Java™ magazin
Java | Architektur | Software-Innovation

LEGACY NEU DENKEN

Migration, die funktioniert

Coverbild erstellt mit ChatGPT (DALL-E)

entwickler.de
Ausgabe 4.2026

Warum DevOps-Teams Security-Skills integrieren sollten

Mehr Sicherheit durch DevSecOps Engineering

In Zeiten zunehmender digitaler Transformation und wachsender Cyberbedrohungen wird es immer wichtiger, die traditionelle Rollenverteilung in der Softwareentwicklung neu zu denken. Dieser Artikel zeigt, wie sich das Berufsbild vom klassischen Programmierer hin zum modernen DevSecOps Engineer entwickelt hat. Angesichts rasant steigender regulatorischer Anforderungen und immer professionellerer Cyberangriffe wird deutlich, warum Sicherheitsaspekte kultur- und wertebasiert in den gesamten Entwicklungsprozess integriert werden müssen.

von Moritz Tiedje und David Burkhart

Bevor es die modernen Software-Engineers gab, gab es die klassischen Programmierer. Sie sind die Schnittstelle zwischen Menschen und Maschine. Sie verstehen die Technik, beherrschen die Programmiersprache und machen aus Featurewünschen, Akzeptanzkriterien und Jira-Tickets funktionierenden Code. Ihre Verantwortung besteht darin, dass Software fertig wird.

Einen ersten Wendepunkt in diesem klassischen Rollenbild markierte die Softwarekrise Mitte der 1960er-Jahre. Die zunehmende Rechenleistung und Komplexität in der Software überforderte die damals verfügbaren Methoden und Techniken.

Edsger Dijkstra drückte die Problematik in einem Satz prägnant aus: „As the power of available machines grew by a factor of more than a thousand, society's ambition to apply these machines grew in proportion" [1]. In der Folge wurde Software zum ersten Mal teurer als Hardware. Dazu kam, dass immer mehr Softwareprojekte zu spät, zu teuer oder fehlerhaft fertiggestellt wurden, wenn überhaupt. In den Jahren 1968 und 1969 gab es zwei NATO-Software-Engineering-Konferenzen, bei denen der Begriff „Software-Engineering" geboren wurde. Das änderte jedoch noch nichts an der grundsätzlichen Idee der Planbarkeit. Inspiriert von anderen Ingenieursdisziplinen wie Maschinenbau oder Elektrotechnik entstanden Ideen für klare Spezifikationen und strukturierte Prozesse: Software sollte genauso planbar sein wie der Bau einer Brücke.

In der Realität stellte sich jedoch heraus, dass es unmöglich ist, alles vorab zu planen, da sich sowohl Anforderungen als auch technologische Möglichkeiten ändern können, und das häufig und schnell. Daher wandelte sich die Vorstellung des Engineerings weg von der Planbarkeit, hin zur Reaktionsfähigkeit und Anpassbarkeit. Im agilen Manifest wird das ausgedrückt durch „Responding to change over following a plan" [2].

Anfang der 90er-Jahre rückte die technische Exzellenz noch stärker in den Fokus. Software-Engineering entwickelte sich so weit, dass moderne Softwareentwicklung, zum Beispiel mit Scrum, ohne moderne Engineering-Praktiken nicht mehr wettbewerbsfähig ist. Scrum ohne solides Software-Engineering wird als Flaccid Scrum [3] bezeichnet. Dessen Folge ist sinkende innere Qualität. Die Komplexität von Software wächst schon von allein stetig, doch bei sinkender innerer Qualität wächst sie rasant. Das führt zu steigenden Aufwänden und Risiken, bis zu einem kompletten Stillstand der Entwicklung. Selbst wenn dieser schlimmste Fall nicht eintritt, werden die Refactoring- und Qualitätsverbesserungsphasen häufig sehr lang. Auswege bieten Engineering-Praktiken aus dem eXtreme Programming oder der Clean-Code-Bewegung. Komplexität wird hier besser gemanagt. Mit einer hohen Testautomatisierung und kontinuierlicher Integration werden Feedbackzyklen geschaffen, die helfen, Komplexität im Griff zu halten. Clean Code unterstützt bei der inneren Qualität von Anfang an, denn die Qualität schlechten Codes rückwirkend zu verbessern ist schwierig bis unmöglich.

