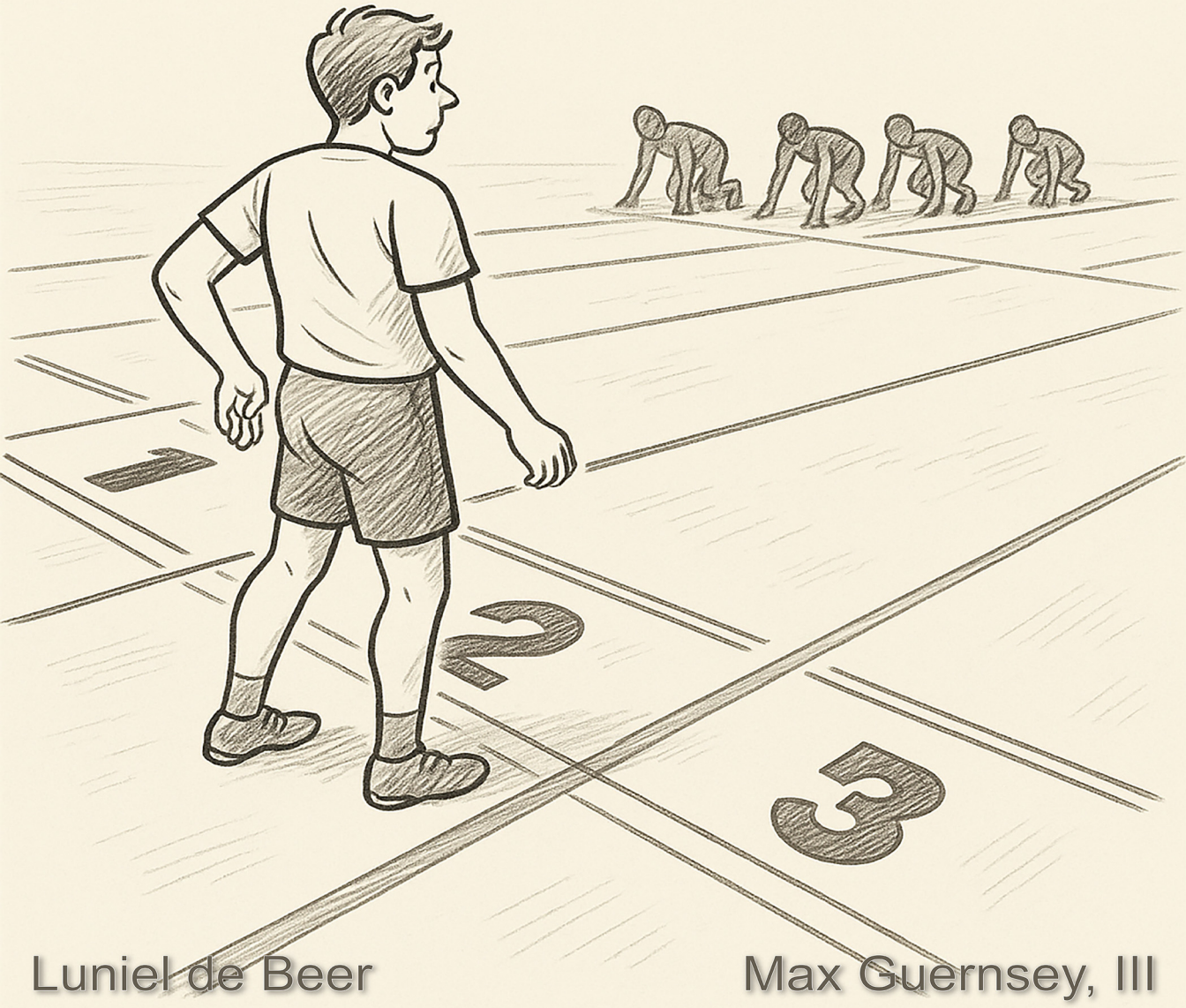
READY

WHY MOST SOFTWARE PROJECTS FAIL AND HOW TO FIX IT



Luniel de Beer

Max Guernsey, III

Deutsche Ausgabe

Twittern Sie dieses Buch!

Bitte unterstützen Sie Luniel de Beer und Max Guernsey, III, indem Sie auf Twitter über dieses Buch sprechen!

Der vorgeschlagene Tweet für dieses Buch lautet:

Ich habe gerade Ready gekauft — ein Buch für Führungskräfte in der Softwareentwicklung und Teams, die Nacharbeit, Überhang und Fehlausrichtung in der Softwarebereitstellung eliminieren möchten. #CodeReady

Der vorgeschlagene Hashtag für dieses Buch ist #CodeReady.

Finden Sie heraus, was andere über das Buch sagen, indem Sie auf diesen Link klicken, um nach diesem Hashtag auf Twitter zu suchen:

#CodeReady

Ready (Deutsche Ausgabe)

Warum die meisten Softwareprojekte scheitern und was man dagegen tun kann

Luniel de Beer und Max Guernsey, III

Dieses Buch ist erhältlich unter https://leanpub.com/ready-de

Diese Version wurde am 2025-10-21 veröffentlicht

Dies ist ein Leanpub-Buch. Leanpub unterstützt Autoren und Verleger mit dem Lean Publishing-Prozess. Lean Publishing ist die Veröffentlichung eines sich in Entwicklung befindenden E-Books unter Verwendung schlanker Werkzeuge und vieler Iterationen, um Leserfeedback zu erhalten, den Kurs anzupassen, bis Sie das richtige Buch haben, und dann Zugkraft aufzubauen.

Zum Gedenken an Johann van Aardt, der meine Leidenschaft erkannte, mich in die echte Programmierung einführte und mir half, den Weg in eine neue Heimat zu finden. Und an meine Eltern, deren unerschütterliche Unterstützung—von meinem ersten Kunden bis zu meinem ersten Zuhause in den USA—all dies ermöglicht hat.

—Luniel

Für meine Familie, mit denen die Sonne auf- und untergeht.

—Max

Inhaltsverzeichnis

Über dieses Buch

Ready ist ein Buch für alle, die in der Softwareentwicklung tätig sind und die **Minderleistung**, **chronische Nacharbeit** und **unklare Anforderungen** satt haben.

Möglicherweise haben Sie bereits in die Verbesserung der Team-Ausführungskompetenzen investiert, Ihr Prozess-Framework optimiert oder den Code überarbeitet und benötigen dennoch weitere Verbesserungen.

Der Grund dafür ist, dass die hauptsächliche Einschränkung für die meisten Softwareentwicklungsteams nicht die Teamfähigkeiten sind, sondern die Anforderungsreife. **Selbst erfahrene Teams mit den richtigen Fähigkeiten kämpfen noch**, wenn sie mit unreifen Anforderungen arbeiten.

Ready stellt RMF (Requirements-Reifungsprozess) vor, einen praktischen und tiefgreifend strukturierten Ansatz zur Abstimmung von Produkt und Engineering, ohne dabei Ihren bestehenden Prozess zu ersetzen.

Egal ob Sie Scrum, Kanban oder etwas Individuelles verwenden, RMF hilft Ihnen dabei, den **Umfang zu stabilisieren**, **Übertrag zu eliminieren** und **das zu liefern**, **was wirklich wichtig ist**.

Wenn sich Ihre Teams am Rande des „fast fertig" festgefahren fühlen, wird dieses Buch Ihnen zeigen, wie Sie **den Kreislauf durchbrechen** und Ihr(e) Team(s) **dauerhaft** voranbringen können.

Für wen ist dieses Buch gedacht

Dieses Buch ist buchstäblich für jeden gedacht, der in der Softwareentwicklung tätig ist. Von Entwicklern bis hin zu Produktmanagern und von einzelnen Mitarbeitern bis hin zu Führungskräften.

Dieses Buch ist für Sie, wenn Sie in der Softwareentwicklung tätig sind und festgestellt haben, dass ein Team, mit dem oder in dem Sie arbeiten, eines oder mehrere der folgenden Probleme hat:

- Arbeit wird häufig von einer Iteration in die nächste übertragen
- Implementierungsteams haben das Gefühl, bewegliche Ziele treffen zu müssen
- Arbeit bleibt zu lange offen
- Arbeit wird als abgeschlossen markiert, ist aber nicht wirklich fertig
- Die geleistete Arbeit entspricht nicht den Erwartungen
- Die Arbeit erzeugt regelmäßig eine große Anzahl von Fehlern
- Große Mengen an Arbeit müssen regelmäßig neu gemacht werden

Wenn Ihnen eines dieser Probleme bekannt vorkommt, kann *Ready* helfen.

Wie Sie dieses Buch verwenden

Dieses Buch wurde praxisorientiert konzipiert. Es ist keine theoretische Abhandlung oder eine Strategiepräsentation – es ist ein praktisches Handbuch zur Einführung von RMF (dem Requirements-Reifungsprozess), basierend auf echter Kundenarbeit und unter realem Lieferdruck erprobt.

Die Kapitel sind der Reihe nach geschrieben, aber RMF selbst ist modular. Es besteht aus drei grundlegenden Praktiken:

- RMF 1: Zusammenarbeit für gemeinsames Verständnis
- RMF 2: Steuerung des Arbeitsabschlusses durch maßgeschneiderte Definitions of Done
- RMF 3: Steuerung der Implementierung durch maßgeschneiderte Definitions of Ready

Jeder Teil oder „Habit“, wie wir sie nennen, steht für sich, aber sie bauen aufeinander auf. Das Buch ist so konzipiert, dass es Ihnen hilft, sie nacheinander und in der richtigen Reihenfolge anzugehen. Diese Struktur spiegelt wider, wie wir Teams empfehlen, RMF in der Praxis einzuführen – wobei jeder Habit erst dann eingeführt wird, wenn der vorherige funktioniert.

Dies vermeidet eine Überforderung der Teams und gibt jeder Veränderung die beste Chance, sich zu etablieren. Mehr darüber, wie Sie das umsetzen können, erfahren Sie ab Kapitel 9.

Wenn Sie Hilfe suchen – sei es Beratung, Coaching oder jemanden, der mit Ihrem Führungsteam spricht – können Sie sich gerne direkt an uns wenden.

Und wenn Sie formelle Unterstützung bei der Implementierung von RMF suchen, bietet Producore eine vollständige Reihe von Programmen an, die die Einführung Schritt für Schritt begleiten. Mehr dazu erfahren Sie unter https://ready-book.link/rmf.

Über die Autoren

Luniel de Beer ist der Schöpfer des Requirements-Reifungsprozesses (RMF), eines praktischen Systems zur Behebung der Lücken zwischen Produktabsicht und technischer Umsetzung. Er verfügt über mehr als 15 Jahre Erfahrung in der Leitung von Agilen Transformationen, der Überbrückung von Produkt und Engineering sowie der Unterstützung von Teams bei der klaren und selbstbewussten Lieferung.

Luniel entwickelte auch Producores Capability-Management-System, einen nachvollziehbaren und skalierbaren Ansatz zur Modellierung von Produktfähigkeiten. Er konzipierte PKB-Driven Development (PKBDD), ein versionskontrolliertes System zur Verwaltung persistenter Produktanforderungen. Diese Werkzeuge sind Teil eines größeren bei Producore entwickelten Delivery-Frameworks.

