Stufe 1 und 2: die Fachvision zuerst

Auf Stufe 1 und 2 ist die Spec ein Mittel zur Verständigung. Ihr Leser ist ein Mensch, meist von der Fachseite. Ihr Zweck: dass Mehrdeutigkeit auffällt, solange sie billig ist.

Anm. 148

Evans 2015 [45].

Anm. 149

Brandolini 2013 [145]; Hofer, Schwentner 2021 [146].

Am Anfang steht die Sprache. Eric Evans: Die Ubiquitous Language ist „*a language structured around the domain model and used by all team members within a bounded context*“. Und: „*Recognize that a change in the language is a change to the model.*“¹⁴⁸ Formate dafür: Event Storming (Alberto Brandolini, seit 2013), Domain Storytelling (Stefan Hofer, Henning Schwentner).¹⁴⁹ **Sie liefern kein Dokument, das ein Agent umsetzen kann. Sie liefern das, was ein solches Dokument voraussetzt.**

Anm. 150

Adzic 2020 [35].

Danach kommen Beispiele. *Specification by Example* funktioniert ohne Werkzeug und ohne definierten Prozess. Sie bringt die Fachseite an den Tisch. Ob die Beispiele automatisiert werden, ist zweitrangig. **Rund ein Drittel der Teams in Adzics Umfrage tut es nicht,** und die Bilanz „*Collaboration won over testing*“ sagt, worauf es ankommt.¹⁵⁰

Anm. 151

Singer 2019 [144].

Für die Frage, was überhaupt gebaut wird, passt ein *Appetit* nach Shape Up besser als eine vollständige Spec. Er legt fest, wie viel ein Vorhaben wert ist, bevor jemand anfängt, es zu beschreiben. Er

bremst die Überproduktion, die Spec-Werkzeuge von sich aus erzeugen.¹⁵¹ Entscheidungen, die schwer umkehrbar sind, gehören in ein ADR. Wenige, kurz, mit Begründung. Mehr Dokumentation braucht eine Organisation auf dieser Stufe nicht – und mehr kann sie auch nicht pflegen.

Anm. 152

Gloaguen et al. 2026 [66].

Anm. 153

Böckeler 2025 [1].

Auch auf Stufe 1 arbeiten heute Agenten im Code. Die ETH-Studie zu AGENTS.md gibt die beste Regel: **Explizite Anweisungen werden befolgt, beschreibende Übersichten bringen nichts und kosten mehr.**¹⁵² Eine Spec für einen Agenten auf dieser Stufe ist kurz, konkret und wegwerfbar. Was gebaut werden soll, welche Beispiele gelten müssen, welche Grenzen nicht verletzt werden dürfen. **Spec-first im Sinne von Böckeler – und das ist hier keine Schwäche, sondern ehrlich.**¹⁵³

8.3

Stufe 3 und darüber: die technische Spec

Ab Stufe 3 gibt es, was die technische Spec braucht: definierte Abläufe für Änderungen, einen Zuständigen je Vertrag, eine Pipeline, die man um eine Prüfung erweitern kann. **Hier lohnt sich das, was der Markt heute als Werkzeugkette verkauft – weil jetzt jemand da ist, der es betreibt.**

Anm. 154

Evans 2015 [45].

Der Vertrag liegt im Repository, in einem maschinenlesbaren Format. Er ist die *Published Language* im Sinne von Evans: „*a well-documented shared language that can express the necessary domain information as a common medium of communication*“. Evans warnt: Wird ein internes Modell direkt als Austauschformat benutzt, „*it essentially becomes frozen*“.¹⁵⁴ **Der Vertrag zwischen Teams ist nicht das interne Modell eines Teams, sondern eine eigene, bewusst gepflegte Sprache an der Grenze.**

Anm. 155

Buf [24]; oasdiff [64].

Anm. 156

Confluent [76].

Anm. 157

Hatfield-Dodds, Dygalo 2022 [63].

Anm. 158

PactFlow [147]; Fellows 2022 [62].

Geprüft wird im Build. `buf breaking` für Protobuf oder `oasdiff` für OpenAPI vergleichen jede Änderung gegen den letzten veröffentlichten Stand und lassen den Build scheitern, wenn etwas bricht.¹⁵⁵ Für Ereignisse gehört ein transitiver Kompatibilitätsmodus in die Schema Registry.¹⁵⁶ Property-basierte Tests aus dem Schema (z. B. mit Schemathesis) prüfen die Implementierung gegen den Vertrag. **Das ist das einzige Glied der Kette mit begutachtetem Nutzen.**¹⁵⁷ Consumer-Driven Contracts nach Pact lohnen sich dort, wo wenige Konsumenten mit klaren Erwartungen auf einen Anbieter treffen. Bei vielen Konsumenten verweist PactFlow auf den statischen Vergleich.¹⁵⁸

Anm. 159

Sato 2014 [102].

Änderungen laufen als *Expand and Contract*, mit Ankündigung im Vertrag, einer Frist und jemandem, der die Contract-Phase durchsetzt.¹⁵⁹ Die Spec ist *spec-anchored*: Sie lebt mit dem Code, und ein Diff gegen die Produktion zeigt, wo sie abweicht. **Prosa-Specs für Agenten haben hier ihren besten Platz**, weil sie auf Verträge und Tests verweisen können, die eine Maschine prüft. Die Prosa sagt, was gewollt ist. Der Build sagt, ob es erreicht wurde.

Ab Stufe 4 kommt Messen dazu: %C&A beim Empfänger einer Spec, die Zahl offener Specs nach Little's Law, Nacharbeit nach der Umsetzung. **Erst hier lässt sich sagen, ob eine Spec einem Team etwas gebracht hat.** Das ist der Grund, warum Wirkungsversprechen von Werkzeugen für Teams unterhalb dieser Stufe nicht geglaubt werden. Sie könnten die Wirkung gar nicht feststellen. Stufe 5 heißt, dass aus den Zahlen etwas folgt. Bringt eine Form der Spec nichts, wird sie geändert oder abgeschafft.

8.4

Wo Agenten schreiben: Evals vor der Spec

Je mehr Code ein Agent schreibt, desto mehr hängt an der Prüfung. Die Prosa-Spec sagt ihm, was gewollt ist. Ob er es erreicht hat, sagen die Prüfungen, die nach ihm laufen. **Für Teams, die viel mit Agenten arbeiten, ist das der wichtigste Hebel des Vorschlags: Die Evals kommen vor der Spec.**

Anm. 160

McCarthy 2026 [148].

Anm. 161

Strathern 1997 [89].

Eval heißt hier eine Sammlung von Prüffällen mit erwartetem Ergebnis, die bei jedem Lauf mitläuft: Akzeptanzkriterien als Tests, Beispiele der Fachseite, Property-basierte Tests aus dem Schema, Vertragsprüfungen an den Grenzen. Zwei Regeln machen daraus mehr als eine Testsuite. Ein Teil der Fälle bleibt dem Agenten verborgen, wie bei StrongDM, damit er nicht auf die Prüfung hin optimiert.¹⁶⁰ Das ist Goodharts Effekt im Kleinen.¹⁶¹ Und die Evals laufen nicht nur, wenn sich der Code ändert, sondern auch, wenn sich Spec, Vorlage oder Modell ändern.