Das Bild des Engineers und seiner Praktiken ist seit den 90ern in konstanter Evolution und konstantem Wandel. 2021 wurde von David Farley „Modern Software Engineering" veröffentlicht [4]. Das Buch traf den

Komplexität von Software wächst stetig, bei sinkender innerer Qualität wächst sie rasant. Das führt zu steigenden Aufwänden und Risiken, bis zu einem kompletten Stillstand der Entwicklung.

Zeitgeist der Software-Engineering-Community sehr gut und ist immer noch aktuell und relevant. Modernes Software-Engineering ist nach Farley eine Lernaktivität. Wir stellen Hypothesen auf und machen Experimente, bauen modulare Architekturen und Infrastruktur, um kontinuierlich integrieren und deployen zu können, messen Verbesserungen, nutzen regelmäßige Feedbackschleifen, entwickeln testgetrieben etc. Die Ansprüche an moderne Software-Engineers sind hoch, trotzdem hat sich dieses Berufsbild bewährt. Software-Engineering ist mittlerweile der Schlüssel zur Bewältigung der ständig wachsenden Komplexität in der Softwarewelt.

Eine kurze Geschichte des DevOps

In der klassischen Softwareentwicklung gibt es zwei separate Rollen: Development und Operations. Development ist verantwortlich dafür, dass Software fertig wird. Operations ist verantwortlich für die Infrastruktur und das Deployment des Codes. Die Software soll stabil laufen und zuverlässig verfügbar sein. Operations verwaltet die Produktionsumgebungen und überwacht die Anwendungen im laufenden Betrieb auf Anomalien und Fehler.

Diese Aufteilung erschien zunächst logisch, erlaubt sie doch den Mitgliedern beider Gruppen, sich auf das zu konzentrieren, was sie am besten können. Damit wurde die komplexe Softwarewelt etwas weniger komplex. In der Praxis jedoch ergaben sich aus dieser Rollenteilung drei große Probleme.

Das erste Problem sind langsame Feedbackzyklen. Ein anonymisiertes Beispiel: Die Developer haben ein großartiges neues Feature gebaut. Die interne Codequalität ist gut, alle Akzeptanzkriterien sind grün, der Product Owner ist zufrieden. Das Inkrement wird an das Operations-Team übergeben. Zwei Wochen später meldet sich das Operations-Team, weil die Performance des großartigen neuen Features miserabel ist. Die Developer arbeiten inzwischen schon an einem ganz anderen Thema. Trotzdem wird mit ein bisschen Chaos und Richtungswechsel eine Story für die Performanceoptimierung ausgearbeitet und kurzfristig eingeplant. Es vergehen etwa zwei Wochen, bis die Developer ein Inkrement liefern, das die Probleme löst. Zwei weitere Wochen später meldet sich Operations erneut: Die Performance ist immer noch schlecht, der Flaschenhals lag an einer anderen Stelle als angenommen. Inzwischen sind bereits sechs Wochen verstrichen, die häufigen Wechsel des Fokus haben auch Wellen in die weitere Releaseplanung anderer Features geschlagen.

Diese Geschichte ist leider kein Einzelfall. Allgemein ist die Beobachtung aus der Praxis, dass die Übergabe von Development zu Operations die Feedbackschleifen langsamer, komplexer und fehleranfälliger macht.

Das zweite Problem ist, dass die geteilte Verantwortung von Development und Operations oft nicht zu den Unternehmenszielen passt. Im agilen Produktmanagement hat sich mehr und mehr eine Outcome-Orientierung [5] anstelle einer Output-Orientierung etabliert. Outcome-Orientierung bedeutet im Produktmanagement, sich bei der Zieldefinition nicht auf einzelne Handlungen zu konzentrieren, sondern auf den Nutzen, den diese bringen sollen. Zum Beispiel wäre für das Problem „Nutzer finden schwer den Weg zur Eingabemaske“ folgendes Ziel output-orientiert: „Der Eingabemaske Button ist rot und blinkt.“ Outcome-orientiert wäre hingegen: „Die Nutzer finden einfach und intuitiv den Weg zur Eingabemaske.“ Sobald ein Output existiert, sind Output-Ziele erreicht. Für Outcome-Ziele gilt das erst, wenn ein Mehrwert für das Unternehmen entstanden ist.