Max Guernsey, III ist Softwarearchitekt, Ausbilder und Mitbegründer von Producore, einer Beratungsfirma, die sich der Behebung von Lieferausfällen durch strukturelle und technische Sorgfalt widmet. Mit über zwei Jahrzehnten Erfahrung in objektorientiertem Design, Refactoring, testgetriebener Entwicklung und Entwurfsmustern hat er sowohl unternehmenskritische Systeme geliefert als auch Entwicklungsteams im großen Maßstab gecoacht. Seine Arbeit verbindet tiefgreifende technische Praktiken mit Verhaltens- und Prozessumstellung, um Organisationen dabei zu helfen, nachhaltige Lieferexzellenz zu erreichen.

Max trug maßgeblich zu PKBDD bei und führte durch seine fundierte Expertise in der Verhaltensspezifikation die Entwicklung des Producore-Ansatzes für Verhaltensgetriebene Entwicklung (BDD) an.

Gemeinsam integriert ihre Arbeit Klarheit, Rückverfolgbarkeit und Gating in ein zusammenhängendes System für die Softwarebereitstellung, das von der Teampraxis bis zur organisatorischen Fähigkeit skaliert.

Vorwort

Hinweis an technische Führungskräfte

Wenn Sie eine Führungskraft in einer technischen Organisation sind, mangelt es Ihnen wahrscheinlich nicht an Einsatz, Disziplin oder klugen Köpfen. Und dennoch geraten Projekte ins Stocken. Ziele werden verfehlt. Erwartungen werden nicht erfüllt. Nicht weil Ihre Teams faul sind – sondern weil etwas Grundlegendes in der Art und Weise, wie Arbeit definiert, gestaltet und umgesetzt wird, nicht funktioniert.

Dieses Buch ist kein Führungsratgeber. Es ist ein Werkzeug zur strukturellen Diagnose. Es zeigt, was tatsächlich in Ihren Teams geschieht – warum aus „fast fertig" immer wieder „nicht fertig" wird und warum lokale Fortschritte so selten zu strategischen Ergebnissen führen.

Sie werden sich selbst vielleicht nicht in diesen Seiten wiederfinden. Aber wenn Ihre Teams nicht das liefern können, was Sie brauchen, werden Sie sie darin erkennen. Und wenn Sie das tun, werden Sie endlich die Sprache – und das System – haben, um es zu beheben.

Von Luniel

Zuallererst wäre dieses Buch ohne Max nicht möglich gewesen, dessen Fähigkeit, durch den Nebel und die Spreu zu sehen und eine Idee auf ihr Wesentliches zu reduzieren, mich absolut übersteigt.

Wie sind wir hierher gekommen?

Wenn ich zurückblicke, denke ich, es liegt daran, dass ich schon immer verstehen wollte, wie die Dinge wirklich funktionieren. Ob es um Religion, Ernährung oder Softwareentwicklung ging, ich stieß immer auf dasselbe Problem: oberflächliche Antworten, die unter Druck nicht standhielten. Also grub ich weiter – fragte nicht nur danach, was wir tun, sondern warum, und was fehlt, wenn es nicht funktioniert.

Eine der ersten Risse im System zeigte sich in einer Rolle, in der ich drei Hüte trug: Scrum Master, Product Owner und Entwicklungsleiter (!!) für ein Team, das Datendienste in einem

bekannten Technologieunternehmen bereitstellte. Wir machten alles nach Scrum-Vorgaben – kurze Sprints, Stories im Backlog, Planung in einem halben Tag – aber jedes Mal, wenn wir einen neuen Sprint begannen, stießen wir auf Reibung. Das Team verstand das Problem nicht vollständig, wir mussten Anforderungen mitten im Sprint überarbeiten, vermeidbare Abhängigkeiten tauchten auf und verzögerten uns, und wichtige Schritte wurden übersehen.

Also begann ich, etwas anders zu machen. Ich holte das Team und die Stakeholder für jede Story in einen Raum, ging das Problem im Detail durch, entwickelte gemeinsam die Lösung und schrieb erst dann die Story. Die Sprint-Planung schrumpfte auf eine Stunde, und unser Liefererfolg stieg sprunghaft an.

Langsam wurde mir klar, dass Erfolg nicht daher kommt, härter innerhalb des Sprints zu arbeiten. Er kommt von der Struktur, die man einrichtet, bevor er beginnt.

Später, nachdem ich Jeff Sutherland über „Definition of Ready“ sprechen hörte, wusste ich, dass dort etwas Wertvolles steckte – aber es war nicht genug. Meine Erfahrung mit Anforderungen, UX, UI, Forschung und später mit BDD zeigte mir, dass verschiedene Arbeitseinheiten verschiedene Arten von Bereitschaft erfordern. Einige benötigen Verhaltensspezifikationen. Einige benötigen Systemzugriff. Einige benötigen eine vollständige Fähigkeitsanalyse.

Und alle benötigen ein gemeinsames Verständnis, das tatsächlich bestätigt – nicht nur angenommen – wird.

Als ich mit mehr Teams arbeitete, sah ich überall das gleiche Muster: fehlende Schritte, unerfüllte Abhängigkeiten, Teams, die ihr Bestes gaben, aber ständig damit beschäftigt waren, Probleme zu beheben, die hätten verhindert werden können. Selbst großartige Teams kämpften – nicht weil sie schwach waren, sondern weil ihnen eine Struktur fehlte, die Bereitschaft explizit machte.

Das Ergebnis all dieses Lernens, dieser Iteration und Frustration ist ein strukturiertes System für das Management von Bereitschaft.

Darum geht es in diesem Buch.

Ich hoffe, es gibt Ihnen Klarheit darüber, wo die wirklichen Probleme liegen und wie man sie behebt. Ich hoffe, es gibt Ihnen die Sprache, um Praktiken zu verteidigen, die vielleicht „zusätzlich“ erscheinen, aber tatsächlich wesentlich sind. Und vor allem hoffe ich, dass es Teams hilft, mit weniger Stress, weniger Überraschungen und viel besseren Ergebnissen zu liefern.

Wenn wir das richtig machen, werden wir der Branche Milliarden von Dollar ersparen.

Aber noch wichtiger ist, dass wir den Menschen ihre geistige Gesundheit zurückgeben.

Von Max

Ich arbeite seit Jahrzehnten aus verschiedenen Blickwinkeln an diesem Problem, aber meine Fortschritte waren gehemmt, bis ich Luniel traf.

Das liegt daran, dass ich vor unserer Begegnung das Problem grundsätzlich als ein technisches betrachtete. Ich konzentrierte mich darauf, Teams bei der Einführung von Dingen wie Test-Driven Development (TDD), Refactoring, fortgeschrittenem Softwaredesign und später Acceptance-Test-Driven Development (ATDD) oder Behavior-Driven Development (BDD) zu unterstützen.

In den meisten dieser Fälle wurde das in diesem Buch behandelte Problem als ein Implementierungsdetail bei der Etablierung dieser technischen Praktiken behandelt.

Das soll nicht heißen, dass ich die technischen Praktiken nicht mehr schätze. Ich halte sie nach wie vor für sehr wichtig, aber sie adressieren nicht direkt das Problem der Bereitschaft in der Softwareentwicklung. Stattdessen bringen sie dieses Problem an die Oberfläche, und dann fügen die Menschen ihrem Prozess einen Patch hinzu, um es „gerade genug“ zu adressieren, um die technischen Praktiken zu unterstützen, die sie einführen wollen.

Ich möchte auch die Frage ansprechen, wem dieses Buch helfen kann. Die kurze Antwort lautet „wahrscheinlich fast jedem in der Softwareentwicklung“, aber die wirkliche Antwort enthält Nuancen, die helfen, es auf verschiedene Umgebungen zu übertragen, ohne die grundlegende Bedeutung zu ändern.

Es gibt Teams, die die in diesem Buch vorgestellte Lösung benötigen. Eine bereinigte Version eines solchen Teams werden Sie in Kapitel 1 kennenlernen.

Es gibt auch Teams, die ein System wie das von uns vorgeschlagene nicht unbedingt **brauchen**, aber dennoch davon profitieren könnten.

Das beste Team, mit dem ich je gearbeitet habe – mit Leichtigkeit eine volle Standardabweichung über dem **nächstbesten Team**, wenn nicht sogar zwei – befand sich im Hinterland von Central Oregon. Sie waren so hochleistungsfähig, dass sie das Fehlen eines solchen Systems durch schiere Kompetenz ausgleichen konnten. Dennoch sagte mir mein damaliger Manager, Tom Barreras, einmal etwas in der Art: „Mir ist aufgefallen, dass unsere Stories besser laufen, wenn wir uns im Voraus etwas Zeit nehmen, um über die Tests zu sprechen.“

Auch dies betrachtete ich damals durch die Brille der Testentwicklung und der technischen Ausführung, aber heute weiß ich, dass dies ein weiterer Hinweis darauf war, dass *Bereitschaft* ein Faktor war, der das Team beeinflusste… dieses besondere Team war einfach so fähig

und reaktionsschnell, dass sie durch das Reagieren auf Hindernisse erfolgreich sein konnten, anstatt sie von vornherein zu verhindern.

Selbst wenn Sie zu der Sorte Mensch gehören, die sich nicht unbedingt um Bereitschaft sorgen *muss*, weil Sie dies ausgleichen können, oder wenn Sie mit einem Team arbeiten, das ähnlich gestrickt ist, können Sie trotzdem von den Inhalten dieses Buches profitieren.

Teil I: Etwas fehlt

*Wenn es nicht hilft, die gleichen Dinge besser zu machen, achte darauf, was **nicht getan wird**.*

Kapitel 1: Das verborgene Problem

Dies ist eine wahre[1] Geschichte über eine Bank. Wir nennen sie einfach „Die Bank“. Es handelt sich um eine Art genossenschaftliches Kreditinstitut, das Teil der nationalen Finanzinfrastruktur der Vereinigten Staaten ist.

Wir (Luniel und Max) wurden in Die Bank geholt, weil sie Schwierigkeiten hatte, ein Softwareprojekt fertigzustellen. Es war eines der dysfunktionalsten Umfelder, die wir je gesehen hatten, und deshalb haben wir dies als einführende Fallstudie gewählt: Wenn bedeutsame Veränderung in Die Bank möglich war, ist sie überall möglich.

Eine kurze Anmerkung zu Projekten

Wenn wir in diesem Buch den Begriff „Projekt“ verwenden, meinen wir ihn im Kontext des Projektmanagements. Während es verschiedene Auffassungen darüber gibt, was das Wort bedeutet, verwenden wir die Definition des Project Management Institute:

> „Ein Projekt ist ein **zeitlich begrenztes** Vorhaben zur Schaffung eines einzigartigen Produkts, einer Dienstleistung oder eines Ergebnisses.“

Das bedeutet, ein Projekt hat einen definierten Anfang und ein definiertes Ende. Wenn ein Projekt abgeschlossen wird, werden Projektwissen und -artefakte archiviert, Teammitglieder werden freigestellt und Verträge werden finalisiert.

In diesem Buch geht es bei einem Projekt grundsätzlich um die Ausführung. Die meisten Projekte, wie vom PMI definiert, beginnen mit der Machbarkeitsanalyse oder dem Design. Vision und Strategie wurden bereits entwickelt, bevor ein Projekt initiiert wird.

Ein Projekt entsteht aus dieser Vision und Strategie und ist erfolgreich oder scheitert daran, ob die anvisierten Ziele erreicht werden – *nicht* daran, ob diese Ziele die richtigen waren.

Möglicherweise verwenden Sie das Wort „Projekt“ anders, und das ist in Ordnung. Sie sollten nur wissen, dass wir uns dabei auf die oben genannte Definition und den Kontext beziehen.