Die Evals folgen derselben Reihenfolge wie die Spec. Auf Stufe 1 und 2 sind es die Beispiele aus dem Gespräch mit der Fachseite, ausführbar gemacht. Ab Stufe 3 stehen sie als Gate im Build. Ab Stufe 4 wird gemessen, wie oft ein Lauf sie im ersten Anlauf besteht und wie viel Nacharbeit danach noch anfällt. **Ohne Evals liest der Agent die Spec, und niemand prüft, was er daraus macht.**

Anm. 162

OWASP 2023 [72].

Anm. 163

Beyer et al. 2016 [149].

Zwei Dinge gehören nicht in die Spec einer einzelnen Änderung, weil sie für jede gelten. Das erste sind nicht-funktionale Anforderungen. Autorisierung ist das deutlichste Beispiel: *Broken Object Level Authorization* führt die OWASP API Security Top 10 an, und im Vertrag steht davon nichts.¹⁶² Sie gehört als Regel an die Grenze, die für jeden Endpunkt gilt, und in eine Eval, die für jeden Endpunkt prüft, dass ein fremdes Objekt verweigert wird. Damit ist sie eine Leitplanke, wie sie der Abschnitt „Ermöglichen statt erzwingen“ beschreibt. Für Latenz und Verfügbarkeit leisten Service Level Objectives dasselbe.¹⁶³

Anm. 164

Jones 2023 [150]; Bitol 2026 [151].

Das zweite sind Daten. Ein Data Contract hält für einen Datensatz zwischen Teams fest, welches Schema er hat, was die Felder bedeuten, welche Qualitätsregeln gelten und wem er gehört. Geprüft wird er in der Pipeline.¹⁶⁴ Er ist die **Published Language für Daten** und setzt dieselbe Stufe voraus wie ein API-Vertrag: jemanden, der entscheidet, und einen Ablauf für Änderungen.

8.5

Welche Spec für welches Team

Aus den zwei Achsen (Prüfbarkeit und Lebensdauer) und der Frage nach dem Reifegrad ergibt sich eine Auswahl. Sie ist eine Faustregel, kein Algorithmus. Die Spalte „Setzt voraus“ nennt die Fragen der Wei- che, die ein Team mit Ja beantworten sollte.

Tab. 04

Welche Spec für welches Team

Ausgangslage	Setzt voraus	Was die Spec leisten soll	Form der Spec	Prüfung	Lebensdauer	Wer auslegt
Ein Team, eigener Code, Fachlichkeit unklar	nichts	Verständigung mit der Fachseite	Glossar, Beispiele, Appetit	Gespräch, Review mit Fachseite	spec-first, wegwerfen	Product Owner
Ein Team, eigener Code, Fachlichkeit klar	Fragen 1, 4	Agenten und neue Leute ausrichten	kurze Aufgaben-Spec, ausführbare Beispiele	Tests im Build	spec-first	Team
Mehrere Teams, Grenzen ohne Zuständige	nichts	Zuständigkeit klären	Liste der Verträge mit Eigentümer, Glossar je Grenze, ADR	Review durch beide Seiten	spec-first	noch niemand – das zu ändern ist die Aufgabe
Mehrere Teams, synchrone Schnittstellen	Fragen 2–5, 8	Abstimmung an der Grenze ersetzen	OpenAPI/Protobuf im Repository	Breaking-Change-Gate, Tests aus Schema	spec-anchored	Anbieter des Vertrags, Eskalation
Mehrere Teams, Ereignisse	Fragen 2–5, 8	Leser und Schreiber entkoppeln	AsyncAPI, Schemas in Registry	transitive Kompatibilität	spec-anchored, Fassungen fest	Eigentümer des Topics
Mehrere Teams, Daten für Auswertungen	Fragen 2–5, 8	Erzeuger und Auswerter entkoppeln	Data Contract: Schema, Bedeutung, Qualitätsregeln	Prüfung in der Pipeline	spec-anchored	Eigentümer des Datensatzes
Externe Konsumenten	Fragen 2–5, 8	Vertrauen und Planbarkeit	veröffentlichter Vertrag mit Version	Gate, Diff gegen Produktion, Deprecation-Frist	statisch je Version	Anbieter, mit Änderungsregel
Sicherheitskritischer Kern	Fragen 4, 8 + unabhängige Prüfung	Fehler im Entwurf finden	formales Modell, Vor-/Nachbedingungen	Model Checker, Property Tests	spec-anchored	unabhängige Prüfung
Agent arbeitet über Stunden allein	Fragen 1, 4	Ziel halten, ohne nachzufragen	Spec mit Akzeptanzkriterien als Tests	Evals mit vorgegebenen Prüffällen	je Lauf	wer den Lauf startet

Zwei Spalten sind wichtiger als die übrigen. Die Spalte „Prüfung“ entscheidet, ob die Spec Fehler verhindert oder nur koordiniert. Die Spalte „Wer auslegt“ entscheidet, ob sie überhaupt koordiniert. Eine Spec ohne Eintrag in der letzten Spalte ist nach dem, was das Recht dazu weiß, eine Absichtserklärung.

8.6

Ermöglichen statt erzwingen

Bleibt die Frage, wer das alles einführt. In größeren Organisationen landet sie bei einem Plattformteam. **Dort entscheidet sich, ob Spec-Driven Development hilft oder schadet.**

Anm. 165

Bottcher 2018 [11].

Anm. 166

Bukoski, Moyles, McGarr 2016 [152].

Anm. 167

Niemen 2020 [153].

Anm. 168

D. Evans 2025 [154].

Evan Bottcher hat 2018 die Regel aufgeschrieben: *„platforms must be compelling to use, they cannot stand on a mandate alone“*.¹⁶⁵ Netflix: Teams hätten die Freiheit, andere Lösungen zu bauen, *„but they also take on additional responsibility for maintaining these solutions“*.¹⁶⁶ Bei Spotify heißt das *Golden Path*.¹⁶⁷ Darren Evans (Google Cloud, 2025): *„A guardrail is not a guide rail; its purpose is to prevent a catastrophic event, not to direct the workflow.“*¹⁶⁸

Für Specs ergibt das eine klare Teilung:

- **Golden Path:** Die Vorlage für eine Spec, das Werkzeug, der Ablauf für Agenten. Das Plattformteam macht sie so gut, dass Teams sie freiwillig nehmen, und misst, ob sie es tun.
- **Leitplanke:** Das Breaking-Change-Gate an einer Teamgrenze. Es gilt für alle, weil ein gebrochener Vertrag andere trifft. Es verhindert einen Schaden, statt einen Weg vorzuschreiben.

Anm. 169

Fenton 2025 [155].

Steve Fenton: *„Make your platform optional, and your policies mandatory.“*¹⁶⁹

Anm. 170

Anjum 2026 [156].

Anm. 171

DORA 2024 [85].

Anm. 172

CNCf 2023 [157].