Ein reines Developer-Team kann sich aber keine Outcome-Ziele setzen. Developer können nur den rot blinkenden Button bauen und an das Operations-Team übergeben. Das Operations-Team kann dann beobachten, ob das Rot-blinkender-Button-Experiment erfolgreich war und dem Developer-Team Feedback geben. In der Praxis führt diese geteilte Verantwortung aber oft dazu, dass weder Operations noch Developer die Verantwortung für den Outcome tragen, sondern nur für ihren Output. „Wir bauen die Software“ und „Wir deployen die Software“ sind Ziele, zu denen sich jeweils eine der beiden Gruppen bekennt, „Wir stellen sicher, dass ein Mehrwert entsteht“ jedoch nicht.

Das letzte Problem sind unklare Überschneidungen. Nicht jedes Problem ist eindeutig ein Fall für das Developer-Team oder für Operations. Wenn zum Beispiel Nutzer mit langen Ladezeiten und Timeouts kämpfen müssen, könnte das Problem in einer zu schwachen Serverinfrastruktur oder in schlecht optimiertem Netzwerkcode liegen. Es ist aber oft nicht trivial zu ermitteln, wie die richtige Lösung aussieht. Sollten sowohl Operations als auch Development mit anderen Themen überflutet sein, kann es zu langen Ping-Pong-Gesprächen kommen. Dabei behauptet jedes Team felsenfest, dass das Problem beim jeweils anderen Team liegt.

Wenn das Problem dem falschen Team zugeteilt wird, kann hohe Verschwendung bzw. Waste [6] entstehen. Zum Beispiel könnte das Operations-Team teure Infra-

struktur kaufen, einrichten und betreiben, obwohl einige simple Optimierungen im Code das Problem gelöst hätten. Umgekehrt könnte das Development-Team teuer den Netzwerkcode bis auf die letzte noch so kleine Ineffizienz optimieren, wenn eine Achtzig-Prozent-Pareto-Lösung ausgereicht hätte und der Rest mit vergleichsweise billiger Infrastruktur gelöst worden wäre.

Das am häufigsten genannte Argument gegen DevOps ist die am Anfang dieses Kapitels erwähnte Komplexität. Wir können nicht beliebig viel Komplexität in ein Team legen. Dass die Entwickler in DevOps-Teams sowohl die Komplexität modernen Software-Engineerings als auch von Operations beherrschen sollen, wird als zu viel betrachtet. Man könne den Entwicklern nicht beliebig viel zumuten. Diese Komplexität ist tatsächlich sehr hoch und nicht zu unterschätzen. Dennoch ist es möglich, sie zu bewältigen. Die Vielzahl erfolgreicher DevOps-Teams, die sich seit der ersten Erwähnung von DevOps in den 2000ern etabliert haben, beweist das. Das bestätigen auch der State-of-DevOps-Report der DORA-Organisation [7] und die entsprechenden Metriken für erfolgreiche Teams. Es wäre auch eine Fehlannahme, dass die Anzahl der Probleme in einem Projekt sinkt, weil es kein DevOps gibt. Im Gegenteil sind die Probleme, die aus langsamen Feedbackzyklen, geteilter Verantwortung und unklaren Überschneidungen resultieren, unserer Praxiserfahrung nach verheerend und schwer zu beheben. Tatsächlich ist es sogar einfacher, komplexe Aufgaben in einem Team zu bewältigen.