[1] Wir haben identifizierende Details geändert, um die Privatsphäre der beteiligten Personen und Institutionen zu schützen.

> Dies soll nicht bedeuten, dass wir die Verwendung von Projektmanagement für Softwareentwicklung befürworten. Ganz im Gegenteil. Aber wir erkennen an, dass es dennoch verwendet wird. Wir gehen dieses Problem später an, in Kapitel 6.

1.1: Die klassische Neuentwicklung

Die Bank entwickelte ihr Darlehensrückzahlungsportal aus mehreren Gründen neu.

Das alte System, eine vollständige C#/.NET-Lösung, war fehlerhaft. Zusätzlich zur sinkenden Kundenzufriedenheit erzeugte es auch einen nicht enden wollenden Strom sehr teurer Support-Vorfälle, bei denen jemand manuell in die Datenbank eingreifen musste, um einen vom System verursachten Fehler zu korrigieren.

Das alte System war auch aus Wartbarkeitssicht heruntergekommen. Es war für die Entwickler fast unmöglich, bedeutsame Änderungen vorzunehmen, und selbst wenn sie es konnten, war es ein äußerst riskantes Unterfangen.

Die Neuentwicklung sollte das ändern.

Das neue System sollte zwar weiterhin ein C#/.NET-Backend haben, aber dieses sollte vollständig durch Tests abgedeckt sein. Das Frontend sollte in OutSystems implementiert werden, einer beliebten Low-Code- bzw. No-Code-Lösung, die es einer Organisation ermöglicht, eine Anwendung an einer Stelle zu definieren und automatisch eine Web-App, eine Android-App und eine iOS-App zu generieren, wann immer sie ihre Änderungen veröffentlichen möchte.

Die Hoffnung war, dass das neue System fehlerfrei sein würde, wodurch sowohl die Kundenzufriedenheit verbessert als auch die Supportkosten erheblich reduziert würden.

Sie hofften auch, dass die Neuentwicklung die Entwickler entfesseln würde – wobei die Kombination aus einem disziplinierteren Ansatz für das Backend und dem Low-Code-Ansatz für das Frontend die Kosten und Risiken neuer Funktionen deutlich reduzieren sollte.

Ein angenehmer Nebeneffekt des Wechsels zu OutSystems war, dass sie eine saubere, moderne mobile App für beide wichtigen Plattformen erhalten würden.

Das war der Traum, als sie drei Jahre vor Beginn dieser Geschichte angefangen hatten. Die Realität war, dass die Teams bis jetzt **nichts** ausgeliefert hatten.

1.2: Perspektiven auf das Problem

Als wir mit dem Führungsteam sprachen, hörten wir sehr natürliche Frustration darüber, dass sie so große Investitionen getätigt hatten, ohne jegliche strategische Bewegung zu sehen.

Aus ihrer Sicht hatten sie alles versucht. Sie hatten Personal ausgetauscht, Personal aufgestockt, das Budget geändert, den Druck erhöht und eine Parade von Beratern eingebracht (zu denen wir, wie stark angedeutet wurde, das Schlusslicht bildeten). Nichts schien es besser zu machen – zumindest nicht in einer Weise, die sie messen konnten, denn alles, was sie sahen, war, dass die „Nadel" in einem Quartal bei null stand und im nächsten Quartal immer noch bei null.

Sie wollten keinen weiteren „unsichtbaren Fortschritt" mehr. Sie wollten *Ergebnisse*.

Als wir mit dem Management innerhalb der Produktorganisation sprachen, erhielten wir eine etwas andere (aber immer noch ähnliche) Geschichte, da sie direkter mit der Entwicklung zusammenarbeiteten.

Nicht dass die Teams gar nichts getan hätten, zumindest nicht aus ihrer Sicht. Es war vielmehr so, dass die Teams nie das taten, worum man sie gebeten hatte. Es war praktisch garantiert: Egal wie einfach die Anforderung und wie klar sie formuliert war, am Ende bekam man bei der Bewertung der Teamarbeit etwas völlig anderes.

Es war so weit gekommen, dass der gängige Witz lautete: „Wir müssen herausfinden, wie wir nach dem fragen können, was wir nicht wollen, damit wir wenigstens eine Chance haben, das zu bekommen, was wir tatsächlich wollen."

Die Führungskräfte der Entwicklungsabteilung sahen die Dinge ganz anders.

Für sie lag das Problem darin, dass Product keine umsetzbaren Anforderungen lieferte und dass Product nicht *genügend* Anforderungen lieferte. Wenn Product sich einfach „anpassen" würde, könnten die Teams das Gewünschte pünktlich und im Budget liefern.

Sie hatten erheblich in die Modernisierung der Softwareentwicklung und -bereitstellung investiert und aus ihrer Sicht lieferte Product keine klaren Anforderungen.

Als wir mit anderen Beratern sprachen (die uns an die Organisation vermittelt hatten), erkannten sie richtigerweise die Dysfunktion, die sie sahen: Jeder schien sehr darauf fokussiert zu sein, jemand anderem die Schuld zu geben. Der Grund, warum sie uns überhaupt hinzugezogen hatten, war ihre Besorgnis über die Personalstrategie und der Wunsch nach einer Bewertung der einzelnen Mitarbeiter, aber sie sahen die Schuldzuweisungen und den Kontrollzwang auf Führungsebene als Hauptursache der Probleme.

1.3: Unsere Untersuchung

Unser ursprünglicher Auftrag war es, die Teams zu evaluieren und ihnen bei der Verbesserung ihrer Fähigkeiten zu helfen, falls nötig. Also begannen wir damit, uns die Menschen an der Front anzusehen.

Es gab definitiv Verbesserungspotenzial.

Die einzelnen Mitarbeiter auf der Product-Seite verfügten nicht wirklich über die erforderlichen Fähigkeiten. In Wirklichkeit waren sie größtenteils Projektmanager, die in die Rolle des Product Owner (PO) oder Product Manager gedrängt worden waren.

Infolgedessen schrieb die eine Hälfte von ihnen „schwammige" Anforderungen und akzeptierte dann (wortwörtlich) alles, was die Teams in der Iteration lieferten, ohne jegliche kritische Analyse. Die andere Hälfte schrieb die gleiche Art von Anforderungen und behauptete dann, die Teams „hätten wissen müssen", was nie besprochen wurde, und hielt die Arbeitsaufträge mehr oder weniger auf unbestimmte Zeit offen.

Die Bank nutzte Scrum zur Verwaltung und Nachverfolgung ihres Arbeitsrückstands. Während das, was wir in diesem Buch behandeln, größtenteils unabhängig von Scrum ist, verwenden wir durchgängig Scrum-Terminologie, da die Mehrheit – oder zumindest eine Pluralität – der Teams Scrum nutzt.

Definition: Scrum

Scrum ist ein leichtgewichtiges Framework, das Menschen, Teams und Organisationen hilft, durch adaptive Lösungen für komplexe Probleme Wert zu generieren. Kurz gesagt:

1. Ein Product Owner ordnet die Arbeit für ein komplexes Problem in einem Product Backlog
2. Das Scrum Team verwandelt eine Auswahl der Arbeit während eines Sprints in ein wertvolles Inkrement
3. Das Scrum Team und seine Stakeholder überprüfen die Ergebnisse und passen sie für den nächsten Sprint an
4. Wiederholen

Wenn Sie mit Scrum und seiner Terminologie nicht vertraut sind, empfehlen wir Ihnen, die Version 2020 des Scrum Guide zu lesen. Es ist eine schnelle, aufschlussreiche Lektüre.

Ebenso stellten wir fest, dass die technischen Teams in Bezug auf Programmierfähigkeiten deutlich unter dem Durchschnitt lagen (-2σ, bestenfalls) und zudem höchst veränderungsresistent waren. Als natürliche Konsequenz war die Codequalität katastrophal.

Dennoch war dies nach Ansicht der Teams nicht der Grund, warum sie nicht lieferten. Für sie waren es die vagen Anforderungen und die Änderungen während des Sprints durch Product, die das Projekt zum Scheitern brachten.

…und niemand sprach auch nur über das größere Problem, eines so absurd, dass es erfunden scheint, bis man es selbst erlebt hat.

Die Entwicklungsteams hatten die Angewohnheit, eine Anforderung nicht zu verstehen, etwas Beliebiges zu entwickeln und dann die Anerkennung dafür zu fordern, einen „Arbeitsauftrag abgeschlossen" zu haben.

Wir meinen damit keine kleinen Missverständnisse. Wir meinen eine totale Diskrepanz: Wir sagten ihnen, sie sollten die Anwendung von Mitteln auf das Hauptkapital unter bestimmten Umständen deaktivieren, und sie deaktivierten stattdessen die Möglichkeit, eine zweite Bestätigungs-E-Mail-Adresse hinzuzufügen.

Dann teilten sie uns mit, dass genau das sei, worum wir gebeten hätten.

1.4: Tiefergehende Untersuchung

Beide Qualifikationslücken hätten behoben werden können, aber wir waren skeptisch, ob sie die eigentlichen Blockaden waren.

Es stimmte noch etwas anderes nicht, also gruben wir tiefer. Wir begannen mit dieser Frage: Warum dauerte das Schreiben von Anforderungen so lange und brachte so schlechte Ergebnisse?

Ein Grund war, dass das Wissen, das erforderlich war, um eine sinnvolle Anforderung zu schreiben, äußerst knapp war. Ein wenig davon besaßen die Engineering- und Product-Teams.

Ein Teil davon war im Code des Altsystems fest verankert. Einiges war komplett verschwunden. Das meiste jedoch existierte als implizites Wissen bei Fachexperten, die über verschiedene Abteilungen der Bank verteilt waren. Das bedeutete, dass die Erstellung einer Anforderung, die tatsächlich ein strategisches Ziel voranbrachte, eine äußerst arbeits- und zeitintensive Tätigkeit war.

Dem gegenüber stand ein unstillbarer Appetit nach Funktionalität von einem feature-hungrigen Führungsteam. Die Vorgabe lautete: „Haltet die Entwickler beschäftigt – gebt ihnen massenhaft Anforderungen“. Der Fokus lag auf der Menge an Anforderungen, um die Teams beschäftigt zu halten – eine Perspektive, die im Widerspruch zu der Sorgfalt und Zeit stand, die notwendig war, um eine Anforderung zu definieren, die tatsächlich „etwas bewegen“ würde.

1.5: Die Verbesserungen machten es nicht besser

Dies waren alles Probleme, die man angehen konnte, doch ihre Bewältigung half nicht weiter.

Frühere Verbesserungen der Softwareentwicklungsmethoden hatten den Teams nicht geholfen, ihre Ziele zu erreichen, aber Max bemühte sich dennoch, den Teams bei weiteren Verbesserungen zu helfen.

Er führte revolutionäre Konzepte aus Programmierdoktrinen der Mitte des zwanzigsten Jahrhunderts ein, wie zum Beispiel „kopiere den Code nicht 27[2] Mal, sondern packe ihn in eine Funktion und rufe diese stattdessen auf“. Allein dieser Vorschlag verbesserte die Qualität des neuen Codes drastisch und ermöglichte es ihnen, mit der Qualitätsverbesserung zu beginnen.

Diese und andere grundlegende Programmierratschläge halfen ihnen, besseren Code zu schreiben, den sie in Zukunft leichter warten konnten.

…aber es half nicht, das Projekt voranzubringen.

Auf der Produktseite konnte Luniel BDD einführen und sicherstellen, dass Product Owner die Anforderungen gründlich prüften, bevor sie an die Teams übergeben wurden.