Die Belege für diese Teilung sind dünn. Eine begutachtete Übersicht über Plattform Engineering (4. Mai 2026) fand unter 88 Quellen nur zwei aus Tier-1-Venues mit dem Thema als Schwerpunkt. Zu Scorecards: *„no peer-reviewed empirical evidence of effectiveness“*. Der Autor empfiehlt trotzdem, *Golden Paths „as enablers instead of mandates“* zu bauen.¹⁷⁰ Der DORA-Bericht 2024 zeigt, was die Alternative kostet.¹⁷¹ Die CNCf hat ihrem Reifegradmodell für Plattformen (November 2023) einen Satz mitgegeben: *„reaching the highest level should not be a goal in itself“*.¹⁷²

8.7

Woran der Vorschlag scheitern würde

Ein Vorschlag, der nicht scheitern kann, ist keiner. Vier Beobachtungen würden diesen widerlegen oder einschränken:

Anm. 173

Hill 2026 [65].

Erstens: Ein sauberer Befund, dass Prosa-Specs auch in unreifen Organisationen Fehler und Nacharbeit senken. Hills Studie (ESEM 2026) enthält einen Hinweis: Wo keine KI-Nutzung erkennbar war, ging eine Spec mit weniger Nacharbeit einher. Hill selbst baut nicht darauf, und ob es an der Reife liegt oder an etwas anderem, sagt die Studie nicht.¹⁷³ **Wäre es unabhängig von der Reife, bräuchte es die Weiche nicht.**

Anm. 174

Urbach et al. 2014 [135]; Leape 2014 [136].

Zweitens: Dass die Fragen der Weiche sich nicht ehrlich beantworten lassen. Reifegrade, die man sich selbst bescheinigt, sind anfällig für Goodharts Effekt. Wenn Teams sich für die technische Spec auf Stufe 3 einordnen, weil das der attraktivere Weg ist, wird aus der Weiche eine Formalität wie die Checkliste in Ontario.¹⁷⁴

Drittens: Dass Organisationen auf Stufe 1 den Vorschlag als Erlaubnis lesen, gar nichts aufzuschreiben. „*Erst die Fachvision*“ ist eine Reihenfolge, keine Ausrede. **Wer nach einem Jahr noch kein Glossar, keine Beispiele und keine Zuständigen hat, ist nicht auf dem Weg zu Stufe 2.**

Anm. 175

Orlanski et al. 2026 [68].

Viertens: Dass Agenten die Fähigkeiten der Stufen selbst mitbringen. Wenn ein Agent zuverlässig nachfragt, wo eine Anforderung mehrdeutig ist, Verträge über Teamgrenzen hinweg prüft und Änderungen ankündigt, verschiebt sich die Grenze, an der eine technische Spec sich lohnt. Heute sieht man dafür keine Belege. SlopCodeBench zeigt eher das Gegenteil.¹⁷⁵ **Ausschließen kann man es nicht.**

09

Preis und Ausblick

9.1

Overhead und Überkonstruktion

Anm. 176

Fellows 2022 [62].

Jede Spec hat ihren Preis – auch die gute. Sie muss geschrieben, gelesen, abgestimmt und gepflegt werden. An jeder Teamgrenze kommt Synchronisation hinzu. PactFlow gibt zu: Der Nutzen stellt sich nicht sofort ein, und der konsumentengetriebene Ansatz skaliert nicht auf viele Konsumenten.¹⁷⁶

Anm. 177

Klein et al. 2009 [158]; Klein et al. 2014 [159].

Am oberen Ende der Skala steht seL4: 8.700 Zeilen C, bewiesen mit rund 200.000 Zeilen Isabelle/HOL, dazu 600 Zeilen Assembler, deren Korrektheit angenommen wurde. Der Aufwand lag bei rund 22 Person Jahren, Werkzeuge und Forschung eingeschlossen.¹⁷⁷ Amazon hält für ShardStore (40.000+ Zeilen Rust) volle Verifikation für „currently impractical“.¹⁷⁸

Anm. 178

Bornholt et al. 2021 [43].

Die Prosa-Spec hat ein anderes Kostenproblem: Sie wird nicht günstiger, nur weil sie genauer wird. Gonzalez' Versuch mit Symphony zeigt die Größenordnung. Die Spec umfasste etwa ein Sechstel der Implementierung.¹⁷⁹ **Die anderen fünf Sechstel hatte jemand entschieden – nur nicht in der Spec.** Wer alle Details aufschreiben wollte, hätte den Code ein zweites Mal geschrieben, in einer Sprache ohne Compiler. **Vor diesem Hintergrund ist Breslavs Satz, dass Leute keine Specs schreiben wollen, weniger eine Klage als eine Rechnung.**¹⁸⁰

Anm. 179

Gonzalez 2026 [70].

Anm. 180

Breslav 2026 [55].

Überkonstruktion ist die naheliegende Fehlform des Vorschlags. Wer die Matrix als Pflichtenheft liest, baut für jedes Team die anspruchsvollste Zeile. Die Matrix ist aber von oben nach unten teurer, und die meisten Teams gehören in die oberen Zeilen.

9.2

Wohin die Wertschöpfung wandert

Anm. 181

Osmani 2026 [160].

Wenn Code billig wird, wird etwas anderes knapp. Addy Osmani (Juli 2026): „*the fundamental constraint in a software factory isn't how much code we can churn out: it's how quickly we can verify it*“.¹⁸¹ Grove hatte 2025 die Spec zum eigentlichen Artefakt erklärt.¹⁸² **Was seitdem zu sehen ist, spricht dafür, dass die Prüfung das eigentliche Artefakt wird:** die Tests, die Verträge mit Prüfwert, die Gates im Build.

Anm. 182

Grove 2025 [10].

StrongDM hat im Februar 2026 beschrieben, wie sein KI-Team Software baut, ohne dass ein Mensch Code schreibt oder prüft. Die Eingabe sind Specs und Szenarien, dazu bewusst zurückgehaltene Testfälle und nachgebaute Abbilder fremder Dienste. Nach eigener Auss-

Anm. 183

McCarthy 2026 [148].

age hat eine solche Fabrik Luft nach oben, solange sie weniger als 1.000 Dollar Token pro Entwickler und Tag verbraucht.¹⁸³ **Bemerkenswert ist weniger die Zahl als die Beschreibung: Der Aufwand steckt vor allem in der Prüfung.**

Anm. 184

SpecOps 2026 [161]; Lahiri 2026 [162].

Anm. 185

Rosa et al. 2026 [163].

Die Forschung fängt gerade erst an. Der erste Workshop zum Thema, SpecOps 2026, fand am 6. Oktober 2026 in Oakland statt, mit einer Keynote von Shuvendu Lahiri.¹⁸⁴ Eine kontrollierte Probandenstudie zu spec-getriebener Codeerzeugung ist als Registered Report angenommen, Ergebnisse gibt es noch nicht.¹⁸⁵ **Im Oktober 2026 gibt es Tagungen zum Thema, aber keine Betriebsdaten.**

Anm. 186

Hart, Moore 1990 [140].

Die Wertschöpfung wandert nicht einfach von Code zu Spec. Sie wandert zu dem, was eine Maschine prüfen kann, und zu dem, was nur Menschen klären können: welches Problem es ist, was die Begriffe bedeuten, wer entscheidet, wenn zwei Teams es verschieden lesen. Hart und Moore würden sagen: Sie wandert zur Restverfügung über das, was nicht in der Spec steht.¹⁸⁶

10

Was bleibt

DIE FRAGE war: Unter welchen Bedingungen macht eine Spezifikation Software besser – und wann beschleunigt sie nur, was ohnehin schief läuft?

