cegeka

Agile Softwareentwicklung

Integrierte Qualität

Januar 2019

JOHAN LYBAERT

Director Agile Software Factory Cegeka

In einer Welt, in der Software unser Leben entscheidend verbessert, ist ein Clean Code der schnellste Weg, wie wir unseren Kunden Unternehmenswert liefern können. Deshalb entwickeln wir eine modulare Software, auf die man sich problemlos verlassen kann und schnell Nutzen bringt.

Selbstverständlich schreibt sich ein Clean Code nicht von selbst. Qualifizierte Fachkräfte müssen hart daran arbeiten. In unserer Software Factory sind circa 700 solcher Fachkräfte tätig. In den letzten 12 Jahren haben wir viele Best Practices im Bereich der agilen Softwareentwicklung übernommen und uns zu eigen gemacht. Wir gestalten einen qualitativ hochwertigen Entwicklungsprozess, indem wir uns nach den Lean-Prinzipien der Reduzierung von Verschwendung richten. Abbildung 1 zeigt, wie Endbenutzer und das Entwicklungsteam zusammenarbeiten und mehrere Best Practices anwenden.

In diesem Dokument erörtern wir diese Best Practices, die allesamt zur Herstellung eines Clean Codes beitragen, d.h. zu einer optimalen Codequalität. Wir teilen unsere Best Practices zum Programmieren und Testen. Theoretisch sind sie alle mehr oder weniger gleich wichtig für die Entwicklung einer qualitativ hochwertigen zukunftsträchtigen Software. Aber in der Praxis entscheiden wir uns bei der Auswahl von Best Practices auf Basis des Kontextes und der Anforderungen des jeweiligen Projekts.

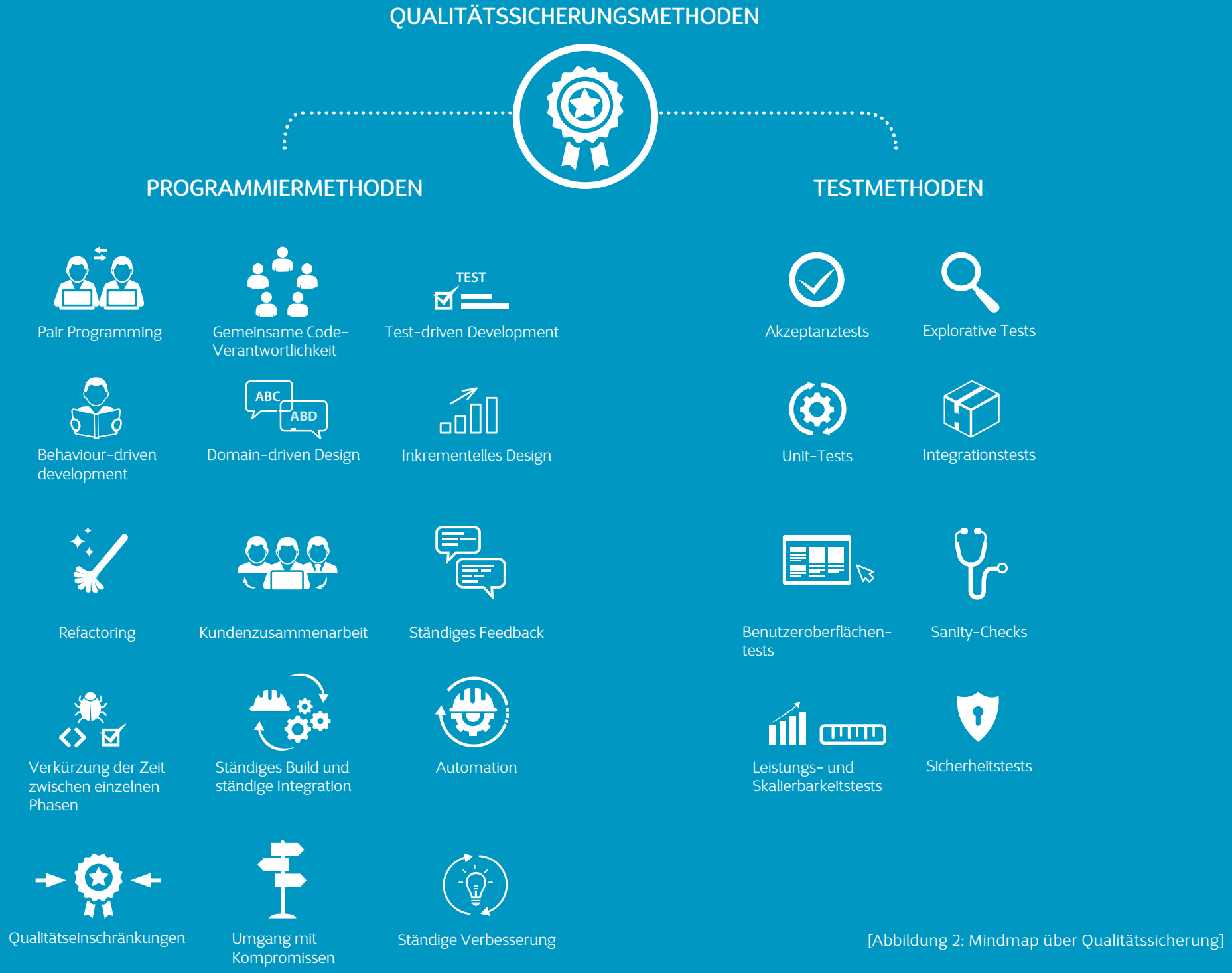
Ich wünsche Ihnen viele Freude beim Lesen.

Johan Lybaert
Director Agile Software Factory Cegeka

[Abbildung 1: Die agile Methode]

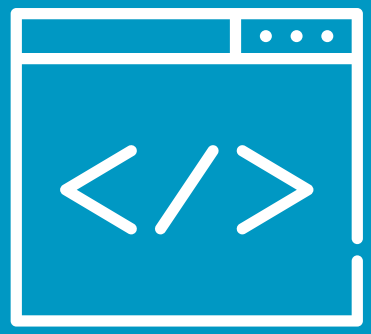
BEST PRACTICES DER QUALITÄTSSICHERUNG

Qualitätssicherungsmethoden sollen verhindern, dass Qualitätsprobleme überhaupt auftreten. Diese Methoden, die in Programmiermethoden und Testmethoden aufgeteilt werden können, sind in Abbildung 2 aufgeführt.



[Abbildung 2: Mindmap über Qualitätssicherung]

Inhalt



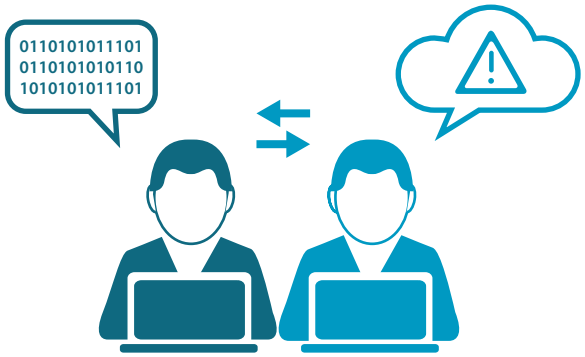
1 Programmiermethoden

PAIR PROGRAMMING

Diese Komponente von Extreme Programming zielt auf die Vermeidung von Qualitätsproblemen ab. Zwei Entwickler arbeiten gleichzeitig gemeinsam an einer Aufgabe, wobei **einer der ‚Fahrer' und der andere der ‚Navigator'** ist. Der Fahrer gibt den Code ein und erklärt dabei, was er tut und warum er es tut. Der Navigator denkt einen Schritt voraus und berücksichtigt mögliche Fallstricke. So sieht er potenzielle Probleme, bevor sie entstehen. Dies führt zu einer besseren Codequalität. Der Fahrer und der Navigator tauschen alle 15 Minuten die Rollen. Deshalb wird diese Methode auch als Ping-Pong- Programmierung bezeichnet.

Durch das Vereinen ihrer Erfahrung kommen Entwickler oft zu Lösungen, die ihnen alleine nicht eingefallen wären. Dies führt häufig zu besserer Produktivität. Darüber hinaus gewährleistet diese Technik Wissenstransfer, weil Best Practices und Codierungsstandards im Team gezielter geteilt werden. Dies führt nicht nur zu einem besser lesbaren Code, sondern trägt außerdem positiv zur Mitarbeiterzufriedenheit bei.

Das Team entscheidet selbständig, wann es in Paaren arbeitet. Wenn ein Code nur von einer Person entwickelt wird, ist eine Codeüberprüfung obligatorisch, bevor eine User-Story als vollständig betrachtet wird.

