DIE ZEITSCHRIFT FÜR DIE MICROSOFT DEVELOPER COMMUNITY

VS1

VisualStudio1.de

66

DE 8,50 EUR
A/CH 9,50 EUR
EU 9,50 EUR

- Lernen lernen
- Steve Carroll – über Visual Studio 2017 und C++
- MS Build 2017
- Mach Fehler - möglichst viele!
- Query Store im SQL Server 2016

(01)41960661085006

MACH FEHLER MÖGLICHST VIELE!

Aus Fehlern lernen und damit die Qualität nachhaltig steigern

Autoren: Dr. VEIKKO KRYPCZYK und OLENA BOCHKOR

Überall liest man über erfolgreiche Methoden zur Fehlervermeidung. Umfangreiche Tests auf allen Ebenen der Softwareentwicklung sollen dazu beitragen, möglichst wenige Fehler zu machen bzw. diese schnell aufzudecken und zu beseitigen. Man kann es auch wie Thomas Alva Edision sehen: „Ich bin nicht 10.000 Mal gescheitert. Ich habe erfolgreich 10.000 Wege gefunden, die nicht funktionieren!“ Dieser Artikel zeigt, dass gescheiterte Projekte und nicht gelungene Vorhaben durchaus einen Sinn haben. Das Scheitern ist die Quelle für vielfältige Lernprozesse.

Für den Einstieg in das Thema möchten wir gern anders beginnen. Gerade aktuell läuft im Fernsehen die Serie „Charité“, welche die Entwicklung des gleichnamigen Krankenhauses und seiner brillanten Forscher und Mediziner, beginnend im 19. Jahrhundert, dokumentiert. Noch heute blicken wir aufrichtig zu den Akteuren. Ihre Erkenntnisse sind der Grundstein dafür, dass viele Krankheiten erfolgreich geheilt werden können. Eines wird durch dieses TV-Event sofort deutlich: Die Weisheit fällt nicht vom Himmel! Versuch und Irrtum gehören untrennbar zusammen. In der Medizin hatten solche Versuche oftmals fatale Folgen für die Betroffenen. Leben

und Gesundheit waren dabei nicht nur einmal gefährdet. Doch wie soll es anders gehen? Wenn wir sofort wüssten, wie das richtige Vorgehen ist, dann gäbe es die gesamte Entwicklung in allen Wissensbereichen nicht. Nun wir sind keine Mediziner, aber die Softwareentwicklung ist auch eine sehr komplexe Disziplin. Ursache-Wirkungszusammenhänge sind selten trivial. Mit anderen Worten: Drehen wir an der einen Stellschraube, dann mag das Ursprungsproblem gelöst sein. Wir können aber nicht garantieren, dass wir soeben an einer anderen Stelle ein neues Problem produziert haben.

Dennoch scheinen Fehler nicht akzeptabel. Selbst wenn ihre Folgen keine direkten negativen Auswirkungen entfalten, wollen wir sie unbedingt vermeiden. Außenstehende sehen in den Fehlern des jeweils anderen eine Schwäche und fachliche Unfähigkeit. Das Management sieht „rot", denn Nachbesserungen, ein Neustart oder die Erarbeitung einer anderen Lösung kosten Zeit und letztendlich auch Geld. Das ist alles richtig, aber auch alternativlos. Aus diesen Erkenntnissen müssen wir eine andere Art und Weise im Umgang mit Fehlern etablieren. Idealerweise lernen wir aus Fehlern. Gelingen solche Lernprozesse, dann können wir die Forderung: „Einen Fehler muss und sollte man nicht zweimal machen" auch gut mit Leben erfüllen.

In diesem Artikel wollen wir uns mit dem Thema speziell für die Softwareentwicklung auseinandersetzen. Auch in unserer Disziplin lässt sich oft aus einem Fehler deutlich mehr lernen, als aus einer schnellen Lösung, welche lediglich den konkreten Missstand heilt. Es gilt eine umfassende Fehlerkultur zu implementieren und daraus das Wissen für künftige Projekte abzuleiten.