Die Antwort hat drei Teile:

Anm. 187

Newcombe et al. 2015 [42]; Hatfield-Dodds, Dygalo 2022 [63]; Tufano et al. 2026 [69].

Erstens: Eine Spec koordiniert – und das ist ihr belegter Nutzen. Sie richtet Menschen und Agenten aus, sie ersetzt Abstimmung an Teamgrenzen, sie macht Mehrdeutigkeit früher sichtbar, wenn jemand hinschaut. **Fehler verhindert sie nur, wo ein Algorithmus die Implementierung gegen sie prüft.** Für TLA+ bei Amazon, für Schemathesis, für Verträge mit Vor- und Nachbedingungen ist das belegt.¹⁸⁷ Für Markdown, das ein Modell liest, gibt es bis heute keine begutachtete Messung, die eine positive Wirkung zeigt – und die größte Messung zu Spec-Artefakten überhaupt findet keine.

- <https://aws.amazon.com/blogs/devops/amazon-q-developer-end-of-support-announcement/>
5. CTech (Calcalist), *Former Snyk founder Guy Podjarny secures \$125M for AI-powered software startup Tessl*, calcalistech.com, 14. November 2024. <https://www.calcalistech.com/ctechnews/article/b1pnxlmg1x>
 6. Joaquin Miller, Jishnu Mukerji (Hrsg.), *MDA Guide Version 1.0.1*, Object Management Group, omg/2003-06-01, 12. Juni 2003. <https://www.omg.org/cgi-bin/doc?omg/03-06-01>
 7. Dave A. Thomas, *MDA: Revenge of the Modelers or UML Utopia?*, IEEE Software 21(3), Mai/Juni 2004, S. 15–17. DOI: 10.1109/MS.2004.1293067
 8. Martin Fowler, *Model Driven Architecture*, Bliki, martinowler.com, 2. Februar 2004. <https://martinfowler.com/bliki/ModelDrivenArchitecture.html>
 9. Object Management Group, *MDA Guide rev. 2.0*, ormsc/2014-06-01, Juni 2014. <https://www.omg.org/cgi-bin/doc?ormsc/14-06-01>
 10. Sean Grove, *The New Code*, Vortrag auf der AI Engineer World's Fair, San Francisco, Juni 2025, Video vom 11. Juli 2025. <https://www.youtube.com/watch?v=8rABwKRsec4>
 11. Evan Bottcher, *What I Talk About When I Talk About Platforms*, martinowler.com, 5. März 2018. <https://martinfowler.com/articles/talk-about-platforms.html>
 12. APIContext, *OpenAPI Specifications in the Real World*, Whitepaper, 1. August 2024 (Herstellerquelle). <https://apicontext.com/resources/api-drift-white-paper/>; dazu J Simpson, *Most APIs Suffer From Specification Drift*, Nordic APIs, 16. September 2024. <https://nordicapis.com/most-apis-suffer-from-specification-drift/>
 13. Wen Siang Tan, Markus Wagner, Christoph Treude, *Detecting outdated code element references in software repository documentation*, Empirical Software Engineering 29(1), Art. 5, 2024. DOI: 10.1007/s10664-023-10397-6 (Preprint arXiv:2212.01479, 2. Dezember 2022)
 14. Alfredo Perez, *I Wrote 238 Specs and Never Read One Again*, alfredo-perez.dev, 31. August 2026. <https://www.alfredo-perez.dev/blog/2026-08-31-i-wrote-238-specs-and-never-read-one-again>

15. ISO/IEC/IEEE 29148:2018, *Systems and software engineering – Life cycle processes – Requirements engineering*, 2. Ausgabe, November 2018. DOI: 10.1109/IEEESTD.2018.8559686
16. IEEE Std 830-1998, *IEEE Recommended Practice for Software Requirements Specifications*, Oktober 1998. DOI: 10.1109/IEEESTD.1998.88286
17. ISO/IEC/IEEE DIS 29148, *Systems and software engineering – Life cycle processes – Requirements engineering*, Entwurf der 3. Ausgabe, 2026. <https://www.iso.org/standard/94091.html>
18. Mike Amundsen, *APIs with Swagger: An Interview with Reverb's Tony Tam*, InfoQ, 9. Juni 2015. <https://www.infoq.com/articles/swagger-interview-tony-tam/>
19. OpenAPI Initiative, *OpenAPI Specification* [3.0.0](#) (26. Juli 2017), [3.1.0](#) (15. Februar 2021), [3.2.0](#) (19. September 2025) und [3.2.1](#) (10. September 2026, mit Revisionstabelle); FAQ, <https://www.openapis.org/faq>
20. OpenAPI Initiative, *Best Practices*, learn.openapis.org, abgerufen am 25. Oktober 2026. <https://learn.openapis.org/best-practices.html>
21. OpenAPI Initiative, *Arazzo Specification v1.0.0*, 29. Mai 2024. <https://spec.openapis.org/arazzo/v1.0.0.html>
22. Protocol Buffers, *History und Language Guide (proto 3)*, protobuf.dev, abgerufen am 25. Oktober 2026. <https://protobuf.dev/history/>, <https://protobuf.dev/programming-guides/proto3/>; Kenton Varda, *Protocol Buffers: Google's Data Interchange Format*, Google Open Source Blog, 7. Juli 2008. <https://opensource.googleblog.com/2008/07/protocol-buffers-googles-data.html>
23. Mugur Marculescu, *Introducing gRPC, a new open source HTTP/2 RPC Framework*, Google Developers Blog, 26. Februar 2015. <https://developers.googleblog.com/2015/02/introducing-grpc-new-open-source-http2.html>; Varun Talwar, *gRPC Project is now 1.0 and ready for production deployments*, grpc.io, 23. August 2016. <https://grpc.io/blog/ga-announcement/>
24. Buf Technologies, *Detecting breaking changes*, buf.build, abgerufen am 25. Oktober 2026. <https://buf.build/docs/breaking/>
25. Kin Lane, *What Is An API First Strategy? Adding Some Dimensions To This New Question*, API Evangelist, 11. August 2014. <https://apievangelist.com/2014/08/11/what-is-an-api-first-strategy-adding-some-dimensions-to-this-new-question/>