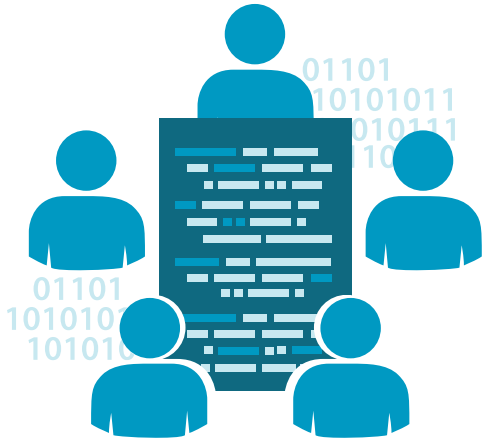


GEMEINSAME CODE-VERANTWORTLICHKEIT

Im Rahmen dieser Methode kann jeder Entwickler jeden Teil eines Codes bearbeiten. In einem agilen Projekt besitzt niemand den Code für sich alleine – er gehört immer dem gesamten Team. Jeder kann damit beginnen, an der nächsten User-Story der Iteration zu arbeiten. Zudem werden Mitarbeiter dazu animiert bzw. es wird sogar von Ihnen erwartet, alle erforderlichen Änderungen am Code vorzunehmen, um die Arbeit des Teams abzuschließen. Ein lesbarer Code ist der Schlüssel zu einer gemeinsamen Code-Verantwortlichkeit. Deshalb müssen Entwickler einen **Code** schreiben, den die Teamkollegen **leicht verstehen** können und der für Mitarbeiter, die den Code zukünftig pflegen, selbsterklärend ist.

Eine gemeinsame Code-Verantwortlichkeit ist einfacher, wenn das Team hinsichtlich der Design- und Codierungsstandards **einheitlich vorgeht.** Das Team muss also einen firmeninternen Stil festlegen, dessen Einhaltung jeder zustimmt. Ein einheitlicher Stil gewährleistet, dass der Code lesbarer ist und dass weniger Zeit in die Reformatierung des Codes auf den persönlichen Geschmack investiert werden muss. Darüber hinaus trägt ein stabiles Team mit geringer Fluktuation positiv zu dieser Methode bei.

In diesem Umfeld gibt es **keine ‚heroischen Einzelkämpfer'** – also Personen, die sich selbst als kritische Ressourcen positionieren und die Einzigen sind, die den Code kennen. Und die deshalb im Falle eines Problems am Wochenende oder im Urlaub in die Firma kommen müssen. Am Ende schaffen sie sich durch die Implementierung ihrer kurzfristigen Lösungen jedoch vermutlich selbst neue Probleme. Eine gemeinsame Code-Verantwortlichkeit macht dies überflüssig.



TEST-DRIVEN DEVELOPMENT (TDD)

Diese Komponente von Extreme Programming zielt auf die Vermeidung von Qualitätsproblemen ab. Hierbei **wird ein Test vor einem Code geschrieben.** Der Analyst schreibt die Testbedingungen für jedes Merkmal kurz, bevor der Code entwickelt wird, auf. Wenn die Entwickler wissen, wie der Code später getestet wird, schreiben sie mit hoher Wahrscheinlichkeit einen Code, der allen Anforderungen entspricht.

In unserer Software Factory gehört es zu den Best Practices, **automatisierte Unit-Tests** für alle Testbedingungen zu schreiben, bevor der Code tatsächlich geschrieben wird. Der Entwickler schreibt den Code, der erforderlich ist, um den Test zu bestehen. TDD ist auch nützlich für den Regressionstest nach einer Softwarewartung, weil man so besser prüfen kann, ob Änderungen des Codes bei der Wartung negative Auswirkungen auf die Software haben.

Test-driven Development steht dafür, nur das zu entwickeln, was wirklich erforderlich ist und zu testen, ob es funktioniert, was eine einfachere Codebasis für Wartung und Änderung zur Folge hat. Weniger Codes bedeuten geringere Komplexität. Und das einfachere Design bedeutet, dass Änderungen und Modifizierungen problemloser durchgeführt werden können. Ein anderer positiver Aspekt ist, dass die Dokumentation ein fester Bestandteil des Codes ist. Deshalb ist sie immer auf dem neuesten Stand.

BEHAVIOUR-DRIVEN DEVELOPMENT (BDD)

Behavior-driven Devlopment bedeutet, dass **Kundenspezifikationen** als Input für die Entwicklung verwendet werden. Die Spezifikationen werden **in der Sprache des Kunde**n geschrieben und schildern das Verhalten, das der Kunde von der Applikation erwartet. Während der Entwicklung der Funktionalität verwandeln die Entwickler die Spezifikationen in automatisierte Tests, indem sie den Test mittels Tools wie JBehave in der Sprache des Kunden mit dem Code verlinken.

Der Unterschied zwischen den Unit-Tests (schriftlich erstellt unter Verwendung von TDD) und automatisierten Integrationstests sowie diesen verhaltensbasierten Entwicklungstests ist, dass sich die ersten beiden auf den Code konzentrieren, während die verhaltensbasierten Entwicklungstests den **Fokus auf das Verhalten legen, das der Kunde** vom Code **erwartet.** Aufgrund der regelmäßigen Durchführung von Tests und der Einbindung in den fortwährenden Integrations- und Bereitstellungsprozess, können wir nicht mehr nur beurteilen, ob der Code sauber und ungebrochen ist (Unit- und Integrationstests), sondern auch, dass er sich so verhält, wie der Kunde es von ihm erwartet. Da diese Tests in der Sprache des Kunden geschrieben werden, sind sie auch eine lebendige Dokumentation des angestrebten Systemverhaltens, das ständig getestet wird.

USER-STORIES

User-Stories beschreiben in einem Satz und in der Sprache des Kunden, einen kleinen Teil der Funktionalität, die zur inkrementellen Entwicklung der Applikation verwendet wird. Sie stellen einen Geschäftswert dar, denn sie erfassen, was ein Benutzer als Teil seiner Aufgabe tut oder tun muss.

Die User-Story ist keine Form der Dokumentation von Geschäftsanforderungen, sondern vielmehr eine Methode, um eine Diskussion zwischen Teammitgliedern zu fördern, die tatsächlich an der Story arbeiten, zusammen mit dem Business, welches das Ergebnis der Story benötigt.

Es ist unerlässlich, dass die User-Story das Ziel des Codes genau definiert und die folgenden Eigenschaften aufweist:

Unabhängigkeit – es ist wichtig, dass Teams die User-Story eigenständig entwickeln, testen und liefern können und dass sie unabhängig bewertet werden kann.

Bestimmbar – jede Story beinhaltet genug Informationen, um von den Entwicklern bewertet werden zu können. Wenn die Story zu ungewiss ist, um bewertet zu werden, kann ein Spike genutzt werden, um die Ungewissheit zu reduzieren und User-Stories zu erstellen, die bestimmbar sind.

Verhandelbar – die User-Story ist keine Vereinbarung über Features, sondern vielmehr ein Platzhalter für die Anforderungen, die besprochen, entwickelt, getestet und akzeptiert werden müssen, und die verhandelbar sind zwischen dem Business und dem Team.

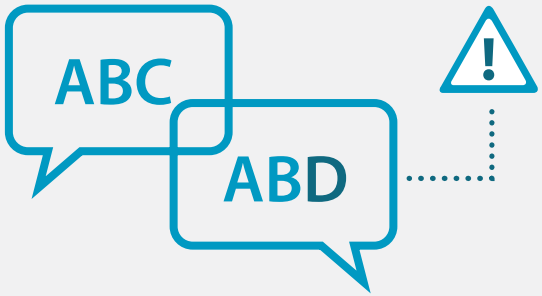
Klein – eine User-Story kann in 1 Iteration implementiert werden.

Wertvoll – jede Story spiegelt den Geschäftswert wider und liefert eine End-to-End-Funktionalität (die sich durch alle Schichten der Architektur zieht).

Überprüfbar – es sind Akzeptanzkriterien und ein Testdesign für die User-Story vereinbart, um Interpretationsprobleme zu minimieren und zu bestätigen, dass die Software funktioniert.

DOMAIN-DRIVEN DESIGN (DDD)

Hierbei handelt es sich um eine Methode der Softwareentwicklung für komplexe Anforderungen, die die Implementierung mit einem Domain-Modell verbindet, in der die Software verwendet wird. In dem Zusammenhang wird eine einheitliche Sprache angewandt, die das Team und der Kunde/die Benutzer miteinander teilen. **Wenn die Wörter, die in der Software verwendet werden, nicht genau zu denen des Business passen, kann dies zu diversen Problemen führen.** Bei einer agilen Entwicklung ist es die Aufgabe des Entwicklers, die Sprache des Benutzers zu sprechen, und nicht die Aufgabe des Benutzers, die technische Sprache zu beherrschen.