Computer sind keine menschlichen Wesen, auch wenn sie sich durch künstliche Intelligenz dem etwas mehr annähern. Trotzdem trauen wir den Maschinen zu, Fehler zu machen. Nicht in allen Situationen können wir uns auf die Kombination von Hard- und Software verlassen. Das Idealbild der verlässlichen IT haben wir im Textkasten: „Die ideale Welt der IT: Verlässlich und ohne Fehler" beschrieben.

DIE IDEALE WELT DER IT: VERLÄSSLICH UND OHNE FEHLER

Jeder von uns kennt die Situation, dass der Computer gelegentlich seinen Dienst versagt. Es kommt vor, dass aus unerfindlichen Gründen die Rechner bzw. Programme abstürzen oder sie nicht wie gewünscht funktionieren. Man spricht von der Verlässlichkeit und Vertrauenswürdigkeit eines Computersystems. Die Verlässlichkeit ist eine Systemeigenschaft, die ausdrückt wie vertrauenswürdig das System ist. Vertrauenswürdig bezieht sich dabei in erster Linie auf das Vertrauen, das die Benutzer an das System haben. Die Verlässlichkeit lässt sich durch folgende Eigenschaften charakterisieren [5]:

- **Verfügbarkeit:** Fähigkeit des Systems, Dienste auf Anforderung zu liefern.
- **Zuverlässigkeit:** Fähigkeit des Systems, Dienste wie spezifiziert zu liefern.
- **Betriebssicherheit:** Fähigkeit des Systems, ohne katastrophale Ausfälle zu laufen.
- **Informationssicherheit:** Fähigkeit des Systems, sich gegen zufällige oder beabsichtigte Angriffe zu schützen.
- **Reparaturfähigkeit:** Möglichkeit der schnellen Wiederherstellbarkeit und Korrigierbarkeit.
- **Wartbarkeit:** Möglichkeit der Anbindung neuer Anforderungen, so dass das bestehende System nicht seine Nutzer verliert.
- **Überlebensfähigkeit:** Darunter wird die Fähigkeit eines Systems verstanden, auch dann verfügbar zu sein, wenn es angegriffen wird und möglicherweise ein Teil des Systems nicht funktionsfähig ist.
- **Fehlertoleranz:** Darunter versteht man, wie weit das System Eingabefehler des Benutzers verhindert bzw. toleriert. Wenn es zu Benutzerfehlern kommt, sollte das System diese möglichst schnell erkennen und entweder eine Korrektur vornehmen oder den Benutzer informieren.

ERRARE HUMANUM EST – IRREN IST MENSCHLICH!

Menschen machen Fehler! Dies ist die Wahrheit des Lebens (Abbildung 1)! In der IT sieht es auch nicht anders aus: Im Durchschnitt sind es 5 bis 10 Fehler pro 100 Anweisungen in einer neu geschriebenen Software. Zum heutigen Zeitpunkt kann man davon ausgehen, dass fehlerfreie Software ab einer gewissen Komplexitätsgrenze praktisch weder erreich- noch nachweisbar ist. Mit steigender Komplexität sinkt die Möglichkeit einen kompletten Überblick über die gesamte Software zu behalten. Selbst teure oder vielfach getestete Software enthält Programmierfehler. Die Frage ist nur, ob die enthaltenen Fehler entdeckt bzw. noch wichtiger, zu einem Problem in der Bedienung werden. Gut brauchbare Programme sind daher auch nicht fehlerfrei, sondern es wird vielmehr davon gesprochen, dass sie stabil laufen und robust sind. Es ist bekannt, dass ein großer Teil der IT-Projekte abgebrochen wird, weil die angestrebten Ziele nicht

Abbildung 1: Fehler – Definitionsversuche.

erreicht wurden. Die Gründe dafür können unterschiedlicher Natur sein: Einerseits kann es sich um fehlerhafte Managemententscheidungen handeln, andererseits kann es an dem Einsatz unausgereifter Technologien liegen.