DevOps ist außerdem für uns als Ingenieure deutlich ansprechender: Auf einen echten Mehrwert hinzuarbeiten und sich diesem mit Hypothesen und Experimenten mehr und mehr anzunähern, ist eine erfüllende, spannende Arbeit, für die wir eine tiefe Leidenschaft entwickelt haben. Wir wollen nicht nur programmieren, wir wollen Dinge verbessern. Es ist deutlich weniger interessant, in einem reinen Developer-Team Output zu produzieren in der Hoffnung, dass sich daraus im großen Ganzen ein wertvoller Outcome ergibt.

DevOps führt hier die Prinzipien der agilen Softwareentwicklung und Praktiken aus eXtreme Programing fort. Technisch werden die Feedbackzyklen automatisierter Tests ergänzt durch Infrastructure as Code, automatisierte Testumgebungen und Monitoring auf Produktionssystemen. Continuous Integration wird weitergetrieben zu Continuous Deployment [8]. Nicht Planbarkeit und perfektes Vorabdesign bringen hier den Mehrwert, sondern schnelle Experimente mit schnellem Feedback und der Möglichkeit, auch schnell zu reagieren. DevOps ist eine Kultur mit Werten, Prinzipien und einer Sammlung von Praktiken und Techniken.

Was bedeutet das konkret für Teams und Teammitglieder? Muss jetzt jeder zum Operations-Experten werden? Genauso wie sich bei Frontend und Backend Spezialisten bilden, gibt es in Full-Stack-Teams auch einige, die beides beherrschen – aber eben nicht zwangsläufig alle. Weiterhin hilft auch die hohe Automatisierung Aufwände und Komplexität gering zu halten. Für mehr als ein Team lohnt sich unter Umständen ein Plattformteam, das es den anderen Teams erleichtern kann, selbstständig zu deployen. Wichtig dabei ist, dass dieses Plattformteam nicht selbst zum Flaschenhals wird. Dienste dieses Teams sollten im Selfservice genutzt werden können [9].

Eine kurze Geschichte des DevSecOps

Klassischerweise gibt es auch die Rolle der Security-Experten. Sie sind dafür verantwortlich, dass das Unternehmen nicht gehackt wird. Dafür wird die Software auf Sicherheitslücken geprüft, Richtlinien werden aufgestellt und geprüft sowie Security-Audits und Penetrationstests durchgeführt. Meistens geschieht das nachgelagert und außerhalb des Entwicklungsteams.

Wie würde es aussehen, wenn man dieselben Prinzipien wie in der agilen Softwareentwicklung und DevOps auf Security anwenden würde? Sicherlich nicht nur als nachgelagerter Prozess in Form eines Audits durch einen externen Spezialisten, sondern, ähnlich wie bei der Qualität, als ständiger Begleiter in der Entwicklung. Damit das gelingen kann, brauchen die Entwickler Entlastung durch Automatisierung und Feedbackmechanismen. Ein Feedbackzyklus, der nur einmal pro Release über ein externes Audit läuft, ist hier zu wenig und zu langsam.

An verschiedenen Stellen des Entwicklungszyklus gibt es modernes Tooling, das Entwicklern helfen kann, Security-Aspekte von Anfang an zu berücksichtigen:

- Ähnlich wie Unit Testing Feedback innerhalb von Sekunden/Minuten geben kann, hilft SAST (Static Application Security Testing) durch statische Analyse des Quellcodes, problematischen Code bezüglich Security schon beim Schreiben oder bei der Integration zu finden und Security-Probleme von Anfang an zu vermeiden.
- Stündlich oder täglich kann mit Hilfe von Tooling im Bereich SCA (Software Composition Analysis) festgestellt werden, ob in der Komposition der Software, also der Zusammenstellung der Software durch verschiedene Bibliotheken, bekannte Security-Issues bestehen. Beispielsweise prüft der OWASP-Dependency-Check [10] die konkreten Versionen der Abhängigkeiten einer Software gegen öffentliche Datenbanken bekannter Schwachstellen (CVEs [11]).
- Nach dem (hoffentlich regelmäßigen) Deployment der Software auf Test- oder auch Produktivsysteme kann mit Hilfe von DAST (Dynamic Application Security Testing) die laufende Anwendung auf Security-Probleme geprüft werden. Ein Beispiel hierfür ist der OWASP Zed Attack Proxy (ZAP) [12].
- Im Rahmen der Planung kann man einen Threat-Modeling-Workshop durchführen. Threat Modeling ist eine strukturierte Vorgehensweise, um mögliche Bedrohungen der Anwendung zu finden und zu be-

werten. Es sollte nicht als einmalige Aktivität verstanden werden, sondern als iterativer Prozess.