26. Zalando SE, *Zalando RESTful API and Event Guidelines*, Regel #100 „MUST follow API first principle“, Stand 29. September 2026. <https://opensource.zalando.com/restful-api-guidelines/#100>
27. Postman, *2025 State of the API Report*, Oktober 2025. <https://www.postman.com/state-of-api/2025/>
28. Ian Robinson, *Consumer-Driven Contracts: A Service Evolution Pattern*, martinowler.com, 12. Juni 2006. <https://martinfowler.com/articles/consumerDrivenContracts.html>
29. Matthew Fellows, *Contract Testing vs. Schema Validation: Know the Difference*, Nordic APIs, 25. November 2025. <https://nordicapis.com/contract-testing-vs-schema-validation-know-the-difference/>
30. Hyrum Wright, *Hyrum's Law*, hyrumslaw.com, abgerufen am 25. Oktober 2026. <https://www.hyrumslaw.com/>; Titus Winters, Tom Manshreck, Hyrum Wright (Hrsg.), *Software Engineering at Google: Lessons Learned from Programming Over Time*, O'Reilly, 2020, Kap. 1.
31. Dan North, *Introducing BDD*, Better Software, März 2006. <https://dannorth.net/blog/introducing-bdd/>
32. Liz Keogh, *Deriving Gherkin from Real Conversations*, lizkeogh.com, 1. September 2014. <https://lizkeogh.com/2014/09/01/deriving-gherkin-from-real-conversations/>
33. Gojko Adzic, *Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing*, Neuri, London, 2009.
34. Gojko Adzic, *Specification by Example: How Successful Teams Deliver the Right Software*, Manning, 2011.
35. Gojko Adzic, *Specification by Example, 10 years later*, gojko.net, 17. März 2020. <https://gojko.net/2020/03/17/sbe-10-years.html>
36. Michael Nygard, *Documenting Architecture Decisions*, Relevance Blog, 15. November 2011. <https://www.cognitect.com/blog/2011/11/15/documenting-architecture-decisions>
37. Leslie Lamport, *The Temporal Logic of Actions*, ACM TOPLAS 16(3), Mai 1994, S. 872–923 (erste Fassung: SRC Research Report 57, April 1990). DOI: 10.1145/177492.177726
38. Leslie Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, Addison-Wesley, 2002.

- 39. Daniel Jackson, *Alloy: A Lightweight Object Modelling Notation*, ACM TOSEM 11(2), April 2002, S. 256–290. DOI: 10.1145/505145.505149
- 40. K. Rustan M. Leino, *Dafny: An Automatic Program Verifier for Functional Correctness*, LPAR-16, LNCS 6355, Springer, 2010, S. 348–370. DOI: 10.1007/978-3-642-17511-4_20
- 41. David Harel, *Statecharts: A Visual Formalism for Complex Systems*, Science of Computer Programming 8(3), Juni 1987, S. 231–274. DOI: 10.1016/0167-6423(87)90035-9
- 42. Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, Michael Deardeuff, *How Amazon Web Services Uses Formal Methods*, Communications of the ACM 58(4), April 2015, S. 66–73. DOI: 10.1145/2699417
- 43. James Bornholt et al., *Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3*, SOSPP '21, ACM, Oktober 2021, S. 836–850. DOI: 10.1145/3477132.3483540
- 44. Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2003.
- 45. Eric Evans, *Domain-Driven Design Reference: Definitions and Pattern Summaries*, Domain Language, 2015.
<https://www.domainlanguage.com/ddd/reference/>
- 46. Martin Fowler, *BoundedContext*, Bliki, martinfowler.com, 15. Januar 2014. <https://martinfowler.com/bliki/BoundedContext.html>
- 47. Kent Beck, *Test-Driven Development: By Example*, Addison-Wesley, 2002.
- 48. Yahya Rafique, Vojislav B. Mišić, *The Effects of Test-Driven Development on External Quality and Productivity: A Meta-Analysis*, IEEE Transactions on Software Engineering 39(6), Juni 2013, S. 835–856. DOI: 10.1109/TSE.2012.28
- 49. Kiro, *Introducing Workflows in Kiro Web*, kiro.dev Changelog, 30. September 2026. <https://kiro.dev/changelog/web/introducing-workflows/>
- 50. Fission AI, *Fission-AI/OpenSpec*, Repository, Release v1.14.1, Stand 14. Oktober 2026. <https://github.com/Fission-AI/OpenSpec>
- 51. BMad Code, LLC, *bmada-code-org/BMAD-METHOD*, Repository, Release v6.12.2, Stand 14. Oktober 2026. <https://github.com/bmada-code-org/BMAD-METHOD>

52. Guy Podjarny, *Announcing skills on Tessel: the package manager for agent skills*, tessl.io, 29. Januar 2026. <https://tessel.io/blog/skills-are-software-and-they-need-a-lifecycle-introducing-skills-on-tessel>
53. Paul Sawers, *With Cosmos, Augment Code wants to give AI coding teams shared memory and context*, tessl.io, 5. Mai 2026. <https://tessel.io/blog/with-cosmos-augment-code-wants-to-give-ai-coding-teams-a-shared-memory-and-context>; Jyoti Bansal, *Harness acquires Augment Code to advance the Autonomous SDLC*, harness.io, 8. Oktober 2026. <https://www.harness.io/blog/harness-acquires-augment-code-cosmos>
54. Agentic AI Foundation, *AGENTS.md*, agents.md, abgerufen am 25. Oktober 2026. <https://agents.md/>
55. Andrey Breslav, *What's next for CodeSpeak: Structured Approach to Specs*, codespeak.dev, 13. Juli 2026. <https://codespeak.dev/blog/spindle-20260713>
56. Thoughtworks, *Spec-driven development*, Technology Radar Vol. 33, Technique, Ring „Assess“, 5. November 2025. <https://www.thoughtworks.com/radar/techniques/spec-driven-development>
57. Thoughtworks, *Technology Radar Vol. 34*, April 2026. <https://www.thoughtworks.com/content/dam/thoughtworks/documents/radar/20260401-technology-radar-vol-34.pdf>
58. OpenAPI Tools, *OpenAPITools/openapi-generator*, Repository, abgerufen am 25. Oktober 2026. <https://github.com/OpenAPITools/openapi-generator>
59. Stoplight, *stoplightio/prism*, Repository, abgerufen am 25. Oktober 2026. <https://github.com/stoplightio/prism>
60. WireMock, *WireMock Cloud: OpenAPI-Import* und *wiremock/wiremock*, abgerufen am 25. Oktober 2026. <https://www.wiremock.io/comparison>
61. Pact Foundation, *How Pact works*, docs.pact.io, abgerufen am 25. Oktober 2026. https://docs.pact.io/getting_started/how_pact_works
62. Matt Fellows, *Pact is dead, long live Pact*, pactflow.io, aktualisiert am 8. März 2022. <https://pactflow.io/blog/bi-directional-contracts/>
63. Zac Hatfield-Dodds, Dmitry Dygalo, *Deriving Semantics-Aware Fuzzers from Web API Schemas*, ICSE '22 Companion (Poster), ACM, 2022, S. 345–346. DOI: 10.1145/3510454.3528637