Wichtig ist, richtig mit den gemachten Fehlern umzugehen. Gemeint ist hier die sogenannte Fehlerkultur. Darunter versteht man die Art und Weise die Fehler wahrzunehmen, diese zu bewerten um anschließend die richtigen Schlüsse zu ziehen. Ein Beispiel: Regelmäßige Reviews und Tests dienen während der Softwareentwicklung dazu, Fehler transparent zu machen, deren Ursachen aufzudecken und somit die möglichen Folgen vorherzusagen. Aus Kostengründen ist natürlich nicht jeder Fehler vollständig zu analysieren. Hinzu kommt noch der Umstand, dass es sich in der Softwareentwicklung nicht lohnt, abgeschlossene Projekte auf enthaltene Fehler zu untersuchen. Wie kommt denn das? Steht diese Aussage nicht im Widerspruch zur oben zitierten Erkenntnis, dass man aus Fehlern lernt? Daran Schuld ist die Eigenart der IT. Softwareentwicklung ist heutzutage hoch dynamisch. Es ist sehr wahrscheinlich, dass beim nächsten Projekt eine neuere Technologie bzw. ein anderes Tool eingesetzt wird. Die Folge: Es können viele Fehler, die eigentlich gleiche oder ähnliche Ursachen haben, in veränderter Umgebung erneut auftreten. Es ist daher äußert wichtig, dass der Begriff Fehlerkultur in der Softwareentwicklung ganzheitlich betrachtet wird. Idealerweise greifen das Projekt- und Risikomanagement, das Qualitätsmanagement, das Innovationsmanagement und das Fehlermanagement nahtlos ineinander.

Eine neue Sichtweise eröffnet eine sogenannte proaktive Fehlerkultur. Was bitte ist das? Diese Vorgehensweise versucht Fehler zu vermeiden bevor diese überhaupt aufgetreten sind. Man sagt dazu auch: Das geht nur, wenn man aus den Fehlern von anderen lernt. Etwas derb, aber genau passend ist das Zitat von Otto von Bismarck: *„Ihr seid alle Idioten zu glauben, aus Eurer Erfahrung etwas lernen zu können, ich ziehe es vor, aus den Fehlern anderer zu lernen, um eigene Fehler zu vermeiden.“* Natürlich hat auch diese Handlungsmaxime ihre Grenzen. In wie weit sie anwendbar und im Unternehmen umsetzbar ist, muss jeder für sich selbst überprüfen [1].

FEHLER

Bevor wir zum Thema „Umgang mit Fehler“ übergehen, versuchen wir kurz zu klären, was eigentlich ein Fehler ist. Was versteht man unter einem Fehler in der IT? Unter Fehler wird *„die Abweichung eines Zustandes oder Vorgangs, bezüglich der zu erfüllenden Aufgaben“* verstanden [2]. Fehler in Programmen werden auch als *Bug* bezeichnet. Sie sind ein unerwünschtes und abweichendes Verhalten. Die Folgen sind vielfältig, u.a.:

- Programmabstürze
- fehlerhafte Ausführung
- mangelnde Leistungsfähigkeit
- zu hoher Ressourcenverbrauch.

In Bezug auf die Projektdurchführung entstehen zu hohe Kosten und die Fertigstellung verzögert sich. Es ist wichtig zwischen den Begriffen Schwachstelle (eng. *fault*), Fehler (eng. *error*) und Ausfall (eng. *failure*) zu unterscheiden [2]:

- Systemschwachstelle: Es handelt sich um ein Merkmal des Softwaresystems, das zu einem Systemfehler führen kann. Dazu gehören zum Beispiel ein nicht korrekt arbeitender Algorithmus oder ein Designfehler.
- Systemfehler: Darunter wird ein fehlerhaftes Systemverhalten verstanden. Hier stimmt das Verhalten des