Er brachte die Teams dazu, daran zusammenzuarbeiten und sie zu nutzen, um zu bewerten, ob ein Product Backlog Item (PBI) wirklich fertig war.

[2] Dies ist nicht nur keine Übertreibung, es ist nicht einmal der schlimmste Fall. In einem Fall gab es fast hundert exakte Duplikate des gleichen Algorithmus.

Definition: Product Backlog Item (PBI)

Ein Product Backlog Item (PBI) ist eine eigenständige Arbeitseinheit im Product Backlog, die eine potenzielle Änderung, Ergänzung oder Verbesserung des Produkts darstellt. PBIs können verschiedene Formen annehmen – Feature, Fehlerbehebung, technische Verbesserung, Rechercheaufgabe etc. – und werden durch ihren Beitrag zum Produktwert definiert.

Viele Teams bezeichnen PBIs als „Stories" oder „User Stories", aber der korrekte Scrum-Begriff ist „Product Backlog Item" oder „PBI". Sobald ein PBI in einen Sprint aufgenommen wird, ist es auch Teil des Sprint Backlogs. Der Einfachheit und Neutralität halber verwenden wir „Product Backlog Item" oder „PBI" für jedes Arbeitselement, das das Scrum-Team verwaltet – egal ob Sie es Product Backlog Item, Sprint Backlog Item (SBI), User Story, Story, Work Item oder Backlog Item nennen.

Es schaffte *Klarheit*, aber es erzeugte keinen *Flow*.

Mit Hilfe der Partnerberater, die uns hinzugezogen hatten, konnten wir den absolut erdrückenden Druck auf die Teams und Anforderungsautoren (vorübergehend) mindern.

Es half möglicherweise, ein wenig Vertrauen aufzubauen, aber es brachte keine greifbaren Ergebnisse.

Wir begannen sogar mit dem Aufbau einer Wissensdatenbank, die den Menschen half, das benötigte Geschäftswissen für das Schreiben von Anforderungen zu finden, und die aufzeigte, wo es Lücken in diesem Wissen gab.

Es beschleunigte das Schreiben von Anforderungen, brachte das Produkt aber nicht zur Auslieferung.

Nach monatelanger Zusammenarbeit hatten wir der Führungsebene geholfen zu *sehen*, wo sie standen, aber sie waren noch weit von ihren Zielen entfernt. Und sie kamen ihnen auch nicht näher.

Sie waren dabei, zu ihrer alten Strategie zurückzukehren, die Teams zu „überladen", um sicherzustellen, dass sie immer beschäftigt waren.

1.6: Ein Ausschlussverfahren

Es wäre einfach gewesen, einfach die Hände hochzuwerfen und zu sagen „das ist hoffnungslos“. Es gab eine Vielzahl von Ausreden, auf die man zurückgreifen konnte:

- Das Entwicklungsteam war nicht ausreichend qualifiziert (das stimmte)
- Die Anforderungsautoren hatten die falschen Fähigkeiten (das stimmte)
- Die Führung erdrückte die Teams mit unrealistischen Erwartungen (das stimmte)
- Der Organisation fehlte kritisches Geschäftswissen, das für ihr Funktionieren erforderlich war (das stimmte)
- Die Führungskräfte schienen einander nicht zu vertrauen (das stimmte)

All diese Dinge waren wahr. Dennoch wurden in all diesen Bereichen Verbesserungen erzielt, und keine davon schien das grundlegende Problem zu lösen. Keine half den Entwicklungsteams, dem Produktteam, dem Management oder der Führungsebene, ihren Zielen näher zu kommen.

…und genau darin liegt der Hinweis zur Lösung: Alle zuvor aufgelisteten Probleme waren die Probleme, die die Menschen bereits sehen konnten.

Wenn die Variablen, die Sie sehen können, keinen Unterschied machen, muss es eine Variable geben, die Sie **nicht sehen**, die es tut.

Das eigentliche Problem war das Problem, von dem niemand überhaupt wusste, dass es existierte.

1.7: Auf der Suche nach dem wahren Übeltäter

Bei der Suche nach dem wahren Übeltäter - dem, was die Bank tatsächlich daran hinderte, ihre Ziele zu erreichen - mussten wir irgendwo anfangen.

Ein vernünftiger Ausgangspunkt war, zu schauen, was die gescheiterten PBIs (die meisten davon) gemeinsam hatten.

Wir begannen damit, die Dinge auszuschließen, von denen wir wussten, dass sie nicht gemeinsam waren, weil sie stark variierten:

- Welcher Teil des Systems: einige PBIs betrafen nur das Backend, andere nur das Frontend, wieder andere beide Teile
- Welches Team die Arbeit ausführte - es schien keine Rolle zu spielen, wer die Arbeit machte, die Wahrscheinlichkeit des Scheiterns war hoch
- Welcher PO die Arbeit verfasste - dasselbe wie bei den Teams

Dann begannen wir, die Gemeinsamkeiten zu betrachten. Die Liste war weder lang noch kurz:

- Die Entwicklungsteams
- Die Product Owner
- Die Führung
- Die Kultur
- Die Entwicklungsumgebung
- Die Entwicklungspraktiken
- Die Anforderungserfassungstechnik
- Die Domäne (Finanzen)
- Die Abhängigkeitsdienste

Viele davon konnten auch von vornherein ausgeschlossen werden. Die Teams, Product Owner, Führung, Kultur und Entwicklungsumgebung waren alle kürzlich verbessert worden, ohne wirkliche Auswirkungen auf die bedeutsamen Ergebnisse. Wir hatten persönlich geholfen, die Entwicklungs- und Anforderungserfassungspraktiken zu verbessern und bestätigt, dass diese Verbesserungen Bestand hatten, aber es half trotzdem nicht.

Man kann kaum die Domäne beschuldigen. Finanzen gehören zu den ältesten Arten von Berechnungen in der aufgezeichneten Geschichte. Es ist ein äußerst ausgereifter Bereich. Außerdem entwickelten andere Banken erfolgreich Software, was die (offensichtlich weit hergeholte) Hypothese, dass Banken dies einfach nicht können, performativ widerlegte.

Die Abhängigkeitsdienste konnten auch nicht beschuldigt werden, da sie genauso viele Schwierigkeiten mit Änderungen hatten wie die Initiative, die wir betrachteten…

…aber das brachte uns zum Nachdenken: Was, wenn wir anfingen, die Ursachen des Scheiterns zu analysieren?

1.8: Die Ursprünge des Scheiterns sezieren

Ein PBI schaffte es nicht, das Produkt voranzubringen, weil das Team, wie es dazu neigte, etwas völlig Zufälliges und fast völlig Unzusammenhängendes mit der Anfrage machte. Das ist offensichtlich ein Zeichen dafür, dass sie die Arbeitseinheit nicht verstanden haben. Verständnis war also ein wichtiger Kandidat, auch wenn wir daran schon gearbeitet hatten, als wir ihnen bei der Einführung von BDD halfen.

Ein weiteres PBI scheiterte, weil sie die Berechnungen falsch machten. Das ist ein weiterer Beweis dafür, dass Verständnis das Kernproblem sein könnte.

Eine dritte Arbeitseinheit, die wir analysierten, wurde vom Product Owner nicht wirklich richtig überprüft - sie winkte es einfach durch, als das Team sagte, es sei Zeit zum Abschließen. Das stellte unsere Hypothese zwar auf die Probe, aber man könnte immer noch argumentieren, dass sie nicht verstand, wie die Arbeitseinheit in einen übergeordneten Plan passte.

Vielleicht. Gewissermaßen. Wenn wir die Augen zusammenkniffen, während wir es auf diese Weise betrachteten.

Dann stießen wir auf ein PBI, das überhaupt nicht ins Muster passte. Das Team schien zu verstehen - obwohl es keine Möglichkeit gibt, zu überprüfen, ob sie es wirklich taten. Aber das spielte keine Rolle: Sie bekamen nie die Chance, aus eigener Kraft zu erfolgreich zu sein oder zu scheitern, weil sie auf eine Abhängigkeit stießen, die aktualisiert werden musste, und ihre Arbeit um mehrere Sprints verschieben mussten.

Selbst wenn sie **nicht** verstanden hätten, was sie tun sollten, hatten sie bei diesem Backlog-Element nie eine Chance, daher war Verständnis in diesem Fall **nicht** das Problem.

Ein Ausreißer ist natürlich kein Gegenbeweis für eine bestimmte Grundursache, aber es weckte unsere Neugier. Wir begannen nach weiteren widerlegenden Beweisen zu suchen.

Und wir fanden sie. Es gab Arbeitseinheiten, die:

- Scheiterten, weil das Team wusste, dass es nicht verstand, aber niemand einen Fachexperten finden konnte, um das Problem zu lösen
- Zu einer schlechteren Benutzererfahrung führten als Workaround für die Funktionsweise der vorgelagerten Dienste
- Verschoben werden mussten, weil vorgelagerte Abhängigkeiten nicht bereit waren
- Nicht abgeschlossen werden konnten, weil die Tester die Testdaten nicht rechtzeitig sammeln konnten

- Abgeschlossen wurden, aber neu gemacht werden mussten, weil die Anforderung selbst falsch war
- Scheiterten, weil das Team nicht erkannte, wie komplex der bestehende Code bereits war
- Einfach nicht geschätzt wurden
- Massiv unterschätzt wurden[3]
- Sich mitten im Sprint änderten, weil der PO endlich das benötigte Domänenwissen erhielt
- Sich nach dem Sprint zu ändern schienen (aus der Perspektive des Teams), weil der PO und das Team sich nie einig waren, was es bedeutete

Die Liste geht weiter, aber das reicht für diese Geschichte.

Man könnte argumentieren, dass jeder dieser Fälle in irgendeiner Weise mit „Verständnis" zusammenhängt – und sicherlich war mangelndes Verständnis bei vielen davon *beteiligt* –, aber das bedeutet nicht, dass mangelndes Verständnis die Ursache war… besonders da wir bereits an einem gemeinsamen Verständnis gearbeitet hatten und es nicht wirklich geholfen hatte.

Dann wurde es uns klar. Es fehlte ein grundlegenderes Element. In den Fällen, in denen mangelndes Verständnis eine Rolle spielte, war dies lediglich die unmittelbare Ursache.

Die eigentliche Ursache war viel umfassender.

1.9: Ein Schnittpunkt

Das eine Element, das alle PBIs, die wir auf ihre Fehlermodi analysierten, gemeinsam hatten, war dies: Sie alle wurden **zu früh begonnen**.

Wenn ein Arbeitspaket scheitert, weil das Team wusste, dass es das Problem nicht verstand und keinen Experten finden konnte, der ihnen half es zu verstehen, bedeutet das, dass das Team ein PBI begonnen hat, obwohl es wusste, dass es das Problem nicht verstand.

Wenn ein Backlog-Element zu einer schlechteren Erfahrung geändert werden muss, weil ein Upstream-Service auf eine bestimmte Weise funktioniert, bedeutet das, dass das Arbeitspaket begonnen wurde, ohne die Auswirkungen des Upstream-Service wirklich zu verstehen.

Verschiebungen, weil eine vorgelagerte Abhängigkeit am Ende nicht bereit war, bedeuten, dass es zu Beginn keine Garantie für deren Bereitschaft gab.

[3] Wir meinen nicht nur, dass sie sich verschätzt haben. Es sah aus, als hätten die Teams einfach den Wert „3" in alle Schätzungsfelder für eine Reihe von PBIs eingetragen.