64. oasdiff, *Breaking Changes*, `docs/BREAKING-CHANGES.md`, abgerufen am 25. Oktober 2026.
<https://github.com/oasdiff/oasdiff/blob/main/docs/BREAKING-CHANGES.md>
65. Brenn Hill, *Specification Artifacts in Open-Source Pull Request Workflows Do Not Reduce Defects: An Empirical Test of Spec-Driven Development Claims*, ESEM 2026, LIPIcs Bd. 394, Art. 84, Schloss Dagstuhl, 5. Oktober 2026. DOI: 10.4230/LIPIcs.ESEM.2026.84
66. Thibaud Gloaguen, Niels Mündler, Mark Niklas Müller, Veselin Raychev, Martin Vechev, *Evaluating AGENTS.md: Are Repository-Level Context Files Helpful for Coding Agents?*, ICLR 2026 Workshop „Agentic AI in the Wild“, arXiv:2602.11988.
<https://arxiv.org/abs/2602.11988>
67. Francesco Dente, Dario Satriani, Paolo Papotti, *Constraint Decay: The Fragility of LLM Agents in Backend Code Generation*, COLM 2026, arXiv:2605.06445. <https://arxiv.org/abs/2605.06445>
68. Gabriel Orlanski et al., *SlopCodeBench: Benchmarking How Coding Agents Degrade Over Long-Horizon Iterative Tasks*, NeurIPS 2026 (Poster), arXiv:2603.24755. <https://arxiv.org/abs/2603.24755>
69. Michele Tufano et al. (Google), *Grounding AI Agents in Contracts: An Empirical Evaluation of Spec-Driven Test Generation*, SpecOps 2026 (Workshop bei SPLASH 2026), arXiv:2608.17177.
<https://arxiv.org/abs/2608.17177>
70. Gabriella Gonzalez, *A sufficiently detailed spec is code*, Haskell for all, 17. März 2026. <https://haskellforall.com/2026/03/a-sufficiently-detailed-spec-is-code>
71. Vaggelis Atlidakis, Patrice Godefroid, Marina Polishchuk, *RESTler: Stateful REST API Fuzzing*, ICSE 2019, IEEE/ACM, S. 748–758. DOI: 10.1109/ICSE.2019.00083
72. OWASP API Security Project, *OWASP Top 10 API Security Risks – 2023*. <https://owasp.org/API-Security/editions/2023/en/0x11-t10/>
73. Alberto Martin-Lopez, Sergio Segura, Antonio Ruiz-Cortés, *A Catalogue of Inter-Parameter Dependencies in RESTful Web APIs*, ICSOC 2019, LNCS, Springer, S. 399–414. DOI: 10.1007/978-3-030-33702-5_31
74. Ohiremen Grace Oluwabunmi, Beom Heyn Kim, *The Hidden Risk of Auto-generated OpenAPI Specifications: An Analysis of WordPress*

- Plugins and Lightweight Specification Correction*, IFIP SEC 2026, IFIP AICT, Springer, S. 331–344. DOI: 10.1007/978-3-032-27993-4_23
75. AsyncAPI Initiative, *AsyncAPI Specification 2.0.0*, 11. September 2019. <https://github.com/asyncapi/spec/releases/tag/2.0.0>; Jonas Lagoni, *AsyncAPI 3.0.0 Release Notes*, [asyncapi.com](https://www.asyncapi.com), 5. Dezember 2023. <https://www.asyncapi.com/blog/release-notes-3.0.0>
 76. Confluent, *Schema Evolution and Compatibility for Schema Registry on Confluent Platform*, Confluent-Dokumentation, abgerufen am 25. Oktober 2026. <https://docs.confluent.io/platform/current/schema-registry/fundamentals/schema-evolution.html>
 77. Melvin E. Conway, *How Do Committees Invent?*, *Datamation* 14(4), April 1968, S. 28–31; dazu Conways Rückschau auf [melconway.com](https://www.melconway.com/Home/Committees_Paper.html). https://www.melconway.com/Home/Committees_Paper.html
 78. Frederick P. Brooks Jr., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley, 1975.
 79. Lyra J. Colfer, Carliss Y. Baldwin, *The mirroring hypothesis: theory, evidence, and exceptions*, *Industrial and Corporate Change* 25(5), 2016, S. 709–738. DOI: 10.1093/icc/dtw027
 80. Nachiappan Nagappan, Brendan Murphy, Victor R. Basili, *The Influence of Organizational Structure on Software Quality: An Empirical Case Study*, ICSE '08, ACM, Mai 2008, S. 521–530. DOI: 10.1145/1368088.1368160
 81. Matthew Skelton, Manuel Pais, *Team Topologies: Organizing Business and Technology Teams for Fast Flow*, IT Revolution Press, 2019.
 82. Nils Brede Moe, Darja Šmite, Maria Paasivaara, Casper Lassenius, *Finding the sweet spot for organizational control and team autonomy in large-scale agile software development*, *Empirical Software Engineering* 26(5), Art. 101, 2021. DOI: 10.1007/s10664-021-09967-3
 83. Deepa Iyer, Kalle Lyytinen, William Robinson, *The Uncertainty Challenge: To Centralize or Decentralize Requirements Engineering Decision-Making in Open Source Software Development?*, *ACM Transactions on Management Information Systems* 17(1), März 2026, S. 1–33. DOI: 10.1145/3766893
 84. Subhajit Datta, *Influence, Information and Team Outcomes in Large Scale Software Development*, APSEC 2019, IEEE, S. 402–409. DOI: 10.1109/APSEC48747.2019.00061
 85. Derek DeBellis et al., *Accelerate State of DevOps Report 2024*, DORA/Google Cloud, 23. Oktober 2024.

<https://dora.dev/research/2024/dora-report/>

86. CMMI Institute (ISACA), *CMMI Model V3.0*, 6. April 2023.
<https://cmmiinstitute.com/Resource-Files/Public/CMMI-Model-Materials/CMMI-Model-Release-Notes>
87. Mark C. Paulk, Bill Curtis, Mary Beth Chrissis et al., *Capability Maturity Model for Software*, CMU/SEI-91-TR-24, Software Engineering Institute, August 1991. <https://apps.dtic.mil/sti/pdfs/ADA240603.pdf>
88. Jens Pöppelbuß, Maximilian Röglinger, *What Makes a Useful Maturity Model? A Framework of General Design Principles for Maturity Models and Its Demonstration in Business Process Management*, ECIS 2011, Paper 28. <https://aisel.aisnet.org/ecis2011/28/>
89. Marilyn Strathern, 'Improving ratings': audit in the British University system, *European Review* 5(3), Juli 1997, S. 305–321. DOI: 10.1017/S1062798700002660
90. Charles A. E. Goodhart, *Problems of Monetary Management: The U.K. Experience*, in: *Papers in Monetary Economics*, Bd. I, Reserve Bank of Australia, Sydney, 1976 (Tagung im Juli 1975).
91. Erik Kamsties, Antje von Knethen, Jan Philipps, Bernhard Schätz, *An Empirical Investigation of the Defect Detection Capabilities of Requirements Specification Languages*, EMMSAD '01, 2001.
<https://www.broy.in.tum.de/publ/papers/emmsad01.pdf>
92. Joel Becker, Nate Rush, Elizabeth Barnes, David Rein, *Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity*, METR, 10. Juli 2025. <https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>
93. DORA/Google Cloud, *2025 DORA Report: State of AI-assisted Software Development*, 23. September 2025.
<https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report>
94. Taiichi Ohno, *Toyota Production System: Beyond Large-Scale Production*, Productivity Press, 1988 (japanisch 1978).
95. Lean Enterprise Institute, *Seven Wastes*, Lean Lexicon, lean.org, abgerufen am 25. Oktober 2026. <https://www.lean.org/lexicon-terms/seven-wastes/>
96. Todd Sedano, Paul Ralph, Cécile Péraire, *Software Development Waste*, ICSE 2017, IEEE/ACM, S. 130–140. DOI: 10.1109/ICSE.2017.20