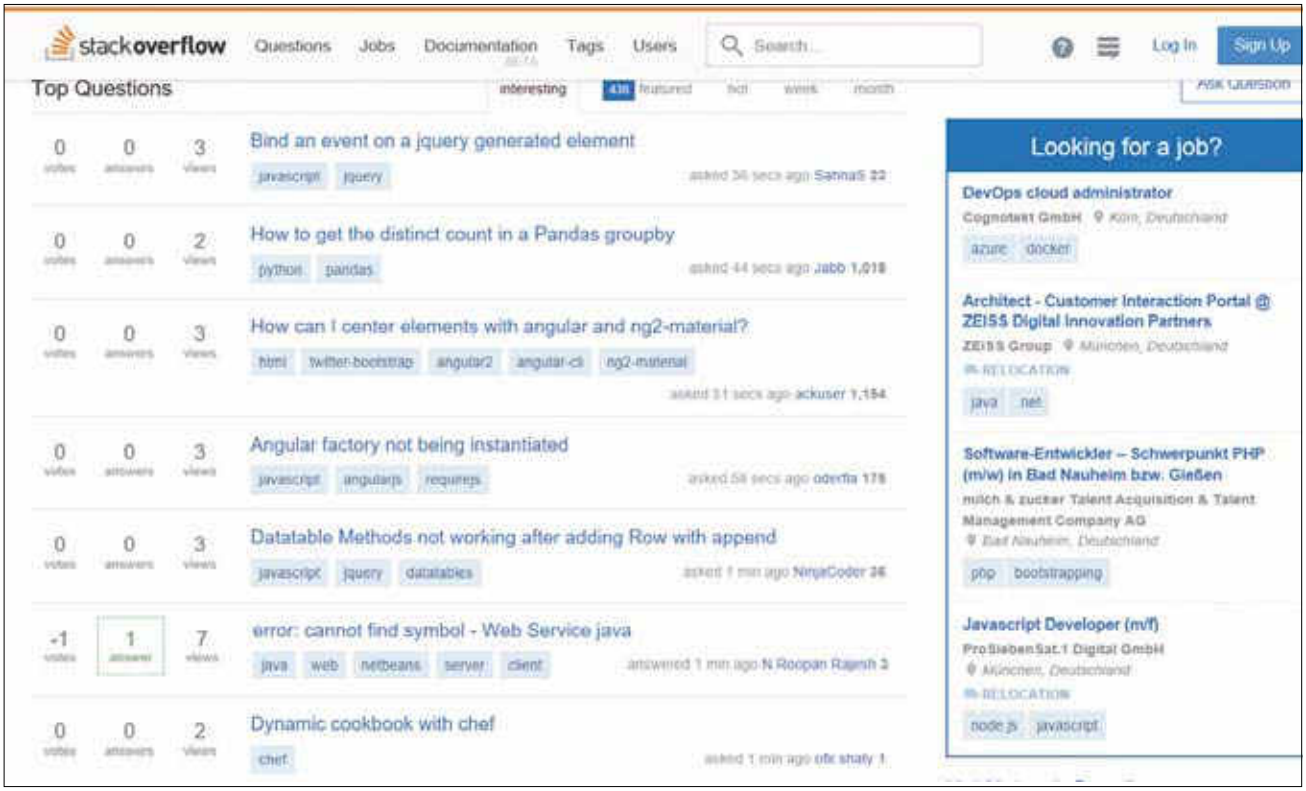

Abbildung 2: Die Plattform *stackoverflow* ist erste Anlaufstelle für Entwickler, wenn es hakt.

Systems nicht mit den Erwartungen der Systembenutzer überein.
- Systemausfall: Hier funktioniert das System nicht oder nicht so, wie es der Benutzer erwartet. Üblicherweise ist der Zeitpunkt weitgehend unbestimmt.
- Fehler durch menschliches Versagen: Darunter versteht man menschliches Verhalten, das zu Fehlern im System führen kann. Das Problem sitzt bekanntermaßen (oft) vor der Tastatur.