Aber auch bei DevSecOps gilt: Die Kultur und die Werte eines Teams sollten nicht darauf ausgerichtet sein, nur ein bestimmtes Tooling anzuwenden, sondern echte Probleme zu lösen und gemeinsam Security von Anfang an im Prozess mitzudenken. So entstand (ähnlich dem agilen Manifest) in der Security-Community das „Threat Modeling Manifest" mit eigenen Werten [13]: „We have come to value:

- A culture of finding and fixing design issues over checkbox compliance.
- People and collaboration over processes, methodologies, and tools.
- A journey of understanding over a security or privacy snapshot.
- Doing threat modeling over talking about it.
- Continuous refinement over a single delivery."

Es geht also darum, ernsthafte Probleme zu lösen, gemeinsam zu lernen und Security (ähnlich wie Qualität) als ständigen Begleiter zu betrachten. DevSecOps ist nicht allein ein technisches Problem oder ein eigenes isoliertes Team, sondern wie DevOps eine Kultur mit Werten, Prinzipien und einer Sammlung von Praktiken und Techniken. Qualität kann in der Softwareentwicklung nur schwer und mit hohen Aufwänden im Nachhinein geliefert werden. Genauso ist es schwer, Security-Aspekte erst im Nachgang zu betrachten. Manchmal liegen die Probleme im grundsätzlichen Design und würden zur Lösung einen fundamentalen Neubau erfordern. Manchmal sind auch bereits erhebliche Sicherheitsmängel im Produktivsystem vorhanden, auf die zu spät reagiert wird.

Bis hier haben wir den Ursprung und die Wertekultur des DevSecOps Engineering beschrieben. Wir wollen uns nun dem aktuellen Zustand der Cyberkriminalität und den Regularien zuwenden, um zu beleuchten, wieso die Nachfrage für DevSecOps Engineering steigt und noch weiter steigen wird.

Die Welt der Cyberkriminalität

Professionalität, Anzahl und Diversität von Cyberangriffen steigen stetig. Die Ursachen sind zahlreiche Enabler und Entwicklungen, die schwer zusammenzufassen sind und sich noch schwerer mit klaren Statistiken, Fakten und Zahlen hinterlegen lassen. Die Dunkelziffer dürfte aus drei Gründen sehr hoch sein: Die Betroffenen geben diese Informationen ungern an die Öffentlichkeit, die Hackergruppen selbst sind keine zuverlässigen Informationsquellen und die Behörden wollen solche Statistiken nicht bereitstellen. Wir versuchen trotzdem, eine grobe Übersicht zu geben.

Es gibt ein Cybercrime-Ökosystem. Der moderne Hacker ist nicht mehr ein Individuum oder eine kleine organisierte Gruppe, wie das in Filmen oder in älteren Nachrichtensendungen dargestellt wird. Der moderne Hacker bewegt sich in einem Netzwerk, in dem es Dienstleister, Zulieferer, Verkäufer und Käufer gibt. Die Motivationen der Akteure sind divers. Es gibt [14]

- privatwirtschaftliche Akteure, die Cyberwaffen entwickeln und verkaufen.
- klassische Hacktivisten, die politische und soziale Ziele verfolgen.
- gewöhnliche Cyberkriminelle, die finanziellen Profit machen wollen.
- Akteure aus dem staatlichen Umfeld, die politische und strategische Ziele verfolgen.