97. Barry Boehm, Victor R. Basili, *Software Defect Reduction Top 10 List*, Computer 34(1), Januar 2001, S. 135–137. DOI: 10.1109/2.962984
98. Tom Preston-Werner, *Semantic Versioning 2.0.0*, semver.org, 18. Juni 2013. <https://semver.org/spec/v2.0.0.html>
99. Steven Raemaekers, Arie van Deursen, Joost Visser, *Semantic Versioning versus Breaking Changes: A Study of the Maven Repository*, SCAM 2014, IEEE, S. 215–224. DOI: 10.1109/SCAM.2014.30
100. Lina Ochoa, Thomas Degueule, Jean-Rémy Falleri, Jurgen Vinju, *Breaking bad? Semantic versioning and impact of breaking changes in Maven Central*, Empirical Software Engineering 27(3), Art. 61, 2022. DOI: 10.1007/s10664-021-10052-y
101. Jerin Yasmin, Yuan Tian, Jinqiu Yang, *A First Look at the Deprecation of RESTful APIs: An Empirical Study*, ICSME 2020, IEEE, S. 151–161. DOI: 10.1109/ICSME46990.2020.00024
102. Danilo Sato, *ParallelChange*, Bliki, martinowler.com, 13. Mai 2014. <https://martinfowler.com/bliki/ParallelChange.html>
103. David Brunt, *Seven or Eight Wastes in Lean Thinking?*, Lean Enterprise Academy, 8. Oktober 2021. <https://www.leanuk.org/lean-wastes/>
104. Mary Poppendieck, Tom Poppendieck, *Implementing Lean Software Development: From Concept to Cash*, Addison-Wesley, 2006, Kap. 4 „Waste“.
105. John D. C. Little, *A Proof for the Queuing Formula: $L = \lambda W$* , Operations Research 9(3), 1961, S. 383–387. DOI: 10.1287/opre.9.3.383
106. Beau Keyte, Drew Locher, *The Complete Lean Enterprise: Value Stream Mapping for Administrative and Office Processes*, Productivity Press, 2004.
107. Amazon Web Services, *Using development value stream mapping to identify constraints to DevOps outcomes*, AWS Prescriptive Guidance, 13. April 2023. <https://docs.aws.amazon.com/prescriptive-guidance/latest/strategy-devops-value-stream-mapping/create-value-stream-map.html>
108. Barry W. Boehm, *Verifying and Validating Software Requirements and Design Specifications*, IEEE Software 1(1), Januar 1984, S. 75–88. DOI: 10.1109/MS.1984.233702
109. Marc Brooker, *Spec Driven Development isn't Waterfall*, Marc's Blog, 9. April 2026. <https://brooker.co.za/blog/2026/04/09/waterfall-vs->

spec.html

110. J. L. Lions (Vorsitz), *ARIANE 5 – Flight 501 Failure – Report by the Inquiry Board*, ESA/CNES, 19. Juli 1996.
<https://esamultimedia.esa.int/docs/esa-x-1819eng.pdf>
111. Mars Climate Orbiter Mishap Investigation Board (Vorsitz Arthur G. Stephenson), *Phase I Report*, NASA, 10. November 1999.
https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
112. Daniel M. Berry, *Ambiguity in Natural Language Requirements Documents*, Foliensatz, University of Waterloo, 2007, Folie 32.
<https://cs.uwaterloo.ca/~dberry/ATRE/Slides/Ambiguity.pdf>
113. Fabian de Bruijn, Hans L. Dekkers, *Ambiguity in Natural Language Software Requirements: A Case Study*, REFSQ 2010, LNCS, Springer, 2010, S. 233–247. DOI: 10.1007/978-3-642-14192-8_21
114. Erik Kamsties, Daniel M. Berry, Barbara Paech, *Detecting Ambiguities in Requirements Documents Using Inspections*, WISE '01, Paris, Juli 2001.
https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/kamsties.berry.paech.p
115. Francis Chantree, Bashar Nuseibeh, Anne de Roeck, Alistair Willis, *Identifying Nocuous Ambiguities in Natural Language Requirements*, RE '06, IEEE, 2006, S. 59–68. DOI: 10.1109/RE.2006.31
116. Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, Sebastian Eder, *Rapid quality assurance with Requirements Smells*, Journal of Systems and Software 123, Januar 2017, S. 190–213. DOI: 10.1016/j.jss.2016.02.047
117. Per Lenberg, Robert Feldt, *Psychological Safety and Norm Clarity in Software Engineering Teams*, CHASE '18, ACM, 2018, S. 79–86. DOI: 10.1145/3195836.3195847
118. *Honorarordnung für Architekten und Ingenieure (HOAI)* vom 10. Juli 2013 in der ab 1. Januar 2021 geltenden Fassung, § 34.
https://www.gesetze-im-internet.de/hoai_2013/__34.html
119. DIN 276:2018-12, *Kosten im Bauwesen*, Beuth, Berlin, Dezember 2018.
120. ISO 19650-1:2018, *Organization and digitization of information about buildings and civil engineering works, including building information modelling (BIM) – Information management using building information modelling – Part 1: Concepts and principles*, Dezember 2018.
<https://www.iso.org/standard/68078.html>

- 121. Bundesministerium des Innern, für Bau und Heimat; Bundesministerium der Verteidigung, *Masterplan BIM für Bundesbauten – Erläuterungsbericht*, November 2021. https://fib-bund.de/Inhalt/Themen/BIM_fuer_Bundesbauten/2021-11-01_BIM_Masterplan.pdf
- 122. §§ 650b, 650c BGB, eingeführt durch das *Gesetz zur Reform des Bauvertragsrechts ...* vom 28. April 2017, BGBl. I S. 969, in Kraft seit 1. Januar 2018. https://www.gesetze-im-internet.de/bgb/_650b.html
- 123. Bent Flyvbjerg, *What You Should Know about Megaprojects and Why: An Overview*, *Project Management Journal* 45(2), 2014, S. 6–19. DOI: 10.1002/pmj.21409
- 124. Michael P. Gallaher, Alan C. O'Connor, John L. Dettbarn, Jr., Linda T. Gilday, *Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry*, NIST GCR 04-867, National Institute of Standards and Technology, August 2004. DOI: 10.6028/NIST.GCR.04-867
- 125. Jack W. Reeves, *What Is Software Design?*, *C++ Journal* 2(2), 1992. https://www.developerdotstar.com/mag/articles/reeves_design.html
- 126. Federal Aviation Administration, *Advisory Circular AC 20-115D: Airborne Software Development Assurance Using EUROCAE ED-12() and RTCA DO-178()*, 21. Juli 2017. https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_20-115D.pdf
- 127. RTCA, *DO-178C: Software Considerations in Airborne Systems and Equipment Certification*, 13. Dezember 2011.
- 128. RTCA, *DO-333: Formal Methods Supplement to DO-178C and DO-278A*, 13. Dezember 2011.
- 129. IEC 62304:2006, *Medical device software – Software life cycle processes*, Mai 2006 (Amendment 1:2015).
- 130. U.S. Food and Drug Administration, *Content of Premarket Submissions for Device Software Functions*, Guidance, 14. Juni 2023. <https://www.fda.gov/media/153781/download>
- 131. Nancy G. Leveson, Clark S. Turner, *An Investigation of the Therac-25 Accidents*, *IEEE Computer* 26(7), Juli 1993, S. 18–41. DOI: 10.1109/MC.1993.274940
- 132. Joint Authorities Technical Review (Vorsitz Christopher A. Hart), *Boeing 737 MAX Flight Control System: Observations, Findings, and Recommendations*, 11. Oktober 2019.