INKREMENTELLES DESIGN

Es geht schlichtweg darum, dass man sich die Zeit nimmt, um **das Design der Software in kleinen Schritten zu verbessern.** Die Optimierung des Designs ist Teil der täglichen Arbeit eines Entwicklers. So kann es auf jede User-Story angewandt werden und wird nicht erst für später aufgehoben. Aufgrund dieser Arbeitsweise denken Entwickler über das Design der Software bereits nach während sie Tests schreiben, den Code implementieren, um diese Tests zu bestehen, und bevor sie ihren Code einchecken. Wenn Entwickler ein Design für ihre aktuelle User-Story entwickeln, berücksichtigen sie bei ihren Entscheidungen über das Design zukünftige User-Stories. Zusätzlich stellt sich durch Nachbesserungen eines Designs automatisch eine Verbesserung ein. Wenn ein Design nachgebessert wird, wird es stärker verfeinert und ist formbarer. Es kann eine große Herausforderung sein, direkt von Beginn an den Wechsel vom Standarddesign zu einem inkrementellen Design zu vollziehen. In der Praxis wird nur begrenzte Zeit investiert, um über das Design nachzudenken, ohne eine funktionierende Software herzustellen. Entwickler stellen ihre Ideen unter Beweis, indem sie sie implementieren, auf aktuelle Bedürfnisse abgestimmt entwickeln und ihr Design möglichst einfach halten.

REFACTORING

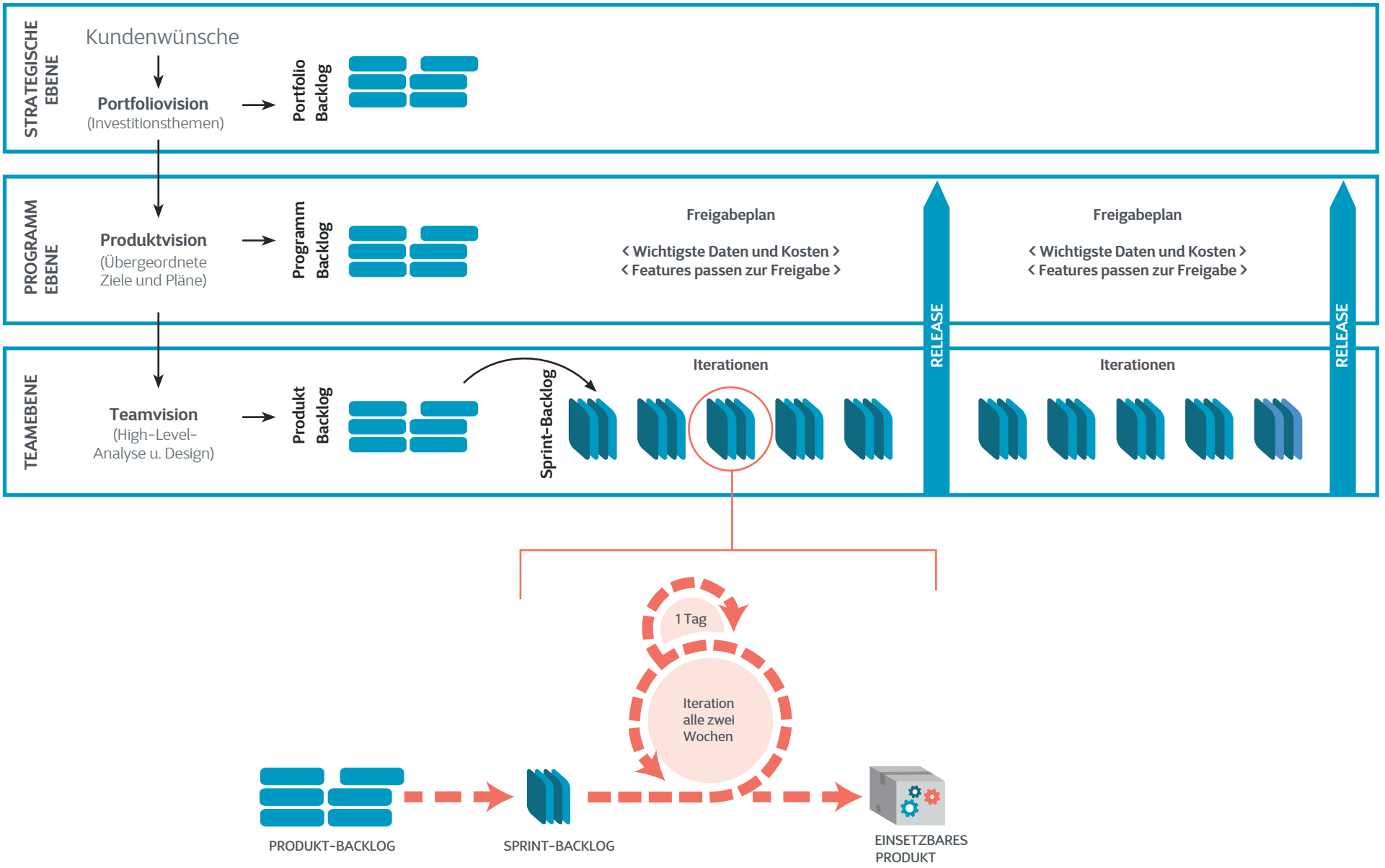
Hierbei handelt es sich um eine Methode, um die **Struktur des Codes systematisch zu verbessern, ohne sein äußeres Verhalten zu beeinflussen.** Dies beinhaltet, dass der Code am richtigen Ort platziert wird, wo andere Entwickler ihn schnell und problemlos finden können, und den Code aufrecht und in Ordnung zu halten. Refactoring bedeutet, dass bekannte Methoden und Variablen genutzt werden, die die Lesbarkeit und Wartungsfreundlichkeit der Codebasis verbessern. Dies ist entscheidend für die **Reduzierung von ‚technischen Schulden'.** Wie Schulden im wahren Leben, müssen auch technische Schulden bezahlt werden, um Fehler aus der Vergangenheit auszugleichen und wieder Fortschritte machen zu können (z.B. im Falle eines Codes, der schlecht entwickelt wurde). Wenn die Schulden jedoch nicht unter Kontrolle gebracht werden, können sie zu einem der größten Risiken für das Projekt werden und die Produktivität deutlich senken. Man kann dies mit einem Kfz-Mechaniker vergleichen, der nie den Boden reinigt, wenn er Öl verschüttet. Wenn er weiter so vorgeht, kann er den Boden so lange schrubben, wie er will. Der Boden wird niemals sauber. Es ist also sinnvoller, den Boden jeden Tag kurz zu wischen.

Refactoring beinhaltet keine ästhetische Organisation des Codes. Es erledigt vielmehr grundlegende Systemarbeiten. Darüber hinaus sollte Refactoring das präzise Funktionieren eines Codes nicht beeinflussen. Infolgedessen erfordert Refactoring eine exzellente Testabdeckung und es ist unauflösbar mit Test-driven Development verknüpft. Refactoring wird den ganzen Tag über kontinuierlich ausgeführt, in kleinen Schritten und mit kleinen Verbesserungen. Das Team darf Refactorings überall in der Codebasis vornehmen: Entwickler benennen Methoden und Variablen um, um die Codebasis lesbarer und verständlich zu machen. Umfangreiches Refactoring ist selten und wird in einer separaten technischen Story gemeistert, die, wie jede andere User-Story, bewertet und priorisiert werden muss. Aggressives Refactoring kurz vor Ende des Projekts bremst Sie nicht aus, sondern beschleunigt Sie vielmehr. Die Ergänzung eines neuen Codes und die Behebung von Fehlern führt auf lange Sicht zu einer besseren, schnelleren und günstigeren Software. Hierdurch bleibt die Wechselkapazität der Applikation erhöht, sodass sogar ältere Applikationen weiter laufen können, ohne die Firma einem Risiko auszusetzen, wie im Dokument „Effects of SIG Quality Model Rating from the Software Improvment Group“ (https://www.sig.eu/files/4114/2563/ 1034/SQM2010-FasterDefectResolution.pdf) beschrieben.

ZUSAMMENARBEIT MIT DEM KUNDEN

Eine starke Einbindung des Kunden – aus geschäftlicher wie aus IT-Sicht – schafft **den erforderlichen Dialog, um bestehende Anforderungen präzise zu definieren und potenzielle Missverständnisse sofort auszuräumen.** Ein gut funktionierendes Team überprüft auch, ob die Implementierung korrekt ist, und zwar durch die Anwendung eines exploratorischen Tests und eines Akzeptanztests. In diesem Prozess ist es wichtig, sich auf die Features zu konzentrieren, die die Firma im Rahmen ihrer Unternehmensvision am dringendsten benötigt, samt der Funktionalität, die für den Benutzer am wertvollsten ist. Wenn ein Benutzer den Mehrwert einer Software erkennt, benötigt er in der Regel weniger funktionale Unterstützung.