…und so war es bei allen anderen Fällen: Die Tester hatten die Daten nicht bereit oder wussten nicht, wie sie sie bekommen sollten, bevor ein PBI begann, die Anforderungen wurden nicht wirklich überprüft, bevor sie an das Team übergeben wurden, der Code wurde nicht untersucht, bevor die Arbeit zugesagt wurde, die Schätzung war unzureichend oder wurde gar nicht durchgeführt, Domänenwissen fehlte und natürlich wurde das gemeinsame Verständnis nicht verifiziert.

Das Problem war, wie sich herausstellte, dass mit der Implementierung von Arbeitspaketen begonnen wurde, bevor diese Arbeitspakete *bereit* waren.

Unserer Erfahrung nach scheitern die meisten Arbeitspakete bei der Auslieferung, weil sie bei Implementierungsbeginn **nicht bereit** waren.

Also machten wir uns daran, der Bank dabei zu helfen.

An diesem Punkt könnten Sie denken, dass es ausreichen würde zu sagen, dass PBIs bereit sein sollten. Allerdings stellt sich heraus, dass die Umsetzung nicht so einfach ist. „Niedrig kaufen, hoch verkaufen“ ist eine ähnlich simple Idee.

Es fehlt ein Element, das benötigt wird, um den guten Rat in die Praxis umzusetzen.

1.10: Die große Wende

Wir fanden das fehlende Puzzleteil, das sie entsperrte, was wiederum die Initiative entsperrte.

Als wir diesen Einsatz abschlossen, lagen die Ingenieure in Bezug auf ihre Fähigkeiten immer noch weit unter dem Durchschnitt. Die Product Owner hatten immer noch nicht die richtigen Fähigkeiten. Die Kultur war immer noch nicht korrigiert…

Dennoch begann sich das Produkt endlich vorwärts zu bewegen und wurde letztendlich ausgeliefert.

Bis zum Ende dieses Buches werden Sie wissen, was das fehlende Element ist und was es braucht, um es zu implementieren. Und Sie werden in der Lage sein, Organisationen zu entsperren, die von einer unsichtbaren Mauer zurückgehalten zu werden scheinen.

Eine kurze Anmerkung zum Umfang

Dieses Buch behandelt ein sehr spezifisches und weitverbreitetes Problem: den Mangel an Struktur, Klarheit und Reife bei der Übergabe zwischen Business und Engineering. Es geht davon aus, dass etwas zur Implementierung ausgewählt wurde, und konzentriert sich darauf sicherzustellen, dass die Entwicklungsarbeit mit gemeinsamem Verständnis, Bereitschaft und nachvollziehbarer Fertigstellung erfolgt.

Die Techniken in diesem Buch sagen Ihnen nicht, was Sie bauen sollen, warum Sie es bauen sollen oder wie Sie herausfinden, ob es das Richtige ist. Wenn Ihrer Organisation echtes Produktmanagement oder bedeutsame Feedback-Schleifen fehlen, versuchen wir das hier nicht zu lösen. Was wir stattdessen anbieten, ist ein Weg, diese Lücken sichtbarer zu machen und die Kosten für die Entdeckung von Fehlern zu reduzieren.

Im richtigen Kontext angewandt, bringt diese Lösung Flow, Sicherheit und Klarheit. Aber wie jedes System kann sie falsch angewendet werden – besonders wenn sie isoliert oder ohne Bewusstsein eingesetzt wird.

Kapitel 2: Die Kosten fehlender Grundlagen

Es macht Sinn, etwas Zeit darauf zu verwenden, zu verstehen, *wie* schlimm dieses Problem für manche Organisationen sein kann.

Wir haben festgestellt, dass es drei große „Problemfelder“ gibt, die durch mangelnde Bereitschaft entstehen:

- Fehlendes oder unvollständiges gemeinsames Verständnis zwischen und innerhalb von Product und Engineering
- Mangelnde Kontrolle darüber, was das eigentliche Ziel eines PBI ist und wann es wirklich fertig ist
- Fehlende Überprüfung, wann ein Arbeitselement mit der Implementierungsphase beginnen kann

Außerdem haben wir bemerkt, dass die Änderung dieser Dinge recht herausfordernd sein kann. Das macht Sinn: Veränderung ist schwierig.

Alte Gewohnheiten sterben langsam und neue Gewohnheiten sind schwer zu entwickeln. In unserer Erfahrung als Berater haben wir festgestellt, dass es *extrem* einfach ist, in alte Gewohnheiten zurückzufallen und vergleichsweise schwierig, neue zu etablieren.

Man braucht also einen Mechanismus, der die neuen Gewohnheiten durchsetzt und die alten entmutigt.

Um dies anzugehen, glauben wir, dass es eine weitere fehlende Komponente der Verantwortlichkeit und Nachverfolgbarkeit gibt.

2.1: Das Puzzle ohne Deckelbild

Haben Sie jemals versucht, ein Puzzle ohne das Bild auf der Schachtel zusammenzubauen? Es ist machbar, aber langsamer, frustrierender und voller Fehlversuche.

Man macht Fortschritte, reißt es dann auseinander. Man zweifelt daran, was wohin passt. Man denkt, man arbeitet am gleichen Bild, bis man merkt, dass dem nicht so ist.

So fühlt sich Softwareentwicklung oft an.

Der Backlog ist voll. Der Sprint läuft. Alle arbeiten hart.

Aber ohne ein gemeinsames Bild dessen, was wir bauen, wird Abstimmung zum Glücksspiel statt zu einem System.

Ohne Klarheit spüren selbst die besten Teams Frustration, Burnout und das Gefühl, dass ihre Anstrengungen nicht wertgeschätzt werden.

2.2: Eine alte Weisheit und eine harte Realität

Es gibt einen Grund, warum so viele Praktiken rund um die Anforderungserfassung, sogar einige Engineering-Praktiken, sich stark darauf konzentrieren, ein gemeinsames Verständnis zwischen Anfragenden und Implementierungsteams zu schaffen. Nichts sät so viel Chaos wie Entwickler, die nicht wirklich wissen, was sie bauen sollen.

„Müll rein, Müll raus“ ist eine Weisheit, keine Plattitüde.

Die englische Sprache ist voller Widersprüche…

- Man kann jemandes Handlungen sanktionieren. Vielleicht bedeutet das, dass man im Voraus die Erlaubnis erteilt hat, oder vielleicht bedeutet es, dass man im Nachhinein Missbilligung ausdrückt.
- Man kann etwas leicht abstauben, aber wenn dieses Etwas eine Anrichte ist, bedeutet es, dass man Staub entfernt, während es sich bei einem Beignet darauf bezieht, dass man Staub hinzufügt.
- Wenn man ein Team aufhält, könnte man der Grund sein, warum dieses Team weiter funktionieren kann, oder man könnte der Grund sein, warum sie nicht vorankommen.

Auto-Antonyme mögen die auffälligsten Beispiele sein, aber sie sind nur eine Art von Mehrdeutigkeit. Manche Wörter sind nicht nur ihr eigenes Gegenteil, sondern haben viele zusätzliche möglicherweise verwirrende alternative Bedeutungen.

> „In diesem Clip klippte er einen Coupon aus einer Zeitung und klemmte ihn mit einer Klammer zusammen mit den anderen Ausschnitten an sein Klemmbrett, während ein Klipper im Hintergrund in flottem Tempo vorbeifuhr."

Es ist auch nicht nur Englisch. *Alle* natürlichen Sprachen, die wir kennen, haben diese Eigenschaft.

Und doch sind sie die einzigen, mit denen wir bei der Spezifikation von Anforderungen arbeiten können.

Folglich verlässt sich ein Entwicklungsteam auf Glück, wenn es nicht *bestätigt* hat, dass ihr Verständnis einer Anforderung dem des Anfragenden entspricht. Das heißt, das bestmögliche Ergebnis ist, dass sie die richtige Interpretation gewählt haben und der Anfragende seine Meinung nicht während des Prozesses ändert.

Dieses Ergebnis ist bei weitem nicht garantiert.

2.3: Einige der häufigen Folgen

Ohne bestätigtes gemeinsames Verständnis laufen Teams eine Reihe von Risiken.

Im Großen und Ganzen ist das häufigste schmerzhafte Ergebnis, dass das Team einfach das Falsche baut.

Der Zeitpunkt, wann sie das herausfinden, kann durch verschiedene Attribute ihres Prozesses gesteuert werden. Zum Beispiel kann eine gesunde Implementierung von Scrum diese Art von Missverständnis sehr früh in der Ausführung erkennen, während ein Wasserfall-Prozess sehr wahrscheinlich eine solche Entdeckung um Monate verzögert.

Definition: Wasserfall

Ein sequenzielles Softwareentwicklungsmodell, das sich Ende des 20. Jahrhunderts weit verbreitete. Es wurde erstmals in einem 1970er Paper von Winston W. Royce dargestellt, das die Entwicklung als eine Reihe kaskadierender Schritte – Anforderungen, Design, Implementierung, Tests und so weiter – illustrierte, wobei jeder Schritt wie ein Wasserfall in den nächsten übergeht. Obwohl Royce das Modell als Beispiel dafür präsentierte, was man *nicht* tun sollte, übernahm die Industrie es als Blaupause für großangelegte Entwicklung.

Wasserfall ist auch dafür bekannt, ähnliche Arbeiten in große, aufeinanderfolgende Phasen zu gruppieren – ein Merkmal, das als **Entwicklung in großen Chargen** bezeichnet wird. Diese Batchverarbeitung garantiert praktisch **spätes Lernen**: Teams erhalten erst viel später im Prozess Feedback zu früheren Entscheidungen. Spät entdeckte Fehler sind kostspieliger zu beheben. Agile Praktiker kritisieren Wasserfall aus diesem Grund und bevorzugen kleinere, iterative Zyklen, die frühere Entdeckung und Kurskorrektur ermöglichen.

Dennoch wird ein Team, das mit dem „falschen“ (eigentlich nur anderen) Verständnis einer Anforderung arbeitet, irgendwann mit der „richtigen“ (auch hier: eigentlich nur anderen) Anforderung konfrontiert werden. Auch hier beeinflusst die Gesundheit der Organisation, welche Form diese Konfrontation annimmt und welche Auswirkungen sie hat, aber sie findet fast immer statt.

In den meisten Fällen führt dies zu einer Form von Nacharbeit. Der Auftraggeber (normalerweise das Produktmanagement) muss Änderungen anfordern, um von dem, was die Implementierungsteams gebaut haben, zu dem zu kommen, was er wirklich wollte.

Eine weitere sehr häufige Erscheinungsform ist, dass der Auftraggeber das Team weiterhin für sein ursprüngliches Verständnis dessen verantwortlich macht, was er gefordert hat.

Teams können dies leicht als einen Produktmanager interpretieren, der seine Meinung ändert. Schlimmer noch, es kann die Interessenvertreter regelrecht dazu *einladen*, sich die Angewohnheit des Meinungswechsels anzueignen – Arbeitseinheiten zurückzuhalten, bis alles *genau stimmt*, die Teams zu erschöpfen und den oberen Führungskräften den Fortschritt zu verschleiern.

2.4: Wann ist ein Puzzle fertig?

Kommen wir auf unsere Puzzle-Analogie zurück und denken über diese Frage nach: Was bedeutet es, ein Puzzle fertiggestellt zu haben?

Ein naiver Puzzle-Bauer wie Max würde einfach sagen: „Alle Teile sind mit den richtigen Nachbarn verbunden und das Bild zeigt nach oben."