https://www.faa.gov/sites/faa.gov/files/2022-08/Final_JATR_Submittal_to_FAA_Oct_2019.pdf

133. U.S. Department of Justice, *Boeing Charged with 737 Max Fraud Conspiracy and Agrees to Pay over \$2.5 Billion*, Pressemitteilung, 7. Januar 2021. <https://www.justice.gov/archives/opa/pr/boeing-charged-737-max-fraud-conspiracy-and-agrees-pay-over-25-billion>
134. Alex B. Haynes et al., *A Surgical Safety Checklist to Reduce Morbidity and Mortality in a Global Population*, *New England Journal of Medicine* 360(5), 2009, S. 491–499. DOI: 10.1056/NEJMsa0810119
135. David R. Urbach, Anand Govindarajan, Refik Saskin, Andrew S. Wilton, Nancy N. Baxter, *Introduction of Surgical Safety Checklists in Ontario, Canada*, *New England Journal of Medicine* 370(11), 2014, S. 1029–1038. DOI: 10.1056/NEJMsa1308261
136. Lucian L. Leape, *The Checklist Conundrum*, *New England Journal of Medicine* 370(11), 2014, S. 1063–1064. DOI: 10.1056/NEJMe1315851
137. § 133 BGB, *Auslegung einer Willenserklärung*. https://www.gesetze-im-internet.de/bgb/__133.html
138. *Übereinkommen der Vereinten Nationen über Verträge über den internationalen Warenkauf* (CISG), Wien, 11. April 1980, Art. 8.
139. BGH, Urteil vom 8. September 2021, Az. VIII ZR 97/19, Rn. 55 f. <https://www.bundesgerichtshof.de/SharedDocs/Entscheidungen/DE/Zivilsenat19.pdf>
140. Oliver Hart, John Moore, *Property Rights and the Nature of the Firm*, *Journal of Political Economy* 98(6), 1990, S. 1119–1158. DOI: 10.1086/261729
141. § 305c BGB, *Überraschende und mehrdeutige Klauseln*. https://www.gesetze-im-internet.de/bgb/__305c.html
142. *Richtlinie 93/13/EWG des Rates vom 5. April 1993 über mißbräuchliche Klauseln in Verbraucherverträgen*, ABl. L 95 vom 21. April 1993, S. 29, Anhang Nr. 1 Buchst. m. <https://eur-lex.europa.eu/eli/dir/1993/13/oj>
143. Bundesministerium der Justiz, *Handbuch der Rechtsförmlichkeit*, 4. Auflage, 2024. <https://hdr.bmj.de>
144. Ryan Singer, *Shape Up: Stop Running in Circles and Ship Work that Matters*, Basecamp, 2019. <https://basecamp.com/shapeup>
145. Alberto Brandolini, *Introducing Event Storming*, Ziobrando's Lair, 18. November 2013.

<https://ziobrando.blogspot.com/2013/11/introducing-event-storming.html>

- 146. Stefan Hofer, Henning Schwentner, *Domain Storytelling: A Collaborative, Visual, and Agile Way to Build Domain-Driven Software*, Addison-Wesley, 2021.
- 147. PactFlow, *What is the difference between consumer driven contract testing and Bi-Directional Contract Testing?*, pactflow.io, undatiert (Herstellerquelle). <https://pactflow.io/difference-between-consumer-driven-contract-testing-and-bi-directional-contract-testing/>
- 148. Justin McCarthy, *Software Factories And The Agentic Moment*, StrongDM, 6. Februar 2026. <https://factory.strongdm.ai/>; Simon Willison, *How StrongDM's AI team build serious software without even looking at the code*, simonwillison.net, 7. Februar 2026. <https://simonwillison.net/2026/Feb/7/software-factory/>
- 149. Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy (Hrsg.), *Site Reliability Engineering: How Google Runs Production Systems*, O'Reilly, 2016, Kap. 4 „Service Level Objectives“. <https://sre.google/sre-book/service-level-objectives/>
- 150. Andrew Jones, *Driving Data Quality with Data Contracts*, Packt, Birmingham, Juni 2023.
- 151. Bitol (Linux Foundation), *Open Data Contract Standard (ODCS)*, Version 3.2.0, Stand 25. Oktober 2026. <https://github.com/bitol-io/open-data-contract-standard>
- 152. Ed Bukoski, Brian Moyles, Mike McGarr, *How We Build Code at Netflix*, Netflix Technology Blog, 9. März 2016. <https://netflixtechblog.com/how-we-build-code-at-netflix-c5d9bd727f15>
- 153. Gary Niemen, *How We Use Golden Paths to Solve Fragmentation in Our Software Ecosystem*, Spotify Engineering, 17. August 2020. <https://engineering.atspotify.com/2020/8/how-we-use-golden-paths-to-solve-fragmentation-in-our-software-ecosystem>
- 154. Darren Evans, *Beyond guardrails: A taxonomy of platform engineering control mechanisms*, Google Cloud Blog, 16. August 2025. <https://cloud.google.com/blog/products/application-modernization/platform-engineering-control-mechanisms>
- 155. Steve Fenton, *Avoiding golden cages in Platform Engineering*, stevefenton.co.uk, 14. November 2025. <https://stevefenton.co.uk/blog/2025/11/avoiding-golden-cages/>

- 156. Mateen Ali Anjum, *Platform engineering and internal developer portals: a multivocal literature review*, *Frontiers in Computer Science* 8:1814498, 4. Mai 2026. DOI: 10.3389/fcomp.2026.1814498
- 157. CNCF TAG App Delivery, Platforms Working Group, *Platform Engineering Maturity Model*, Whitepaper, 1. November 2023.
<https://tag-app-delivery.cncf.io/whitepapers/platform-eng-maturity-model/>
- 158. Gerwin Klein et al., *seL4: Formal Verification of an OS Kernel*, SOSP '09, ACM, Oktober 2009, S. 207–220. DOI: 10.1145/1629575.1629596
- 159. Gerwin Klein, June Andronick, Kevin Elphinstone, Toby Murray, Thomas Sewell, Rafal Kolanski, Gernot Heiser, *Comprehensive Formal Verification of an OS Microkernel*, *ACM Transactions on Computer Systems* 32(1), Art. 2, Februar 2014. DOI: 10.1145/2560537
- 160. Addy Osmani, *Software Factories, Light and Dark*, Elevate, 22. Juli 2026. <https://addyo.substack.com/p/software-factories-light-and-dark>
- 161. *SpecOps 2026: 1st International Workshop on Specification-Driven Development Life Cycle*, SPLASH/ISSTA 2026, Oakland, 6. Oktober 2026. <https://conf.researchr.org/home/splash-issta-2026/specops-2026>
- 162. Shuvendu K. Lahiri, *Intent Formalization: Assessing the Quality of AI-Generated Formal Program Specifications*, Keynote, SpecOps 2026, Oakland, 6. Oktober 2026. <https://conf.researchr.org/home/splash-issta-2026/specops-2026>
- 163. Giovanni Rosa, David Moreno-Lumbreras, Gregorio Robles, Jesús M. González-Barahona, *Understanding Specification-Driven Code Generation with LLMs: An Empirical Study Design*, Registered Report (Stage 1), SANER 2026, arXiv:2601.03878.
<https://arxiv.org/abs/2601.03878>

■ EOF