Fehler können in allen Phasen der Softwareentwicklung auftreten. Es gilt: Je später ein Fehler entdeckt wird, desto größer ist der Aufwand zur Beseitigung! Das Ziel ist es daher, Fehler so früh wie möglich zu entdecken und ihre wirklichen Ursachen auch herauszufinden. Nimmt man dieses nicht ernst, können sich daraus ernsthafte Probleme entwickeln. Beispielsweise stellt man bereits während des Entwurfs der Software fest, dass das gewählte Framework eine Reihe von Schwachstellen aufweist. Erste Probleme werden bereits deutlich. In diesen Fällen sucht man in dieser frühen Phase bereits nach sogenannten *Workarounds*, um bestimmte Funktionen zu realisieren. Man wird das Gefühl nicht los, hier ggf. den falschen Ansatz gewählt zu haben. Anstatt an dieser Stelle konsequent nochmals einen Schritt zurückzugehen und nach einer anderen grundsätzlichen Lösung Ausschau zu halten, wird einfach weitergemacht. Es wird sich schon eine Lösung finden... Dieses Szenario ist nicht wirklich selten, durchaus realistisch und menschlich. Ein Projektabbruch und Neustart kostet Zeit und Geld. Die bisherigen Arbeiten haben bereits einen beachtlichen Teil des Budgets verbraucht. Also machen wir einfach weiter?! Das Phänomen der *Sunk costs* (versunkene, oft auch als irreversible Kosten bezeichnet) lässt uns weitermachen und wir laufen mit aller Anstrengung gegen die Wand! Damit das nicht passiert, brauchen wir eine positive Fehlerkultur. Es ist das Thema unseres nächsten Abschnittes.

UMGANG MIT FEHLERN

Wie geht man mit den Fehlern in der IT um? Als erstes ist zwischen *Fehlern zur Entwicklungszeit* und *Fehlern zur Laufzeit* eines Systems zu unterscheiden. Man spricht auch von *Fehlervermeidung* während der Entwicklung und Fehlertoleranz während der Laufzeit. Kommen wir zuerst zur Fehlervermeidung: Wie lassen sich die Fehler in der Softwareentwicklung vermeiden? Ist dieses überhaupt möglich? Wir haben bereits erwähnt, dass man Fehler nicht zu 100% vermeiden kann. Softwareentwicklung ist i.d.R. zu komplex. Das Ziel ist aber einer Minimierung der Fehler. Die dafür verfügbaren Methoden sind Ihnen vertraut. Dazu zählen: regelmäßige Reviews, ausführliche Tests und eine umfassende Fehleranalyse. Letztendlich ist der Fehler zu lokalisieren und nach Möglichkeit nachhaltig zu beseitigen.

Betrachten wir die Fehlertoleranz. Darunter versteht man die Fähigkeit eines Systems seine Funktion zu erfüllen, auch mit einer begrenzten Anzahl fehlerhafter Komponenten. Fehlertoleranz zur Laufzeit wird durch Fehlerentdeckung, Fehlerlokalisierung, Fehlermeldung und Fehlerbehandlung erreicht. Nach der Entdeckung eines Fehlers soll dessen genauer Ursprung im System gefunden werden, d.h. die Quelle soll lokalisiert werden um den Fehler später erfolgreich zu behandeln [3]. Nachfolgend betrachten wir eine schrittweise Vorgehensweise mit den Fehlern [4]. Aus Fehlern lernt man! Dieses ist jedoch nur dann der Fall, wenn die Fehler gründlich analysiert und aufgeklärt werden. Das Ziel der Aufklärung besteht darin, die Umstände, unter welchen es zu Fehlern gekommen ist, zu untersuchen. Dieses ist in der Regel nicht einfach und es verursacht hohe Kosten. Hinzu kommt das Argument, dass diese Art von Aufwand sich nicht kurzfristig rechtfertigen lässt. Eine falsche (fehlerhafte ?) Sichtweise des Managements besteht oft darin, dass Entwickler nur dann produktiv sind, wenn sie neue Software produzieren. Das ist eine sehr einfältige Sichtweise. Lernprozesse und eine aktive Fehleranalyse sind genauso wichtige Aufgaben. Werden die modernen agilen Vorgehensmodelle richtig umsetzt, dann erkennt man die Bedeutung dieser Aussage. Regelmäßige Reviews dienen genau dazu. Getroffene Entscheidungen sollen reflektiert werden. Wurden die Ziele erreicht? Gab es besondere Schwierigkeiten bei der Umsetzung? Wo genau lagen die Probleme? Welche Lösung wurde gefunden? Ist die Lösung grundsätzlich geeignet oder handelt es sich vielmehr um einen Workaround?