Zusammenarbeit mit dem Kunden erfordert ein eingehendes Verständnis des Prozesses sowie einen Kunden, der sich für das Projekt einsetzt. Das Validierungsteam des Kunden sollte sich regelmäßig mit dem Entwicklungsteam zusammensetzen, insbesondere in den einzelnen Testphasen. Hier ergibt sich die Gelegenheit aufzuzeigen, wie die Kundenanforderungen in das Gesamtbild passen. Gleichzeitig führt dies auch zu fundierteren Entscheidungen seitens des Entwicklungsteams. Das Vorgehen ist eine gute Übung für das Validierungsteam, die entsprechenden Informationen visuell in Form von Informationstafeln, Postern und Whiteboards zu präsentieren.

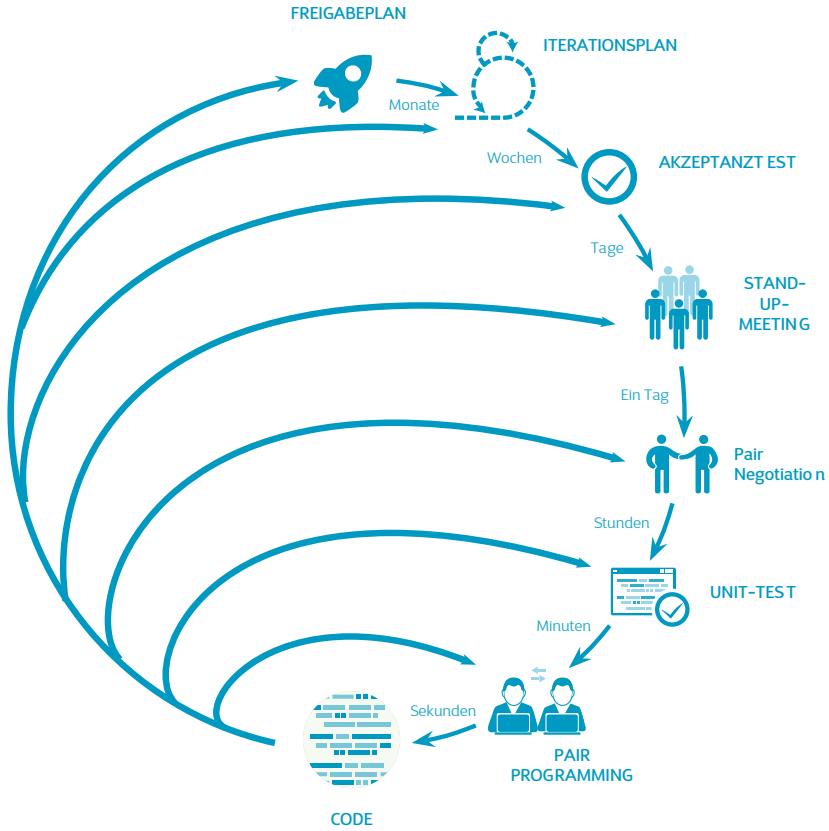


[Abbildung 3: Kundenzusammenarbeit – Starke Einbindung des Kunden gewährleistet die Definition der richtigen Anforderungen]

STÄNDIGES FEEDBACK

Scrum und Extreme Programming sorgen beide für einen qualitativ hochwertigen Prozess – durch das Entwickeln in kleinen inkrementellen Schritten und kurzen Iterationen. Diese agilen Methoden ermöglichen Kunden und Teams, eng zusammenzuarbeiten, mit ständig wechselseitigem Feedback. Dieses Feedback trägt zur Optimierung der Codequalität bei, da der Entwickler das Produkt tagtäglich untersucht und anpasst, um die richtige Qualität zu gewährleisten und, was noch wichtiger ist, das richtige Produkt.

Feedback ermöglicht auch **Prioritäten zu setzen** – falls erforderlich durch einen Vergleich der finanziellen Folgen eines gemeldeten Fehlers mit den Kosten seiner Behebung. Die Methoden von Extreme Programming und Scrum sind absolut komplementär. Deshalb können Teams beide anwenden.



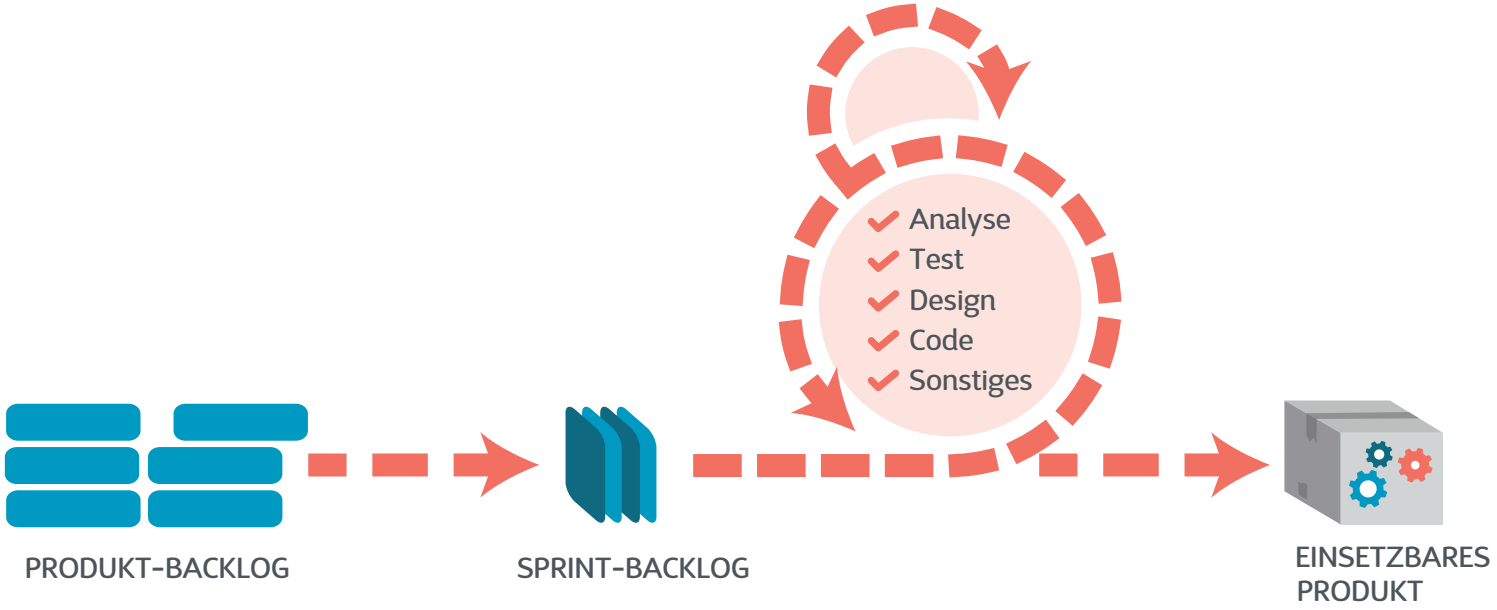
[Abbildung 4: Planungs-/Feedbackschleifen: durch ständiges Feedback auf verschiedenen Ebenen wird Qualität geschaffen]

VERKÜRZEN DER ZEIT ZWISCHEN EINZELNEN PHASEN

Diese wichtige Technik, um im Entwicklungsprozess Qualität zu schaffen, verkürzt die Zeit zwischen Entwicklung, Test und Fehlerbehebung auf ein Minimum. Statt Fehler zu loggen, lösen Entwickler sie lieber sofort. In den meisten Fällen ist das Loggen von Fehlern ohnehin reine Zeitverschwendung. Das Testen des Codes direkt nach der Entwicklung und das Beheben von Fehlern unmittelbar nach ihrer Entdeckung macht das Loggen überflüssig. Vielmehr unterbricht ein langer Zeitraum zwischen dem Schreiben und Testen des Codes sowie dem Beheben von Fehlern die Kontinuität. Dies führt zu Verzögerungen aufgrund von Aufgabenwechseln, Wissenslücken und eines mangelnden Fokus.

Unsere **‚Definition of Done'** ist in diesem Zusammenhang ein nützliches Tool – eine Liste von Anforderungen, die erfüllt werden müssen, bevor eine User-Story als abgeschlossen gilt – was dem Team ermöglicht, eine finale Prüfung unabhängig voneinander durchzuführen.

Es gibt eine positive Korrelation zwischen schneller Fehlerbehebung und der Qualität einer Software, wie im Dokument „Faster Issue Resolution with Higher Technical Quality of Software" beschrieben (https://www.sig.eu/files/4114/2563/1034/SQM2010-FasterDefectResolution.pdf).



[Abbildung 5. Perfektes Timing: Verkürzung der Zeit zwischen Entwicklung, Test und Fehlerbehebung führt zu einer qualitativ hochwertigeren Software.]