Diese Akteure kommunizieren miteinander, arbeiten sich untereinander zu und tätigen finanzielle Transaktionen untereinander (dabei kommt es auch oft untereinander zu Betrug und Diebstahl in vielen verschiedenen Formen). Zum Beispiel kann ein Business-E-Mail-Compromiser (BEC) E-Mail-Adressen, denen vertraut wird, an einen Initial Access Broker (IAB) verkaufen. Der IAB nutzt diese Adressen, um Zugriff auf geschützte Systeme zu erlangen und diesen an Ransomware-Experten weiterzuverkaufen. Die Ransomware-Experten führen dann einen Erpressungsversuch durch mit Software, die sie wiederum von einem Crime-as-a-Service-(CaaS-) Provider gekauft haben. Dieses Ökosystem wächst stetig [15]:

- Die Digitalisierung in Europa nimmt kontinuierlich zu und damit steigt die Angriffsfläche für Cyberkriminelle. Erwähnenswert ist hier auch, dass eine Menge Infrastruktur, die bisher vor Ort war, aktuell in die Cloud umgezogen wird.
- Das Darkweb ist nach wie vor ein Enabler, der fähigen Akteuren komplette Anonymität erlaubt.
- Kryptowährung ist auch ein Enabler, da sie hohe finanzielle Transaktionen außerhalb jeglicher staatlicher Kontrolle ermöglicht.
- End-to-End-Verschlüsselung in privater Kommunikation ist Standard geworden, selbst für nichttechnikaffine Nutzer.
- Es gibt sichere Zufluchtsorte für Cyberkriminelle. Zum Beispiel müssen Ransomware-Erpresser in Russland keine strafrechtliche Verfolgung fürchten, solange sie nur europäische Unternehmen angreifen.

Die Emergenz von KI und LLMs macht kriminellen Akteuren in vielerlei Hinsicht die Arbeit noch einfacher [15]. Es gibt KIs, zum Beispiel DarkBard, WolfGPT oder WormGPT, die speziell für den Einsatz in Cyberkriminalität trainiert wurden. Unabhängig davon lassen sich auch herkömmliche KIs leicht missbrauchen, um zum Beispiel Schadsoftware ohne großes technisches Know-how schnell zu vibecoden oder die Rolle eines Sicherheitsberaters einzunehmen. Zudem gibt es nun die Möglichkeit, über Deep Fakes in Videos andere Identitäten zu imitieren. Betrugsformen, für die zuvor viele

billige Arbeitskräfte benötigt wurden, lassen sich nun mit Chatbots automatisieren.

All diese Aspekte führen nicht immer direkt zu nennenswerten Angriffen, nähren aber ein wachsendes Ökosystem, indem sie Cyberkriminalität einfacher, billiger und/oder sicherer machen. Das bedeutet nicht, dass alle der hier genannten Entwicklungen „schlecht" sind. Zum Beispiel ist unsere eigene Arbeit durch den Einsatz von KI-Tooling in manchen Aspekten schneller und effizienter geworden. Außerdem sind viele der oben genannten Entwicklungen unvermeidbar, wenn individuelle Privatsphäre und ein Recht auf Anonymität weiterhin hohe Güter in unserer Gesellschaft bleiben sollen. Wir führen diese Aspekte hier nicht auf, weil wir sie generell ablehnen. Wir weisen nur auf die Nachteile hin, die sie mit sich bringen und das kriminelle Ökosystem begünstigen.

In Summe ist ein stetiger Anstieg der Angriffe auf europäische Bürger und Unternehmen sichtbar. Ein erfolgreicher Ransomware-Angriff auf ein kleines oder mittelständiges Unternehmen ist nicht mehr wilde Sci-Fi, sondern Alltag. Wenn ein Unternehmen in der digitalen Welt präsent ist, dann sollte seine Cybersicherheit im Verhältnis zu seinem Umsatz stehen. Sonst ist es unserer Ansicht nach nur eine Frage der Statistik und der Zeit, bis ein Angriff Erfolg hat. Umso wichtiger ist darum das Berufsbild des DevSecOps Engineers. Sie sind kein Ersatz für echte Security-Experten, können aber durch Anwendung vieler kleiner Praktiken und einen bewussten Umgang mit der eigenen Verantwortung maßgeblich zur Sicherheit eines Unternehmens beitragen.