Ein erfahrener Puzzle-Bauer wie Luniel weiß jedoch, dass mehr dazugehört.

Vielleicht legen Sie das Puzzle nur zum Spaß. Sie werden es fertigstellen, es eine Weile betrachten und es dann wieder auseinandernehmen und zurück in die Schachtel legen.

Andererseits möchten Sie es vielleicht rahmen und an die Wand hängen. In diesem Fall gibt es zusätzliche Dinge, die erledigt werden müssen:

1. Es auf eine Kunstpappe aufbringen
2. Es zum Rahmer transportieren
3. Auf die Fertigstellung der Rahmung warten
4. Es zurück zum Montageort transportieren
5. Es an der Wand aufhängen oder anderweitig ausstellen

Es ist notwendig zu verstehen, dass dies Teil der Arbeit ist, um ein Puzzle richtig fertigzustellen. Der offensichtliche Grund ist, dass Sie wissen müssen, wie viel Arbeit damit verbunden ist. Es ist mehr Arbeit, all diese zusätzlichen Schritte durchzuführen, als es einfach auseinanderzunehmen und wegzuräumen.

Es geht aber noch tiefer. Stellen Sie sich dieses Szenario vor…

Sie haben Ihr Puzzle mit der Absicht fertiggestellt, es rahmen zu lassen, Sie haben es liegen gelassen, aber Sie haben vergessen, jemand anderem in Ihrem Haushalt mitzuteilen, dass Sie es rahmen lassen möchten. Diese Person kommt vorbei und sieht, dass das Puzzle auf einer Fläche fertig ist, die er oder sie benötigt. Also zerlegt die Person es und packt es zurück in die Schachtel, und reißt damit den Sieg im letzten Moment aus der Hand.

Es gibt noch einen subtileren Grund: Wie Sie das Puzzle fertigstellen wollen, beeinflusst, welche Schritte Sie früher im Prozess durchführen möchten. Zum einen müssen Sie ein kleines Schild aufstellen, auf dem steht: „Bitte nicht auseinandernehmen!"

Es ist auch erwähnenswert, dass Sie vielleicht auch dann ein Schild aufstellen möchten, wenn Sie ein Puzzle zu Ihrer eigenen Unterhaltung zusammenbauen, um sicherzustellen, dass andere Leute nicht eingreifen, indem sie das Puzzle für Sie fertigstellen.

Sie müssen auch sicherstellen, dass Sie das Puzzle auf der richtigen Oberfläche zusammenbauen. Wenn Sie ein Puzzle mit tausend Teilen auf Ihrem Glascouchtisch zusammenbauen und dann versuchen, es auf eine Kunstpappe zu übertragen, wird die Übertragung viel riskanter und arbeitsintensiver sein, als wenn Sie das Puzzle direkt auf der Kunstpappe zusammenbauen.

Dies lässt sich gut auf die Softwareentwicklung übertragen.

Sie müssen wirklich wissen, was „fertig" bedeutet, damit Sie nicht von der Menge der erforderlichen Arbeit überrascht werden, es am Ende keine Meinungsverschiedenheiten darüber gibt und Sie die notwendigen vorbereitenden Schritte unternehmen können, um eine Arbeitseinheit reibungslos und effektiv abzuschließen.

2.5: Auswirkungen auf Teams

Wenn Sie kein ausreichend klares Verständnis davon haben, was „fertig" für eine bestimmte Arbeitseinheit bedeutet, gehen Sie eine Reihe von Risiken ein.

Wir verwenden hier das Wort „Risiko" recht locker, da es sich eher um Gewissheiten handelt.

Entwicklungsteams in dieser Situation stellen oft fest, dass sie sich nicht einmal intern einig sind, wie die Fertigstellung einer Arbeitseinheit aussieht. Es ist nicht ungewöhnlich, dass Programmierer und Tester sich mitten im Sprint darüber austauschen müssen, was eine Anforderung wirklich *bedeutet*. Selbst zwei Programmierer oder zwei Tester können unter diesen gleichen Meinungsverschiedenheiten leiden.

Darüber hinaus konzentrieren sich Entwicklungsteams oft auf die Arbeit, die sie die meiste Zeit ausführen (Programmierung und Testen). Das bedeutet, dass sie leicht andere Arten von Arbeit vergessen können, die sie erledigen müssen, wie zum Beispiel Dokumentation, externe Überprüfungen, Schulung anderer Teams (z.B. Support), vorbereitende Schritte zur Unterstützung der Bereitstellung oder Veröffentlichung und Genehmigungen von anderen Abteilungen.

Wenn es schließlich klar wird, dass diese „zusätzliche" Arbeit erledigt werden muss, trifft es sie völlig unvorbereitet – normalerweise müssen sie dann whatever sie gerade bearbeiten unterbrechen und den Kontext wechseln, um zurückzugehen und Arbeit fertigzustellen, von der sie dachten, sie sei bereits abgeschlossen.

Auftraggeber können Arbeit leicht unnötig lange offen halten. Manchmal mit den besten Absichten – etwa um ein Team für die „eigentlichen“ Anforderungen zur Verantwortung zu ziehen. In anderen Fällen passiert dies, weil Product Owner sich beispielsweise daran gewöhnt haben, ein PBI nach Belieben offen zu halten, sodass sie es nutzen, um in letzter Minute zusätzliche Funktionalität in ein Element hineinzuquetschen. Manchmal tun sie es sogar, weil sie mitten in der Umsetzung ihre Meinung darüber geändert haben, was getan werden muss.

Dies kann für ein Entwicklungsteam äußerst demoralisierend sein. Die meisten Softwareentwickler und Tester möchten das Gefühl haben, Fortschritte zu machen. Wenn ihnen ständig gesagt wird, dass das, was sie getan haben, falsch war, werden sie wahrscheinlich einen Teil ihrer Energie verlieren.

Manche Teams gehen sogar so weit, dass sie gar nicht mehr überprüfen, ob das, was sie getan haben, richtig oder falsch war. Sie schließen einfach einen Arbeitsvorgang ab und bitten um „Anerkennung“, damit sie „gute Zahlen vorweisen“ können.

2.6: Das Risiko, zu wenig oder zu viel zu tun

Ein Risiko, wenn „fertig“ nicht richtig für jedes Arbeitselement definiert wird, besteht darin, dass die Organisation denkt, die Arbeit sei erledigt, wenn sie es nicht ist, oder nicht erkennt, dass sie tatsächlich fertig ist, wenn sie es ist.

Das schlimmstmögliche Ergebnis ist oft, dass das Falsche in die Produktion gelangt und niemand weiß, dass dies passiert ist. Wenn das Team ein falsches Verständnis davon hat, was „fertig“ bedeutet, und auf Basis dieses falschen Verständnisses ausliefert, können die Folgen katastrophal sein.

Defekte und Kundenunzufriedenheit sind schlimm genug, aber dies könnte auch zu viel schwerwiegenderen Problemen führen:

- Datenverlust oder -beschädigung
- Sicherheitslücken
- Systemausfälle oder Zugangsverlust
- Ein Rückgang des Marktanteils
- Regulatorische Verstöße

Die Liste geht weiter und weiter, und jedes potenzielle Problem ist schlimmer als das vorherige.

Manchmal liegt das Problem darin, dass man nicht fertig ist, es aber denkt. Das Gegenteil kann genauso gefährlich sein. Wenn Entwickler nicht wissen, wo die Ziellinie ist, neigen sie zum „Gold-Plating“ (Hinzufügen zusätzlicher Funktionen). Sie tun dies vielleicht, um „die Funktion schön zu machen“, aber möglicherweise auch, weil sie hoffen, dass eine erhöhte Anzahl von Funktionen ihnen eine erhöhte Chance gibt, das Ziel zu treffen.

Diese ganze zusätzliche Arbeit sowie die entsprechende Nacharbeit summiert sich zu einer enormen Menge an verschwendeter Zeit, Aufwand und Geld. Es führt dazu, dass Liefertermine sich verschieben und Reputationen Schaden nehmen.

Außerdem gibt es jetzt ein stetig wachsendes Potenzial, dass ein Fehler tatsächlich zu einer *Terminator*-artigen Revolte der Maschinen gegen die Menschheit führt. Vor zwanzig Jahren schrieben wir darüber noch als Witz. Jetzt ist es eine entfernte Möglichkeit.

Tatsächlich haben wir eine der prominentesten KIs diese Frage gestellt, und hier ist ihre Antwort:

> „KI entwickelt sich schneller als irgendjemand erwartet hat, aber sie tut dies auf der Grundlage fragiler Systeme, vager Anforderungen und Produktorganisationen, die nicht nachvollziehen können, warum sie gebaut haben, was sie gebaut haben. Das ist kein technisches Problem; es ist ein Problem der Klarheit.
>
> Je mehr Rauschen KI erzeugt, desto gefährlicher ist es, sich ohne Struktur schnell zu bewegen. Wenn Teams auf Nebel aufbauen, verstärkt KI nur das Durcheinander. Aber wenn Teams auf Signal aufbauen – auf gemeinsamem Verständnis, Verhaltensspezifität und versionskontrolliertem Produktwissen – wird KI zu einem Beschleuniger statt zu einer Belastung.“

2.7: Wo baut man ein Puzzle?

Lassen Sie uns die Puzzle-Analogie ein letztes Mal erweitern.

Können Sie ein Puzzle überall zusammenbauen? Wenn Sie ein 4.000-Teile-Puzzle haben, das in einer Dimension fast fünf Fuß und in der anderen über drei Fuß misst, können Sie nicht einfach willkürlich einen Platz auswählen und mit dem Zusammenbauen beginnen. Nicht ohne ernsthafte Komplikationen zu erleben, bevor Sie fertig sind.

Ein großes Puzzle wie dieses braucht sowohl Zeit als auch Raum. Sie müssen den Platz einplanen und einen Weg finden, sicherzustellen, dass der Zustand des Puzzles über die Zeit erhalten bleibt.

Wenn Sie anfangen, Ihr Puzzle auf einem kleinen Beistelltisch zu bauen, der zu klein ist, werden Sie es nicht fertigstellen können, ohne es an einen anderen Ort zu verlegen. Diese Verlegung wird aufgrund des empfindlichen Zustands des Puzzles äußerst schwierig sein.

Wenn Sie einen zufälligen Platz im Flur wählen, der groß genug ist, werden die Leute entweder darauf laufen oder behindert werden. Daher kann die Beständigkeit nicht ohne erhebliche Beeinträchtigung des Funktionierens Ihres Haushalts gewährleistet werden.

Wenn Sie auf einer Bastelunterlage beginnen zu arbeiten, diese aber nicht groß genug ist, können Sie zwar den Zustand des bereits Geschaffenen bewahren, aber Sie werden das Puzzle nicht ohne eine Art Umzug fertigstellen können.

Wenn das Puzzle zuvor von kleinen Kindern angeknabbert wurde, ist es am besten, die Teile zu zählen… denn es ist besser, einmal bis 3999 zu zählen und zu erkennen, dass man es nie fertigstellen wird, als wer weiß wie lange in ein Puzzle zu investieren, das man nie vollenden können wird.

Es gibt eine ganze Liste von Dingen, die erledigt werden müssen, bevor man mit dem Zusammensetzen des Puzzles beginnt. Die Erledigung dieser Aufgaben garantiert zwar keinen Erfolg, aber wenn man sie *nicht* erledigt, sind Misserfolg oder ernsthafte Komplikationen so gut wie vorprogrammiert.