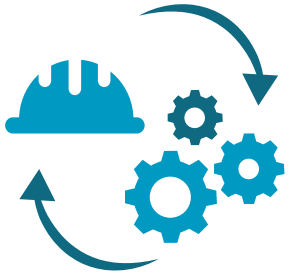
KONTINUIERLICHES BUILD UND KONTINUIERLICHE INTEGRATION

Bei den meisten agilen Methoden ist die Durchführung von regelmäßigen und häufigen Builds zu empfehlen, mindestens täglich, wenn nicht stündlich. „Häufig integrieren und schnell durchfallen!“ Bei Extreme Programming empfiehlt sich eine kontinuierliche Integration, **inklusive eines entwickelten Codes und einer automatisch getesteten Unit,** unmittelbar nach dem Einchecken und der Integration ins gesamte System. Die Verringerung der Lücke zwischen den Builds auf ein Minimum, verkürzt auch die Zeit, die für Integration aufgewendet wird. Bei wichtigen Wasserfall-Projekten können die Phasen der Integration und des Regressionstests sehr lang sein. Dieses Problem kann durch regelmäßige Builds und häufige Integration vermieden werden.

Wir nutzen **Build-Scripts,** die den neuen Code in den vorhandenen Code integrieren, um ein funktionierendes Gesamtsystem zu schaffen. Dies geschieht, bevor der Code an das Versionskontrollsystem übertragen wird, sodass Probleme (im neuen Code selbst oder zwischen dem neuen Code und den externen Komponenten) schnell ermittelt und gelöst werden können. Die Teams nutzen ein Quellcode-Repository und verhalten sich, als wenn sie bereits ab Tag 1 des Projekts in Produktion wären. Entwickler etablieren einen Eincheckprozess und integrieren den Code mehrmals am Tag in kleinen Blöcken. Die kontinuierliche Integration von Änderungen durch die Entwickler im Laufe des Tages beschleunigt und vereinfacht den gesamten Prozess der Softwareentwicklung, -integration und -bereitstellung.

Die kontinuierliche Integration des Codes löst Fehler nicht nur früher, gleichzeitig verringert sie auch die Kosten für Änderungen und sorgt für eine sichere Bereitstellung.

Außerdem gilt: Je mehr Automation, desto besser.



AUTOMATION

Die Teams nutzen ein automatisches Build-System und Entwickler **reparieren einen beschädigten Build immer sofort.** Zusätzlich kann ein automatischer Regressionstest durchgeführt werden, um den Arbeitsaufwand im Rahmen des Aufspürens von potenziellen Qualitätsproblemen zu reduzieren, bevor sie in einer Live-Umgebung auftreten. Gegebenenfalls sollten alle wiederkehrenden Aufgaben möglichst automatisiert werden, um so die Risiken eines menschlichen Versagens zu minimieren.

QUALITÄTSEINSCHRÄNKUNGEN

Nicht-funktionale Anforderungen werden, wenn möglich, auch in Form von User-Stories beschrieben z.B. im Bereich der Benutzerfreundlichkeit („Als Benutzer möchte ich mit der Tab-Taste durch ein Formular navigieren können“), Leistung („Als Benutzer hätte ich gerne eine Antwort innerhalb von x Sekunden nach Absenden eines Formulars“) oder Skalierbarkeit („Als System kann ich 100 Benutzer gleichzeitig ohne Auswirkung auf die Antwortzeiten unterstützen“). Solche Anforderungen werden auch als Qualitätseinschränkungen bezeichnet, weil sie die Grenzen der implementierten Funktionalität aufzeigen.

Eine wichtige Regel ist, dass **Einschränkungen integriert werden, nachdem eine User-Story** innerhalb einer Iteration **abgeschlossen ist.** Denken Sie beispielsweise an das Absenden eines Formulars und die darauf folgende Erwartung einer Antwort innerhalb von 0,2 Sekunden. Die Leistung des Formulars hat vermutlich bei der ersten Iteration keinen Vorrang, aber nachdem diese Qualitätseinschränkung greift, sollten alle folgenden Formulare innerhalb von 0,2 Sekunden reagieren. Der Mechanismus, der für eine Validierung der Leistung erforderlich ist, sollte ebenfalls eingerichtet werden (z.B. automatisierte Leistungstests).

Einige Qualitätsmerkmale sind so offensichtlich und inhärent in unseren Softwareentwicklungsmethoden und Arbeitsweisen, dass wir sie zu jeder Zeit achten möchten, wie **Testabdeckung, Clean Code und saubere Designmethoden.** Diese Qualitätseinschränkungen werden von der ersten User-Story an anerkannt und bilden einen wesentlichen Teil unserer Definition of Done: die **Einschränkungen muss man erfüllen, bevor eine User-Story als abgeschlossen gilt.** Bestimmte Vereinbarungen über die Architektur, die mit dem Kunden geschlossen wurden, können auch auf diese Weise abgewickelt werden, z.B. Namenskonventionen, Serviceorientierung etc.

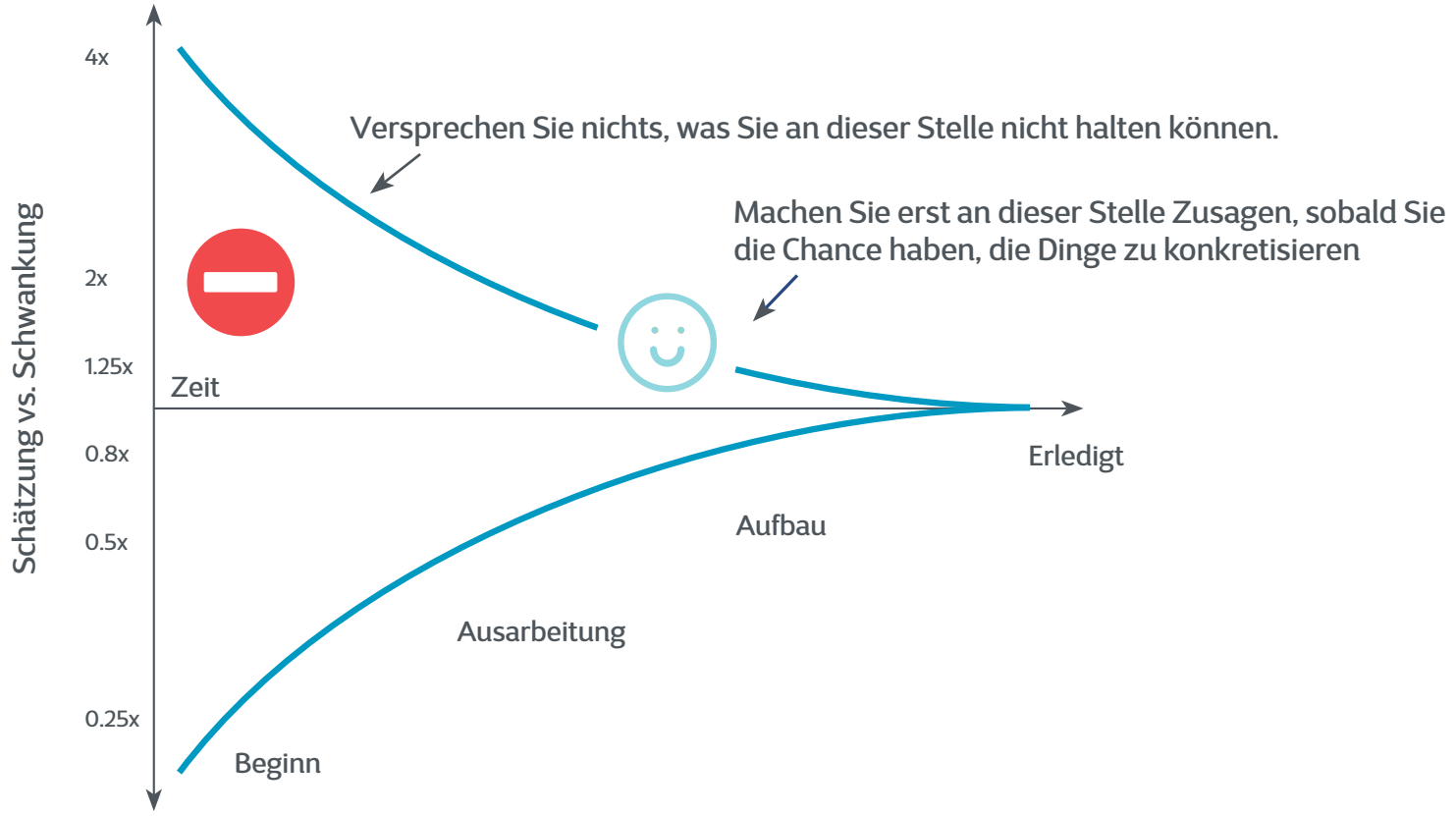
UMGANG MIT KOMPROMISSEN

Berücksichtigen Sie, dass Qualität nur ein Teilaspekt eines Projekts ist – **Zeit, Kosten und Umfang** sollten ebenfalls berücksichtigt werden. Manchmal gibt es wirtschaftliche Gründe dafür andere Faktoren über Qualität zu stellen.