Die Welle der Regularien

Die Behörden reagieren auf die Entwicklungen im Bereich der Cyberkriminalität unter anderem auch mit Regularien. Zuallererst sei hier die Datenschutz-Grundverordnung (DSGVO) genannt, die 2016 beschlossen wurde und seit 2018 anzuwenden ist. Hier geht es auch darum, dass persönliche Daten nicht nur rechtmäßig verwendet werden, sondern auch ausreichend geschützt sein müssen. Wer alte Software mit offenen Sicherheitslücken betreibt und darin personenbezogene Daten verarbeitet, schützt diese Daten nicht ausreichend.

Auf die Datenschutz-Grundverordnung folgte eine Welle weiterer Regulierungen. NIS (2016) und NIS2 (2022) haben den Schutz kritischer Infrastruktur zum Ziel. NIS2 erweitert dabei den Kreis der betroffenen Unternehmen: Neben Energie-, Gesundheits- und Verkehrsinfrastruktur sind nun auch Branchen wie digitale Dienste, Abfallwirtschaft, Lebensmittelbranche oder die chemische Industrie betroffen – und damit auch deren Zulieferer und Dienstleister. Die konkreten Forderungen der Richtlinie betreffen z. B. Incident Management, Risikomanagement und Supply Chain Security. Diese Bereiche können mit DevSecOps sehr gut bedient werden. Für Supply Chain Security verwenden Teams mit DevSecOps-Mindset Tooling aus der Kategorie „Software Composition Analysis (SCA)", um z. B. ihre Abhängigkeiten automatisch aktuell zu halten (wie etwa mit dem Renovate Bot [16]) oder sie auf bekannte Security-Lücken zu prüfen (wie etwa mi dem OWASP Dependency Check [17]).

2019 wurde der Cybersecurity Act beschlossen, der im Wesentlichen einen EU-weit einheitlichen Rahmen für Cybersecurity-Zertifizierungen schuf und die ENISA (European Union Agency for Cybersecurity) als zentrale Instanz etablierte. 2022 folgte der Finanzsektor, für den der Digital Operational Resilience Act (DORA) aufgestellt wurde. Er geht über reine Resilienz hinaus im Sinne von nicht nur „nicht gehackt werden" zu „schnell reagieren können". Wer DevOps lebt und eine hohe Automatisierung vom Quellcode bis ins Zielsystem geschaffen hat, kann schnell reagieren.

Die neueste und umfassendste Regulation, die aktuell nach und nach in Kraft tritt, ist der Cyber Resilience Act von 2024. Er betrifft alle Hersteller von Produkten mit digitalen Elementen und fordert noch nicht dagewesene Standards, zum Beispiel eine Meldepflicht bei Sicherheitsvorfällen oder ein stetiges Vulnerability Management über den gesamten Produktlebenszyklus.

Fazit

Zusammengefasst: Security muss ein integrierter Teil des Entwicklungsprozesses sein („Security is now a feature, not an afterthought" [18]). Qualität umfasst auch Security und kann nur schwer im Nachhinein geliefert werden. Vielmehr sollte man sie von Anfang an berücksichtigen. Gute Security braucht einen hohen Grad an Automatisierung. Die Komplexität heutiger Software übersteigt bei Weitem das, was ein einzelner Mensch im Kopf beherrschen kann. Darum ist es absolut notwendig, automatisierte Feedbackzyklen zu schaffen. Auch die Umsetzung, z. B. die Aktualisierung von Abhängigkeiten auf aktuelle Versionen, kann automatisiert werden. DevSecOps-Know-how muss in den Teams etabliert sein. Das bedeutet nicht, dass jedes Teammitglied ein Security-Experte werden muss. Trotzdem sollte der Security-Skill im Team vorhanden sein. Cyberangriffe sind Realität. Die Bedrohungslage ist hoch, Cyberkriminalität professionalisiert sich mehr und mehr. Die Regulatorik wird zunehmen. Nach der DSGVO, die auch wegen des Datenschutzes zum Teil Security streift (Daten müssen geschützt sein), gibt es NIS, NIS2, DORA, CSA, CRA … Wer DevSecOps bereits lebt, findet sicher einen Weg, diese Richtlinien zu erfüllen, wenn sie nicht schon durch vorhandenes DevSecOps Engineering erfüllt werden.