Nur wenn die Fehleranalyse sorgfältig betrieben wird, hat man die Möglichkeit daraus nützliches Wissen zu generieren, welches später sinnvoll eingesetzt und verteilt werden kann. Dabei ist es wichtig, das Wissen zu teilen. Es wäre traurig, wenn man die Erkenntnisse aus Fehlern nur für sich alleine behält. Damit dieses aktiv geschieht, ist eine positive Fehlerkultur wichtig. Es darf kein Problem sein, einen Fehler zu machen. Vielmehr sollte der Grundsatz gelten, dass ein Fehler eines Mitarbeiters für das gesamte Team genutzt werden kann. Wir alle sollten aus dem Fehler lernen. Fehler werden damit zu wichtigen Schritten auf dem Weg zur Erkenntnis. Am Sinnvollsten ist die Fehleranalyse, wenn von den Ergebnissen möglichst viele Softwareentwickler erfahren, beispielsweise durch Berichte auf Foren oder Beträge in Zeitschriften. Auf diese Weise können wertvolle Erkenntnisse weitergegeben werden. Diese Form der Arbeitsweise ist in Entwicklerkreisen auch schon weit verbreitet. Hat man ein technisches Problem, so sucht man aktiv im Internet nach einer Lösung. Plattformen wie *https://stackoverflow.com/* haben eine riesige Resonanz (Abbildung 2). Im nächsten Schritt wäre es positiv, wenn das nicht anonym im Unternehmen ablaufen muss. Das Problem eines Kollegen mit der zugehörigen Lösung bzw. der Weg dorthin sollten geteilt werden!

Wie sollen Fehler ausgewertet werden? Eine brauchbare Basis für die notwendige Fehleranalyse wird fast überall bereits verwendet. Das Zauberwort lautet Bugtracking. Bugtracker sind Fehlerbearbeitungssysteme für die Softwareentwicklung. Dieses Werkzeug wird eingesetzt, um Programmfehler zu erfassen und zu dokumentieren. Damit lässt sich ziemlich viel erreichen. Voraussetzung ist es, dass alle Beteiligten daran mitwirken. Im ersten Schritt wird die Klassifizierung der entdeckten Fehler verfeinert. Es werden nicht nur die Schwere der Auswirkung und der Aufwand zur Behebung erfasst, sondern auch die Ursachen. Auf diese Weise können im Rahmen

der Testauswertung nicht nur Codeteile als Schwerpunkte identifiziert werden, sondern möglicherweise auch Schwachstellen in der Aufgabenbeschreibung. Technisch kann es zum Beispiel durch eine entsprechende Konfiguration der Trackingsysteme unterstützt werden.