Es können auch **Situationen** entstehen, **wo es mehr kostet, sich auf Qualität zu konzentrieren als auf die Probleme, die der Entwickler zu verhindern versucht.** Ein Beispiel für agile Methoden, solche Kompromisse in der Theorie zu erkennen, ist die Akzeptanz von Nachbesserungen (Refactorings) aufgrund eines Mangels an detaillierten Spezifikationen und eines kompletten Designs direkt zu Beginn eines Projekts. Bei herkömmlichen Methoden wurden in der Vergangenheit detaillierte funktionale und technische Analysen konzipiert, um Qualität schon früh im Projektzyklus zu verbessern. Aber mit der Zeit haben viele Menschen erkannt, dass diese Methoden kontraproduktiv sind – und dies war die Geburtsstunde der agilen Methoden.

Auch wenn Entwickler wenig komplexe visuelle Features entwickeln, die eine begrenzte Auswirkung haben, ist es vermutlich besser, weniger Zeit in Qualitätssicherung zu investieren, weil das Risiko von anstehenden Qualitätsproblemen gering ist. Und selbst, wenn Probleme auftreten würden, wären ihre Auswirkungen nur gering. Selbstverständlich handelt es sich hierbei um eine Grundsatzentscheidung und es ist sicherlich schwer zu erkennen, wo man die Grenze ziehen sollte.

Um **an den richtigen Features zur richtigen Zeit zu arbeiten,** sollte das Team den Produkt-Backlog priorisieren. Hierdurch kann es sich auf die Features konzentrieren, die das Business am dringendsten benötigt. Der beste Ansatz ist, wenn das Team nach einem festen Zeit- und Materialbudget arbeitet. Dies gibt ihm die Flexibilität, effizient auf jegliche Prioritäten zu reagieren, die der Kunden setzt. Beim Festlegen von Prioritäten können Sie Risiken minimieren, indem Sie die Ausführung von komplexen User-Stories mit technischer Ungewissheit zu Beginn des Projekts planen.



[Abbildung 6. Realitätstest: Die Genauigkeit von Schätzungen hängt von der Projektphase ab.]

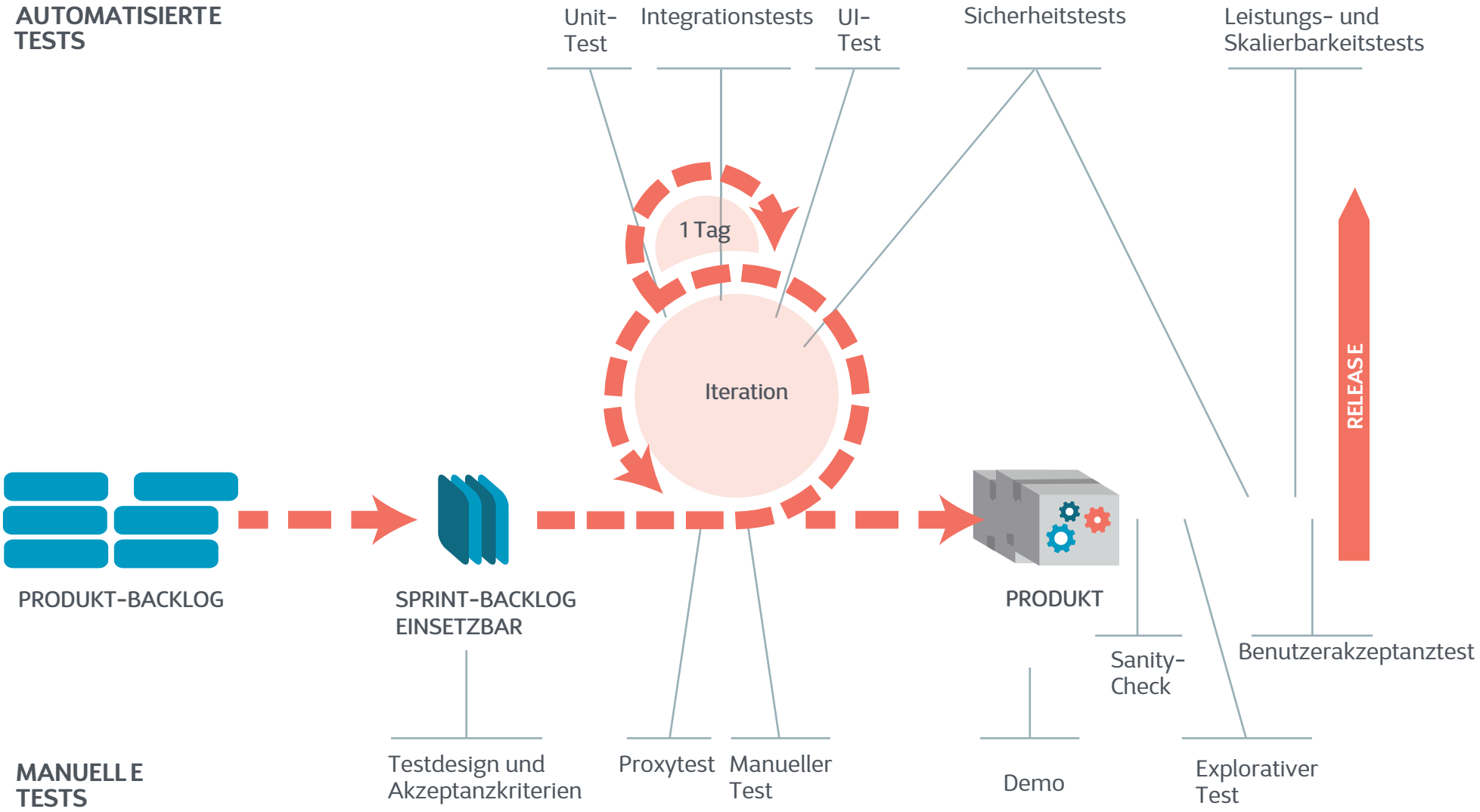
KONTINUIERLICHE VERBESSERUNG

Retrospektiven (Feedback, Untersuchungen und Anpassungen) nach Sprints, Demos, Codeüberprüfungen, ausgiebigen Tests, Anwendung ausgereifter Projektmanagementmethoden, aktivem Coaching etc. **helfen Menschen, ihre Arbeitsweisen kontinuierlich zu optimieren,** sowohl innerhalb eines Teams als auch zwischen verschiedenen Teams. Neben bestimmten Projekten bieten wir unzählige Möglichkeiten, unseren Mitarbeitern dabei zu helfen, neue Erkenntnisse zu sammeln und ihr Wissen zu erweitern, z.B. durch interne Wissen-Sessions, regelmäßige Trainings, Lesekreise, Kompetenzzentren.

2

Testmethoden

Neben Qualitätssicherungsmethoden führen wir eine Reihe von systematischen Tests durch, um mögliche Fehler in einer frühen Entwicklungsphase zu entdecken.



[Abbildung 7. Systematischer Test: erkennt schnell alle Fehler.]

AKZEPTANZTESTS:

Jede User-Story basiert auf einer Reihe von **objektiven Kriterien,** die verwendet werden können, um zu prüfen, ob die Funktionalität richtig implementiert wurde. Der Kunde definiert die Akzeptanzkriterien, die von den Kunden-Proxys dokumentiert sowie übersichtlich zur User-Story hinzugefügt werden. Entwickler nutzen Akzeptanzkriterien, um ein Produkt zu entwickeln. Kunden-Proxys und der Kunde nutzen sie in finalen Akzeptanztests. Einige der Akzeptanztests können sogar automatisiert werden, sodass die Akzeptanzkriterien während Änderungen an der Software kontinuierlich geprüft werden können. Diese Automation wird anhand von Tools durchgeführt, die es ermöglichen, Testszenarios in der Sprache der Benutzer zu schreiben und mittels einem Wiki auszuführen. Kunden-Proxys und Entwickler arbeiten gemeinsam an der Automatisierung von Szenarien. Eine Serie von Akzeptanztestszenarien basierend auf einer Serie von User-Stories (Features) wird als Systemtest bezeichnet.

EXPLORATIVE TESTS:

Ein explorativer Test wird ohne vorab festgelegte Testszenarien ausgeführt. Er wird nach jeder Iteration **vom Kunden durchgeführt.** Der Tester erkennt stufenweise, wie die Applikation funktioniert, indem er verschiedene Eingaben und Navigationspfade ausprobiert. Diese Testweise ähnelt der Art und Weise, wie ein Endbenutzer später mit der Applikation arbeitet. Ein explorativer Test deckt häufig relevantere Probleme auf, als wenn sich nur strikt an definierte Testszenarien gehalten wird.