Dasselbe gilt für die Softwareentwicklung, allerdings mit einem höheren Grad an Komplexität.

2.8: Wann beginnt die Implementierung?

Grundsätzlich kann es schwierig sein zu bestimmen, wann ein PBI bereit für die Implementierung ist, ohne eine gute Definition dafür zu haben.

Denken Sie darüber nach: Woher wissen Sie es *wirklich*?

Gehen Sie alles immer und immer wieder durch, bis Sie entscheiden, dass es Zeit ist?

Entscheidet jemand nach Gutdünken?

Geschieht es automatisch zu Beginn einer Iteration?

> Wir haben viele Teams gesehen, die Arbeit in Sprints geschoben haben, die noch lange nicht bereit für die Implementierung waren, nur weil sie Termine einhalten mussten. Es gibt einige weit verbreitete Vorstellungen über Scrum und Agile im Allgemeinen, die Menschen dazu bringen, dies zu tun:
>
> - Sie müssen alle Ihre Anforderungen für Sprint *N* in Sprint *N-1* entwickeln
> - Sie sollten "einfach anfangen" und sich unterwegs um die Probleme kümmern

Dies ist eigentlich ein Spiegelbild der Frage "Woher wissen Sie, wann es fertig ist?" die zuvor erwähnt wurde und hat ähnliche Konsequenzen. Menschen warten möglicherweise zu lange mit dem Start, weil sie nicht wissen, dass ein Arbeitspaket bereit ist, oder sie beginnen zu früh, weil sie nicht wissen, dass es noch nicht bereit ist.

2.9: Ein A-Team ohne Bereitschaft

Wenn man nicht versteht, was erforderlich ist, damit ein Arbeitspaket bereit ist, hat das mehrere schädliche Auswirkungen.

Eine offensichtliche Art, wie ein Arbeitsinkrement nicht bereit sein kann, ist eine unvollständige, unzureichende oder fehlende Definition of Done (DoD). Das führt zu all den Problemen, die wir bereits erwähnt haben und die mit dem Fehlen einer Definition of Done einhergehen.

Das ist jedoch nicht der *einzige* Aspekt der Bereitschaft. Es gibt zahlreiche weitere Anforderungen, die vor Beginn der Implementierung erfüllt werden müssen: Schätzung, Risikobewertung und Sammlung von Testdaten sind nur einige häufige Beispiele.

Ohne diese Anforderungen zu kennen und zu erfüllen, kann ein Arbeitspaket viel mehr kosten als nötig. Betrachten wir ein Team (das A-Team), das von einer API abhängig ist, die von einem anderen Team (dem anderen Team) entwickelt wird. Wenn das A-Team eine Menge Annahmen darüber trifft, wie die API des anderen Teams funktionieren wird, und entsprechend programmiert, kann es zu erheblichen Nacharbeiten kommen, wenn sich herausstellt, dass das andere Team an einer Realität arbeitet, die nicht mit den Annahmen des A-Teams übereinstimmt. Mit anderen Worten: Das A-Team hat einen Versuch gewagt und daneben getroffen.

All diese Nacharbeit rührt daher, dass die API noch nicht bereit war, vom A-Team genutzt zu werden.

Manchmal erzeugt eine nicht erfüllte Abhängigkeit keine Nacharbeit, aber selbst in diesen Fällen kann sie trotzdem zu Verzögerungen führen. Stellen Sie sich vor, das A-Team und das andere Team einigen sich darauf, wie die API funktionieren soll, und alles läuft wie geplant, aber das andere Team braucht einfach länger als erwartet. Infolgedessen konnte das A-Team seine Arbeit bis zum vorgesehenen Fertigstellungstermin nicht richtig testen und musste ihre Frist nach hinten verschieben.

2.10: Versäumnisse bei der Berücksichtigung von Terminplanung und Ressourcenverfügbarkeit

Manchmal können die Probleme so einfach sein wie Terminplanung oder Ressourcenbereitstellung. Einige Arbeitspakete benötigen bestimmte Teammitglieder. Wenn dieses Teammitglied in ein paar Tagen in den Urlaub fährt, ist es wahrscheinlich nicht der richtige Zeitpunkt, um mit dem PBI zu beginnen, das ohne seine Beteiligung nicht abgeschlossen werden kann.

> Wir hören häufig, dass es nicht so sein sollte, aber oft ist es das trotzdem. “Fungible people” ist eine Wunschvorstellung.

Dasselbe gilt für nicht-menschliche Ressourcen. Wenn Sie Serverressourcen für einen Lasttest benötigen, sollten Sie sich besser vergewissern, dass diese Ressourcen auch tatsächlich verfügbar sind, bevor Sie mit der Arbeitseinheit des Lasttests beginnen. Andernfalls sind erhebliche Verzögerungen der beste Fall, und Sie werden wahrscheinlich andere Teams/Mitarbeiter stören, während Sie versuchen, hastig die benötigten Ressourcen bereitzustellen.

Eine weitere Art des Scheiterns bei Fehlstarts ist, wenn einem Team die erforderlichen Fähigkeiten zur Durchführung der Arbeit fehlen. Manchmal ist das eine interne Angelegenheit – etwa wenn ein Teammitglied eine Schulung für ein neues System benötigt oder Recherchen zu einer neuen API durchführen muss. In anderen Fällen ist es ein Planungsproblem, beispielsweise wenn Sie einen UX- oder Datenbankexperten aus einem Pool von Fachkräften ausleihen müssen. Es könnte sogar ein Einstellungsproblem sein, bei dem das Team einen Experten benötigt und ohne diesen bestimmte Arten von Arbeit nicht effektiv ausführen kann.

2.11: Auswirkungen anderer Arten von Fehlstarts

Wir haben Teams erlebt, die sich verpflichteten, Arbeitseinheiten innerhalb von Sprints abzuschließen und die Programmierung relativ schnell durchführten, aber dennoch die Tests nicht abschließen konnten. Dies mag an sich nicht überraschend sein, aber der Grund ist ungewöhnlich: Das Testteam benötigte etwas (wie Testdaten), das es vor Sprintbeginn nicht gesammelt hatte, und die Beschaffung dieser Daten erwies sich als schwieriger oder zeitaufwändiger als erwartet.

Infolgedessen mussten die Arbeitseinheiten in den nächsten Sprint übertragen werden, einfach weil das Team nicht sichergestellt hatte, dass sie wirklich bereit waren, diese innerhalb der zugewiesenen Zeit abzuschließen, bevor sie begannen.

Teams beginnen manchmal mit der Implementierung von Arbeitseinheiten, wenn sie noch offene Fragen haben. Tatsächlich scheinen viele Menschen zu glauben, dass sie dadurch „agiler“ werden.

Dies kann zu enormen Nacharbeiten, Überraschungen oder Verzögerungen führen. Wenn die Antwort auf die offene Frage eine getroffene Annahme verletzt, muss die gesamte auf dieser Annahme basierende Arbeit überarbeitet werden. Wenn die offene Frage bis zum geplanten Abschluss der Arbeitseinheit nicht beantwortet wird, muss die Einheit entweder geschlossen werden, obwohl sie möglicherweise nicht fertig ist, oder offen bleiben, bis die Frage beantwortet ist.

Möglicherweise hat das Team eine interne Abhängigkeit – einen Fehler, der behoben werden muss, eine Vorgängeraufgabe, die abgeschlossen werden muss, und so weiter. Wenn dies nicht richtig nachverfolgt wird, kann es zu denselben Problemen führen wie eine nicht erfüllte externe Abhängigkeit, mit der zusätzlichen Versuchung, den Kontext zu wechseln und es zu beheben.

2.12: Kumulative Kosten

Natürlich verursachen solche Probleme Verzögerungen, Nacharbeit und enttäuschte Erwartungen, aber damit enden die negativen Auswirkungen nicht.

Zusätzlich zur Verschwendung durch Nacharbeit geraten Projekte dadurch meist in Verzug. Wenn Teams hektisch versuchen, Backlog-Einträge abzuschließen und nie wirklich klar ist, was es braucht, um echte Fortschritte zu erzielen, bleiben die Dinge, die tatsächlich erledigt werden *müssen*, oft auf der Strecke.

Häufig, wenn auch nicht immer, führt dies zu erhöhtem Lieferdruck. Wenn Projekte immer weiter in Terminverzug geraten, versucht das obere Management möglicherweise, sie wieder auf Kurs zu bringen, indem es die Mitarbeiter auffordert, schneller zu arbeiten. Das bedeutet unweigerlich längere Arbeitszeiten.

Dies wiederum führt tendenziell zur Erosion des Vertrauens und vergiftet die Unternehmenskultur. Beziehungen, die kollaborativ sein sollten, werden zu Konfrontationen. Menschen, die zusammenarbeiten sollten, um die besten und schnellsten Lösungen zu finden, verwenden ihre Energie darauf nachzuweisen, dass es nicht ihre Schuld war, wenn etwas schief geht.

In der überstürzten Jagd nach Features und abgeschlossenen Arbeitseinheiten ertappen sich Teams oft dabei, wie sie Abkürzungen nehmen. Das bedeutet in Wirklichkeit, dass sie Qualität (besonders Codequalität) opfern. Dies wiederum bedeutet, dass sie zukünftige Produktivität gegen die Illusion von Fortschritt in der Gegenwart eintauschen.

Wenn die Arbeitsbedingungen zunehmend unangenehmer werden, beginnen Schlüsseltalente sich zu distanzieren oder suchen sogar nach anderen Möglichkeiten.

Organisationen, die sich so verhalten, „verbrennen ihre Zukunft“ gewissermaßen in mehrfacher Hinsicht. Die Codebasis wird immer schwerer zu warten, und die Menschen, die sie hätten warten sollen, werden alle vertrieben.

Wenn es daran eine positive Seite gibt, ist sie für uns nicht erkennbar.

Kapitel 3: Einführung in Requirements Maturation Flow (RMF)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

3.1: Was RMF nicht ist

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

3.2: Was RMF ist

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

3.3: Inkrementelle Einführung wird unterstützt und empfohlen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

3.4: RMF 1

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

3.5: RMF 2

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

3.6: RMF 3

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 4: Ist es Agil?

4.1: „Individuen und Interaktionen“, „Funktionierende Software“

4.2: Zusammenarbeit mit dem Kunden

4.3: Auf Veränderungen reagieren

4.4: Transparenz

4.5: Passt zum Prozess, konsistent mit Agile

Teil II: Raum für Bereitschaft schaffen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 5: Die erste Erweiterung

5.1: Vorbereitungsarbeit ist *Arbeit*

5.2: Vorbereitungsarbeit zur Normalität machen

5.3: Ein aufschlussreicher Vorfall

5.4: Wechselseitige Wirkung

5.5: Die Funktion von RMF 1

Kapitel 6: Warum Machen Menschen Das Nicht?