FEHLERMETRIKEN

Eine verzögerte Fehlerentdeckung kann zu einem erheblichen Kostenanstieg führen. Der Grund dafür ist, dass die Fehler sich exponentiell fortpflanzen [1]. Fehlermetriken helfen die Fehler frühzeitig zu entdecken und so das Qualitätsniveau des Produktes konstant zu halten. Ganz abstrakt kann der Prozess der Fehlerentdeckung mit der Fischerei verglichen werden: *Der Fischer (Qualitätssicherer) fängt die Fische (Fehler).* Die Wirksamkeit des Fischers wird danach gemessen, wie viele Fische er fängt bzw. wie viele ihm durchs Netz schlüpfen. Die Wirksamkeit der *Fehlerentdeckung* misst sich daher ebenso anhand von gefundenen Fehlern. Zu einer modernen und guten Fehlerkultur gehört es also, dass man das Auffinden eines Fehlers grundsätzlich positiv bewertet. Das Motto lautet: Wir haben den Fehler gefunden, bevor diesen der Kunde entdeckt und er zu Problemen und Unzufriedenheit führt. In der IT gilt grundsätzlich, dass die Wirksamkeit der Fehlerentdeckung erst im Nachhinein gemessen werden kann. Eine weitere Metrik ist die Fehlereindämmung. Diese misst die Wirksamkeit der eingesetzten Methoden zur Fehlerentdeckung und versucht zu vermeiden, dass Fehler in spätere Phasen übertragen werden. Man spricht von der *Wirksamkeit der Phaseneindämmung (WPE) bzw. Wirksamkeit der Gesamteindämmung (WGFE).* Die Kennzahlen kann man auch berechnen:

$$WPE = \frac{Anzahl\ gefundener\ Fehler, die\ in\ dieser\ Phase\ eingeführt\ wurden}{Gesamtanzahl\ der\ eingeführten\ Fehler, die\ in\ dieser\ und\ auch\ in\ späteren\ Phasen\ gefunden\ wurden} * 100\%$$

$$WPE = \frac{Anzahl\ Fehler\ gefundenen\ vor\ Release}{Gesamtfehler\ gefundenen\ einschließlich\ jener\ nach\ Release} * 100\%$$

Sobald die Fehler gefunden sind, wird ein Versuch unternommen, diese zu korrigieren und in späteren Situationen den gleichen Fehler zu vermeiden (Lerneffekt!).

Ein Ziel ist der Abbau von Fehlern. Auch dazu gibt es eine Metrik und zwar die Kennzahl über die *Wirksamkeit eines Fehlerabbaus (WFA).* Technische Methoden zur Fehlerreduktion sind zum Beispiel Unit-Tests und Codereviews. Die Kennzahl wird wie folgt definiert:

$$WPE = \frac{Anzahl\ Fehler\ gefundene\ und\ entfernt\ durch\ Prozess\ X}{Gesamtzahl\ der\ Fehler\ gegenwärtig\ vorhandenen\ bei\ Ausführung\ von\ Prozess\ X} * 100\%$$

LITERATUR UND LINKS

[1] Wallmüller, E.: Software Quality Engineering. Ein Leitfaden für bessere Softwarequalität, Hanser Verlag, 2011

[2] *https://de.wikipedia.org/wiki/Fehler*

[3] Teufel, C.: Fehlertoleranz, *www-ti.informatik.uni-tuebingen.de/~ruf/seminar0102/Fehlertoleranz.pdf*

[4] Lampe, J.: Aus Fehlern lernen, Business Technology, 04.2016

[5] Sommerwille I.: Software Engineering, Pearson Verlag, 2012

FAZIT UND AUSBLICK

Die Autoren plädieren für eine offene Fehlerkultur. Softwareentwicklung ist eine Mischung aus ingenieurmäßigem Vorgehen und kreativer Arbeit. Dabei spielt Erfahrung und Wissen zwar eine sehr große Rolle, aber in vielen Projekten wird im wahrsten Sinne des Wortes Neuland betreten. Es werden Methoden angewendet, welche man in der bisherigen Situation so nicht verwendet hat. Um aus Fehlern zu lernen, sollte man als Wissensbasis nicht nur die eigenen Fehler haben, sondern auch auf die Fehlversuche der Teammitglieder zurückgreifen können. Wenn Kollege Müller bereits festgestellt hat, dass sich das gewählte Architekturmuster für die betreffenden Anwendungstypen nicht eignet, dann müssen wir den oft leidvollen und frustrationsgeplagten Weg des Scheiterns und Neustartens nicht noch mal gehen. Ein probates Mittel sind regelmäßige Reviews, so wie es zum Beispiel Scrum vorgibt. Tatsächlich sollten wir den gleichen Fehler nicht nochmals machen. Schließich gibt es genügend neue Fehler, so dass sich nachmachen nicht lohnt.