UNIT-TESTS:
Hierbei handelt es sich um eine a**utomatisierte Methode, um separate funktionale Module oder Units zu testen.** Unit-Tests ziehen sich durch die gesamte Applikation, testen alles von der Geschäftslogik der Applikation bis hin zur Datenbank. Diese Tests sind quasi unabhängig vom restlichen Code. Wenn eine Unit von einer anderen Unit abhängt, wird ein Mock-Framework verwendet, um eine Proxy-Implementierung von bestimmten Objekten zu ermöglichen.

Es gibt viele Vorteile für das Schreiben von Unit-Tests für den Code:

1. Es kommt zu unmittelbarem Feedback, wenn Änderungen am Code vorgenommen werden und ein Unit-Test abbricht

2. Die Fehlersuchzeit ist stark reduziert, denn, wenn ein Unit-Test scheitert, weiß der Entwickler genau, wo das Problem liegt.

3. Die Kosten für einen Regressionstest sind wesentlich geringer, weil ein erneuter manueller Test nicht notwendig ist, wenn ein neuer Teil des Codes freigegeben wird.

4. Bei der Bereitstellung für die Produktion besteht eine größere Sicherheit, weil es eine Reihe von automatisierten Tests für die Validierung des Codes gibt.

INTEGRATIONSTESTS:

Integrationstests sind **automatisierte Tests, die sich auf bestimmte Gruppen von logisch verlinkten Komponenten konzentrieren.** Es gibt zwei Arten von Integrationstests: Blackbox und Whitebox. Blackbox-Integrationstests können nur die Schnittstellen der Komponenten verwenden, um die Funktionalität zu testen. Whitebox-Integrationstests werden in Verbindung mit einer anderen Unit oder Komponente verwendet, die eng verbunden ist, z.B. ein Repository und seine Datenbank oder ein Gateway und sein externes System. Der Whitebox-Integrationstest kann die Datenbank oder das externe System direkt kontaktieren, um die notwendigen Validierungen durchzuführen.

BENUTZEROBERFLÄCHENTESTS:

Benutzeroberflächentests navigieren durch die **grafische Benutzeroberfläche** (GUI) der Applikation, um den Screenflow und den Inhalt (nicht die Geschäftslogik oder das Layout) zu überprüfen. Bei einer webbasierten Applikation können diese Tests unter Verwendung von Tools wie Selenium automatisiert werden.

SANITY TESTS:

Bei Sanity-Checks handelt es sich um Tests, die **nach jeder neuen Softwareinstallation** ausgeführt werden, sowie für jede Umgebung, in der die Software installiert ist. Sie werden angewendet, um die Grundfunktionen des integrierten Systems (z.B. Start der Applikation oder Zugriff auf externe Schnittstellen) zu prüfen. Es ist nur sinnvoll, andere Tests auszuführen, wenn diese Tests erfolgreich sind. Sanity-Checks sollten sofortiges Feedback liefern, um Zeit zu sparen. Deshalb sollten sie möglichst automatisiert sein.

PERFORMANCE- UND SKALIERBARKEITSTESTS:

Performance- und Skalierbarkeitstests sind zwei Arten von nicht-funktionalen Tests. Performancetests bestätigen den **effizienten Betrieb** der Applikation, indem sie die Antwortzeiten der GUI oder des Service messen. Sie können automatisiert werden, indem Zeitinformationen bei der Ausführung einer Integration oder bei einem Akzeptanztest bereitgestellt werden. Skalierbarkeitstests untersuchen, wie sich die **Leistung bei einer wachsenden Anzahl an gleichzeitigen Benutzern** entwickelt. Beide Tests erfordern die Verfügbarkeit einer bestimmten Umgebung, die für die nachfolgenden Produktionsumgebungen repräsentativ ist.

SICHERHEITSTESTS:

Ein Sicherheitstest ist, wie jeder andere Test, in unserem agilen Prozess integriert und Teil der täglichen Arbeit eines agilen Teams. Er gehört zur Aufgabenliste für jeden Sprint, um **das Unternehmen vor Sicherheitsbedrohungen zu schützen,** ohne den Freigabezyklus zu bremsen. Gemeinsam mit dem funktionalen Teil einer Story werden auch **Authentifizierung und Autorisierung geprüft.** Diese Tests können automatisiert werden. Im Rahmen weiterer spezieller Sicherheitstests unterstützt ein Sicherheitsspezialist von Cegeka das Team beim Definieren von Tests, beim Einstellen von Tools und beim Testen der Sicherheit einer Applikation.

3

Umsetzung

Was benötigen Sie zur Implementierung agiler Best Practices?

Es dreht sich alles um den Menschen.

Es ist mit Sicherheit für niemanden, der bereits mit Softwareentwicklung zu tun hatte, überraschend: Arbeitet man mit schlechten Ressourcen, hat man zwangsläufig schlechte Ressourcen. Wenn Teams also wie gut gewartete Maschinen funktionieren sollen, sind die richtigen Leute mit der richtigen Mentalität und den richtigen Tools unerlässlich.

Wir teilen unsere Erfahrungen in den folgenden Bereichen:

- Die richtigen Leute ins Boot holen
- Die richtigen Leute halten und ihnen helfen, ihre Fähigkeiten zu weiterzuentwickeln.

UNSER REKRUTIERUNGSGEHEIMNIS, UM DIE RICHTIGEN LEUTE INS BOOT ZU HOLEN

In den letzten 10 Jahren hat Cegeka seine Methoden der agilen Softwareentwicklung stark verfeinert. Dennoch ist es keineswegs leicht, erfahrene Agile Developer, Business Analysten und Projektmanager für unsere Agile Software Factory zu finden.

Wir können nicht einfach Mitarbeiter von unseren Wettbewerbern abwerben. Stattdessen müssen wir Mitarbeiter mit der richtigen Einstellung finden und sie trainieren, disziplinierte Agile Practitioner zu werden.

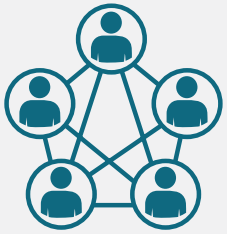
Aufgabenvielfalt dank verschiedener Projekte in unterschiedlichen Branchen

Kompetenzzentren

Wachstumschancen in technischen und funktionalen Funktionen oder als Agile Coach

Werden Sie zu einem (besseren) Agile Practitioner

Kontinuierliches Lernumfeld



Autonomie am Arbeitsplatz und selbstverwaltende Teams

HAUPTGRÜNDE, WARUM MENSCHEN IN DER CEGEKA AGILE SOFTWARE FACTORY ANFANGEN MÖCHTEN

Engagement auf allen Ebenen, Teamwork (auch mit den Mitarbeitern unserer Kunden)

ÜBERZEUGUNGSARBEIT UND ERFAHRUNG

Wir müssen immer noch viel Überzeugungsarbeit leisten, um agile Softwareentwicklung bekannt zu machen. Anders als vor 10 Jahren kennen die meisten Leute in der Softwareentwicklung zwar agiles Projektmanagement und agile Entwicklung. Unsere Erfahrung zeigt aber, dass dies noch lange nicht heißt, dass sie die agilen Prinzipien auch wirklich anwenden.

LEBENSLANGES LERNEN:
Es gibt einige Leute, die von Anfang an vollständig auf agile Prinzipien und Cegekas Best Practices eingestellt sind. Deshalb suchen wir nach Mitarbeitern mit echter Lernbereitschaft – Mitarbeiter, die in der Lage sind, den Status Quo zu hinterfragen und sich selbst zu entwickeln, und auch die Methoden, die wir aktuell anwenden. Unsere Recruiter und Personalleiter suchen immer nach Mitarbeitern, die einer Best Practice nicht einfach blind folgen, sondern die Fähigkeit besitzen, auch den Grund für unsere Arbeitsweisen zu verstehen.

VON DER ÜBERZEUGUNGSARBEIT ZUR ERFAHRUNG:
Um die richtigen Mitarbeiter zu finden, präsentieren wir uns auf Jobmessen und Events. Wir organisieren Besuche in unserer Software Factory für interessierte Kandidaten, sodass diese die vorherrschende Atmosphäre kennenlernen, unsere Teams bei der Arbeit erleben und mit unseren Mitarbeitern sprechen können. Im weiteren Einstellungsverfahren laden wir vielversprechende Kandidaten ein, einen halben oder ganzen Tag gemeinsam mit einem unserer Teammitglieder zu verbringen. So können Kandidaten selbst erfahren, wie es ist für Cegeka zu arbeiten. Letztlich arbeiten Menschen, um Ihre rationalen wie emotionalen Bedürfnisse gleichermaßen zu decken. Bei Cegeka haben wir nichts zu verbergen. Wir bieten einen völlig transparenten Arbeitsplatz.