In diesem Artikel stehen viele Ansätze und Literaturverweise, die helfen können, wenn man sein Wissen in DevSecOps vertiefen möchte. Man kann sich in DevSecOps auch gut mit Learning by Doing professionalisieren. Baut ein kleines Stück Security-Automatisierung in euren Entwicklungsprozess ein oder nehmt euch die OWASP Top Ten [19] und macht eine erste kleine einstündige Threat Modeling Session im Team.

Ein starkes DevSecOps-Team zu etablieren ist, wie so viele andere Dinge in der agilen Softwareentwicklung, nicht etwas, das sofort geschieht, sondern ein iterativ-inkrementeller Prozess in vielen kleinen Schritten.

Moritz Tiedje ist seit 2015 professioneller Softwareentwickler bei andrena objects und beschäftigt sich mit allem, was hilft, Software effizienter und wertvoller zu entwickeln. Dabei konzentriert er sich hauptsächlich auf agile Software-Engineering-Praktiken und Softwarearchitektur. Auch verwandten Bereichen wie agile Frameworks, Coaching-Methoden, IT-Security, Moderationstechniken, Teamdynamiken usw. gilt sein Interesse. Seine Projekte haben ihm gezeigt, dass Herausforderungen selten nur technischer Natur sind.

David Burkhart ist seit 2004 in der professionellen Softwareentwicklung tätig. Sein besonderes Interesse gilt Clean Code Development, XP und Scrum. Bei andrena objects arbeitet er als Softwareentwickler, Trainer und Coach für agile Methoden und gibt seine langjährigen Erfahrungen im Bereich automatisiertes Testen, Clean Code und Continuous Integration in Kundenprojekten weiter. Neben seiner praktischen Tätigkeit in verschiedenen Technologien wie Java, JavaScript und Cloud teilt er sein Wissen regelmäßig auf Konferenzen wie dem Objektforum, Entwicklertag und Java Forum Stuttgart. Als Teil des Fachbeirats gestaltet er die XP-Days Germany maßgeblich mit. Seine Arbeitsphilosophie basiert auf Teamarbeit und Pair Programming, wobei er aktiv gegen Wissensinseln und für Transparenz, automatisiertes Feedback und hohe Automatisierung eintritt. Um kundenorientierte Lösungen zu entwickeln, folgt er dem MVP-Gedanken und nutzt z. B. „Specification by Example“ für die Entwicklung passender Tests.

Links & Literatur

[1] Dijkstra, Edsger W.: „The Humble Programmer“; Association for Computing Machinery, 1972

[2] https://agilemanifesto.org

[3] Fowler, Martin: https://martinfowler.com/bliki/FlaccidScrum.html

[4] Farley, David: „Modern Software Engineering“; Addison-Wesley, 2021

[5] Narayan, Sriram: https://martinfowler.com/bliki/OutcomeOriented.html

[6] Poppendieck, Mary; Poppendieck, Tom: „Implementing Lean Software Development“, Addison-Wesley, 2006

[7] https://dora.dev

[8] Feedback Loops in XP: http://www.extremeprogramming.org/map/loops.html

[9] Skelton, Matthew; Pais, Manuel: „Team Topologies: Organizing Business and Technology Teams for Fast Flow“; IT Revolution, 2019

[10] https://owasp.org/www-project-dependency-check/

[11] https://www.cve.org

[12] https://www.zaproxy.org

[13] https://www.threatmodelingmanifesto.org/#values

[14] Europäischer Rat der Europäischen Union: https://www.consilium.europa.eu/de/policies/top-cyber-threats/

[15] Europol: https://www.europol.europa.eu/publication-events/main-reports/internet-organised-crime-threat-assessment-iocta-2024

[16] https://github.com/renovatebot/renovate

[17] https://owasp.org/www-project-dependency-check/

[18] https://www.forbes.com/councils/forbesbusinesscouncil/2025/07/30/security-is-now-a-feature-not-an-afterthought/

[19] https://owasp.org/Top10/