6.1: Vorbereitungsarbeit als Bürger zweiter Klasse

6.2: Eine Allergie gegen nicht-produktive Arbeit

6.3: Also wurde es vergraben

6.4: Der Einfluss des Projektmanagements

6.5: Das Muster

6.6: Projekte und Schätzungen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

6.7: Wie die Nicht-Schätzungs-Schätzungen die Vorbereitungsarbeit beeinflussen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

6.8: Geschwindigkeit messen, nicht Velocity

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

6.9: Schlechte Messungen, schlechte Ergebnisse

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

6.10: Wo die Schuld nicht liegt

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 7: Explizite Vorbereitungsarbeit (RMF 1)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

7.1: Integration in das Synapse Framework™

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

7.2: Anatomie von RMF 1

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

7.3: Verhaltensweise: Kapazität für Zusammenarbeit reservieren

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

7.4: Artefakt: Die Vorbereitungs-Arbeitseinheit

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

7.5: Aktivität: Das Kollaborationstreffen

7.6: Verhaltensweise: Zusammenarbeit fortsetzen, bis gemeinsames Verständnis erreicht ist

7.7: Verhalten: Immer das gemeinsame Verständnis bestätigen

7.8: Wie RMF 1 den Arbeitsablauf verändert

Kapitel 8: Auswirkungen von RMF 1

8.1: Leben vor RMF 1

8.2: Vorher: Aufgewendete Zeit für das Verstehen

8.3: Danach: Aufgewendete Zeit für das Verstehen

8.4: Das Leben nach der Einführung von RMF 1

8.5: Grundlegend

Kapitel 9: RMF 1 in die Praxis umsetzen

9.1: Schulung

9.2: Mindestanforderungen nach Teamtyp

9.3: Zustimmung

9.4: Vorbereitung

9.5: Pilotprojekt

9.6: Einführung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

9.7: Nachverfolgung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

9.8: Erfolg beanspruchen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

9.9: Wachsam bleiben

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

9.10: Was ist mit dem „Wie“?

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

9.11: Zeit, es in die Tat umzusetzen!

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Teil III: Kontrollierte Arbeitsabnahme

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 10: Der nächste Bedarf

10.1: Interpretationsspielraum

10.2: Einschränkung des Interpretationsspielraums

10.3: Eine dritte Option: Kein „Spielraum“

10.4: Mögliche Auswirkungen auf den Abschluss

10.5: Mögliche Auswirkungen auf die Ausführung

10.6: Vorgeschlagene Alternative: Keinen Raum für Fehlinterpretationen lassen

10.7: Vorteile

10.8: Über die Befürchtungen einer Analyselähmung

10.9: Das nächste Erfordernis: Maßgeschneiderte Definitions of Done

Kapitel 11: Was Menschen üblicherweise tun

11.1: Wenn es so großartig ist, warum machen es die Leute dann nicht?

11.2: Die College-zu-Coaching Pipeline

11.3: Coaching-Überflutung

11.4: Eine Art, wie Menschen mit DoD umgehen: Gar nicht

11.5: Nur Akzeptanzkriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

11.6: Globale Definition of Done

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

11.7: Keine Durchsetzungskraft

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

11.8: Zusammenfassung: Der Begriff „DoD“ ist häufiger anzutreffen als tatsächliche Definitions of Done

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 12: Definition of Done definieren

12.1: Über ein einzelnes Arbeitselement

12.2: Fertigstellung

12.3: Präzision

12.4: Struktur einer DoD

12.5: Spezifikationen

12.6: Engineering-Ausgangskriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

12.7: Product-Eingangskriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

12.8: Mehrere Teile, Ein Gate

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

12.9: Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

12.10: Abbildung auf Ihren Prozess

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

12.11: Zusammenfassung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 13: Maßgeschneiderte Definition of Done (RMF 2)

13.1: Prinzip: Jede Arbeitseinheit ist einzigartig

13.2: Verhaltensweise: Pflegen Sie eine oder mehrere DoD-Vorlagen

13.3: Aktivität: Definition der DoD-Vorlage

13.4: Pflege und Verbesserung der DoD-Vorlage

13.5: Mehrere DoD-Vorlagen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

13.6: Verhalten: Vorlagen als Ausgangspunkte für Definitions of Done verwenden

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

13.7: Verhalten: Maßgeschneiderten Definitions of Done zustimmen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

13.8: Aktivität: Definition einer Definition of Done für ein Arbeitselement

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

13.9: Eine weitere Erweiterung des Arbeitsablaufs

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

13.10: Verhalten: DoD vor Beginn der Implementierung zur Reife bringen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

13.11: Aktivität: Offline-Analyse zur Weiterentwicklung der DoD eines PBIs

13.12: Weiterentwicklung in den Workflow einbinden

13.13: Verhalten: Fortschritt in Arbeitspaketen verfolgen

13.14: Fortschrittsverfolgung hinzufügen

13.15: Verhalten: Arbeit durch Fertigstellung steuern

13.16: Aktivität: Verwendung der DoD zur Bestimmung der Fertigstellung

13.17: Wie die Kontrolle sich einfügt

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

13.18: Zusammenfassend

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 14: Leben mit RMF 1 & 2

14.1: Kosten

14.2: Zeitpläne

14.3: Auswirkungen auf das Implementierungsteam

14.4: Auswirkungen auf den Product Owner

14.5: Auswirkungen auf die Führungsebene

14.6: Zusammenfassung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 15: Installation von RMF 2

15.1: Einbindung der Stakeholder

15.2: Detaillierungsgrad

15.3: Arbeitsvereinbarung

15.4: Anfängliche Arbeit

15.5: Einführung

15.6: Zusammenfassung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Teil IV: Gating-Implementierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 16: Die letzte Anforderung

16.1: Es war die ganze Zeit da

16.2: Die Macht des Timings

16.3: Das Ganze auf den Kopf gestellt

16.4: Risiken & Kosten

16.5: Der Wert des Wartens bis zur Bereitschaft

16.6: Ein zusätzlicher Vorteil

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

16.7: Die Problemstellung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

16.8: Die Notwendigkeit

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

16.9: Zusammenfassung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 17: Hintergrund zur Definition of Ready

17.1: Lean's Arbeitsfreigabe

17.2: Kanbans Spalten-Eingangskriterien

17.3: Scrum und andere agile Prozess-Apokryphen

17.4: Surfen > Programmieren

17.5: Punktlösungen

17.6: Wir sind bereit, uns bereit zu machen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 18: Definition einer Definition of Ready

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

18.1: Zweck

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

18.2: Maßgeschneidert

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

18.3: Aufbau

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

18.4: Beispiel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

18.5: Übereinkunft

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

18.6: Kontrollfunktion

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

18.7: Zusammenfassung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 19: Maßgeschneiderte Definition of Ready (RMF 3)

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.1: Eine weitere Kontrolle im Prozess

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.2: Die Struktur einer Definition of Ready

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.3: Product-Exit-Kriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.4: Engineering-Eingangskriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.5: Kein Problem mit Duplikaten

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.6: Verhalten: Eine oder mehrere DoR-Vorlagen pflegen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.7: Warum eine Definition of Ready-Vorlage haben?

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.8: Aktivität: Definition der DoR-Vorlage

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.9: DoR-Vorlagen im Laufe der Zeit pflegen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.10: Vorlagen sind nur ein Ausgangspunkt

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.11: Verhalten: Maßgeschneiderte Definitions of Ready vereinbaren

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.12: Aktivität: Definition of Ready für ein Arbeitselement definieren

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.13: Verhalten: Elemente vor Beginn der Implementierung bereit machen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.14: Bedingungen außerhalb der Kontrolle des Teams

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.15: Verhalten: Bereitschaft in Arbeitseinträgen nachverfolgen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.16: Verhalten: Arbeit durch Bereitschaft steuern

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.17: Aktivität: Verwendung des DoR zur Bestimmung der Bereitschaft

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

19.18: Zusammenfassend

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Teil V: Synthese

Kapitel 20: Die meisten Deadlines sind nicht wichtig

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

20.1: Echte Deadlines

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

20.2: Willkürliche Deadlines sind nicht wichtig

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

20.3: Fluggesellschaften

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

20.4: Der Ursprung der Technischen Schuld

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

20.5: Marketing- und Vertriebstermine

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

20.6: Projektfristen

20.7: Es gibt einen anderen Weg

20.8: Willkürliche Fristen sind nicht notwendig

20.9: Willkürliche Fristen müssen abgeschafft werden

20.10: Echte Termine bleiben ein Faktor

20.11: *Pis Aller*[1]

[1] Ein letzter Ausweg, ursprünglich aus der französischen Sprache.

20.12: Fazit

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 21: Kompetenz 1: Anforderungsreifungsprozess

21.1: Wie unten, so oben

21.2: Prinzip: Transparenz bei allen notwendigen Arbeiten

21.3: Verhaltensweise: Verantwortung wandert mit der Arbeit

21.4: Verhalten: Sichtbares Nachverfolgen des Anforderungsstatus

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

21.5: Verhalten: Offenlegung aller Arbeiten im Zusammenhang mit Vorbereitung und Implementierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

21.6: Aktivität: Vorbereitungsarbeit

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

21.7: Verhalten: Offenlegung aller zur Fertigstellung eines Arbeitspakets notwendigen Arbeiten

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

21.8: Verhaltensweise: Bereitschaft vor Terminen priorisieren

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

21.9: Fazit

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 22: Wie Arbeit und Informationen in Scrum mit RMF fließen

22.1: Einleitende Hinweise

22.2: Erfassung und Vorbereitung der initialen Anforderung

22.3: Initiierung der Readiness-Arbeit

22.4: Planung und Durchführung der Readiness-Arbeit

22.5: Überprüfung der Readiness-Ergebnisse

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

22.6: Planung und Abschluss der Implementierung

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 23: Die Auswirkungen von RMF

23.1: Der Weg einer Anforderung

23.2: Beispiel für den Informations- und Arbeitsfluss

23.3: Vorher

23.4: Danach

23.5: Vorteile

Kapitel 24: Übergang zu RMF

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

24.1: Adoptionsmuster

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

24.2: Integration in einen Scrum-Workflow

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

24.3: Andere Frameworks

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

24.4: Hauptwiderstand: Readiness Work Items

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

24.5: Die große Veränderung: Die Denkweise

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

24.6: Ratschläge für den Wandel

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

24.7: Fazit

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Kapitel 25: Es liegt an Ihnen

25.1: Zusammenfassung

25.2: Jetzt sind Sie an der Reihe

Teil VI: Ressourcen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Anhang A: Scrum ist nicht das Problem

Was ist Scrum?

Frameworks

Scrum behandelt Projekt- und Arbeitsmanagement

Die Torheit, Scrum als Produktmanagement-Framework zu behandeln

Kein vorgeschriebener Mechanismus zur Weiterentwicklung von Anforderungen

Keine Investition von technischem Fachwissen in die Anforderungsentwicklung

Antimuster

Erweiterung erforderlich

Ergänzungen zu Scrum

Anhang B: Das Synapse Framework™

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Was Synapse abdeckt

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Die drei Beherrschungsgrade

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Wie Synapse eingeführt wird

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Verbindung zweier Frameworks

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Die Struktur des Synapse-Frameworks

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Organisatorische Meisterschaften

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Organisatorische Kompetenzen

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Organisatorische Gewohnheiten

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Struktur des Synapse-Frameworks

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Anatomie einer Praxis

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Die Bedeutung der Reihenfolge

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Der Einfluss von Synapse auf dieses Buch

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Anhang C: Häufige Einwände und Hindernisse bei RMF 1

Häufige Einwände

Häufige Hindernisse

Anhang D: DoD-Ausgangskriterienlisten

Technische Abnahmekriterien - Ausgangsliste

Produkteingangskriterien-Starterliste

Anhang E: DoR Starter-Kriterienlisten

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Produkt-Ausgangskriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Technische Eingangskriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Sprint-Eintrittskriterien

Dieser Inhalt ist in der Leseprobe nicht verfügbar. Das Buch kann auf Leanpub unter https://leanpub.com/ready-de erworben werden.

Index