PEER-TO-PEER-REKRUTIERUNG:
Wir halten es für wichtig, die sozialen Netzwerke unserer Mitarbeiter auszuschöpfen. Deshalb begrüßen wir es, wenn sie uns gute Kandidaten vorschlagen und wir revanchieren uns, wenn ‚ihr Kandidat' eingestellt wird. Selbstverständlich erhalten unsere Mitarbeiter soziale Anerkennung, aber unserer Meinung nach verdienen sie zusätzlich einen kleinen Bonus für ihre Empfehlung. Immerhin setzen sie ihren Ruf aufs Spiel, wenn sie uns eine Empfehlung geben.

SOZIALE MEDIEN:
Der Kampf um Talente hat sich mittlerweile auf die digitale Ebene ausgeweitet. Deshalb nutzen wir im Rahmen von Employer Branding und Rekrutierungsmaßnahmen digitale Marketingmethoden. Wir sprechen Leute mit einer bestimmten Funktion oder aus einem bestimmten Bereich über LinkedIn an. Oder wir posten unsere Stellen auf Twitter und Facebook über unserer Firmenprofile und in spontanen Social-Sharings unserer Mitarbeiter.

REKRUTIERUNG VON HOCHSCHULABSOLVEMTEN

Unsere Recruiter und unsere Geschäftsleitung pflegen gute Kontakte zu Universitäten und Hochschulen. Sie präsentieren sich auf Jobmessen und bei Events, und organisieren Besuche in unserer Software Factory, insbesondere für Studenten, die kurz vor dem Abschluss stehen.

Einstellung von jungen Absolventen

Wir stellen regelmäßig neue Absolventen ein. So können wir eine gesunde Mischung von Teammitgliedern gewährleisten - erfahrene und fortgeschrittene Mitarbeiter sowie Nachwuchskräfte. Dies verschafft uns in eine gute Marktposition. Es ist sehr nützlich, dass wir über verschiedene Qualifikationsebenen verfügen, da nicht jede Aufgabe ein erfahrenes Teammitglied erfordert. Gleichzeitig wäre es unfair, einer Nachwuchskraft die Funktion eines Scrum-Experten zuzuweisen. Wenn man seine Teammitglieder nicht geschickt managt, geht etwas schief.

Passt perfekt.

Durch den Einsatz all dieser Rekrutierungsstrategien, von der Überzeugungsarbeit bis zu Empfehlungen, möchten wir die Erwartungen von Kandidaten und unserer Firma aufeinander abstimmen. Wir möchten die ideale Balance zwischen den Inhalten einer Aufgabe und der Persönlichkeit sowie den Ambitionen eines Kandidaten finden. Wir müssen von Natur aus agil sein - dies ist Teil eines wachstumsorientierten Unternehmens.

UNSERE UNTERNEHMENSKULTUR, UM DIE RICHTIGEN LEUTE AN BORD ZU HALTEN

Unsere Mitarbeiterfluktuation, die um 20 % niedriger ist als im Branchendurchschnitt, zeigt, dass wir die richtigen Leute an Bord halten können. Wir sehen unsere agile Kultur als primären Faktor zur Bindung unserer Mitarbeiter, gefolgt von kontinuierlichem Lernen (inklusive Trainings) und Wissensaustausch.

Lernen ist Teil von Agilität

Lernen ist ein wesentlicher Bestandteil agiler Methoden. Jedes neue Projekt ist eine Lernerfahrung, und zwar wie man Teams aufstellt, wie man den Geschäftswert ermittelt, wie man mit dem Kundenunternehmen arbeitet etc. Mitarbeiter sind Teil eines sich selbst organisierenden Teams, was oft eine Zusammenarbeit zwischen Nachwuchskräften und erfahrenen Mitarbeitern bedeutet. Dies hilft uns, schnell das Know-How unserer Nachwuchskräfte zu erweitern. Die Teams organisieren Retrospective Meetings, um sich über Lernaspekte auszutauschen und Verbesserungen der Arbeitsweise zu erkennen.

Eigenentwicklung

Mitarbeiter übernehmen Verantwortung für ihre persönliche Entwicklung. Sie können extern an formalen Trainingsprogrammen teilnehmen, interne Kurse besuchen, Teil einer Innovationsgemeinschaft werden oder Gilden besuchen. Gilden bestehen aus Menschen verschiedener Teams mit der gleichen Funktion, die zusammentreffen, um sich über ihre Lernmethoden und Best Practices auszutauschen. Zweimal im Jahr organisiert Cegeka ein größeres Meeting zum Wissensaustausch, an dem Mitarbeiter aus allen Abteilungen teilnehmen. Die Inhalte des Meetings werden von ‚unten nach oben' entwickelt: Mitarbeiter können Themen für eine Präsentation oder einen Workshop vorschlagen, die sie machen bzw. durchführen möchten. Eine weniger formale Maßnahme sind Lesezirkel, in denen die Lernaspekte von Büchern mit neuen Ideen über Softwareentwicklung oder agiles Projektmanagement geteilt werden. Die Idee dahinter ist recht einfach: Mitarbeiter können pro Monat den Inhalt eines Buches kennenlernen, aber sie müssen nur eins im Jahr selbst lesen und kurz erläutern.

TEILNAHME AN LERNINITIATIVEN

20% Mitarbeiter der Software Factory besuchen jedes Jahr **Kompetenz- und Innovationszentren**

80% nehmen an **Lesezirkeln teil**

30% sind Teil von Gilden

60% besuchen Meetings **zum Wissensaustausch**

Über den Tellerrand hinausblicken

Wir ermutigen Mitarbeiter, an allen möglichen externen Nutzergruppen für bestimmte Technologien (Java, .NET etc.) oder Methoden (z.B. Agile Consortium) teilzunehmen.

Wir laden mehrmals pro Jahr international anerkannte Redner aus verschiedenen Bereichen ein, eine Präsentation zu geben oder einen Workshop durchzuführen. Diese Veranstaltungen stehen mittlerweile auch Kunden und Dritten offen.

Unternehmen 2.0: geteilte Verantwortlichkeit

Diese Initiativen werden zwar angeboten, aber letztlich sind unsere Mitarbeiter für ihr eigenes berufliches Wachstum verantwortlich und niemand wird zur Teilnahme gezwungen. Lernen und Entwicklung sind allerdings ein wesentlicher Bestandteil von Besprechungen über Leistungsbeurteilungen. Cegeka ist also ein gutes Beispiel für ein Unternehmen 2.0: Beide Seiten, der Mitarbeiter und der Arbeitgeber, sind verantwortlich dafür, dass alles gut läuft.

ÜBER CEGEKA

Mehr als 4200 Mitarbeiter in 10 europäischen Ländern sind der Schlüssel zu unserem Erfolg – und zu dem unserer Kunden. Denn Cegeka unterstützt Unternehmen dabei, in der digitalen Welt nicht nur zu überleben, sondern zu wachsen. Mit Hilfe modernster IT-Lösungen, innovativer strategischer Konzepte und einer Herangehensweise, in deren Zentrum immer die Praxis steht, sorgen wir dafür, dass unsere Kunden das erhalten, was sie für das Erreichen ihrer Ziele brauchen: eine zukunftsfähige und effiziente IT, die den entscheidenden Mehrwert liefert. Und einen zuverlässigen Partner an ihrer Seite.

Cegeka Deutschland gehört zur Cegeka Group. Unser Portfolio umfasst die agile Softwareentwicklung und das Coaching zu agilen Methoden ebenso wie Cloud und Managed Services, die Abwicklung von Infrastrukturprojekten und die Vermittlung von IT-Experten. Natürlich stellen wir auch eigene Business-Applikationen wie die FINAS Suite für Finanzdienstleister bereit. So stehen wir mehr als 2.500 Kunden europaweit zur Seite, wenn es darum geht, die Herausforderungen der digitalen Transformation zu meistern. Unsere Teams sind geprägt durch den Ansporn, Innovationen zu schaffen und gemeinsam mit unseren Kunden die bestmöglichen Lösungen für ihre Anforderungen zu entwickeln.

Für weitere Informationen besuchen Sie uns auf www.cegeka.de

HAUPTSITZ:
Putzbrunner Straße 71
81739 München
Deutschland

WEITERE STANDORTE:
Frankfurt a. M. / Neu-Isenburg
Köln
Nürnberg
Oldenburg
Steinfeld

kontakt@cegeka.de
www.cegeka.de
www.twitter.com/cegeka_DE
www.linkedin.com/cegeka-deutschland
www.facebook.com/cegeka.deutschland