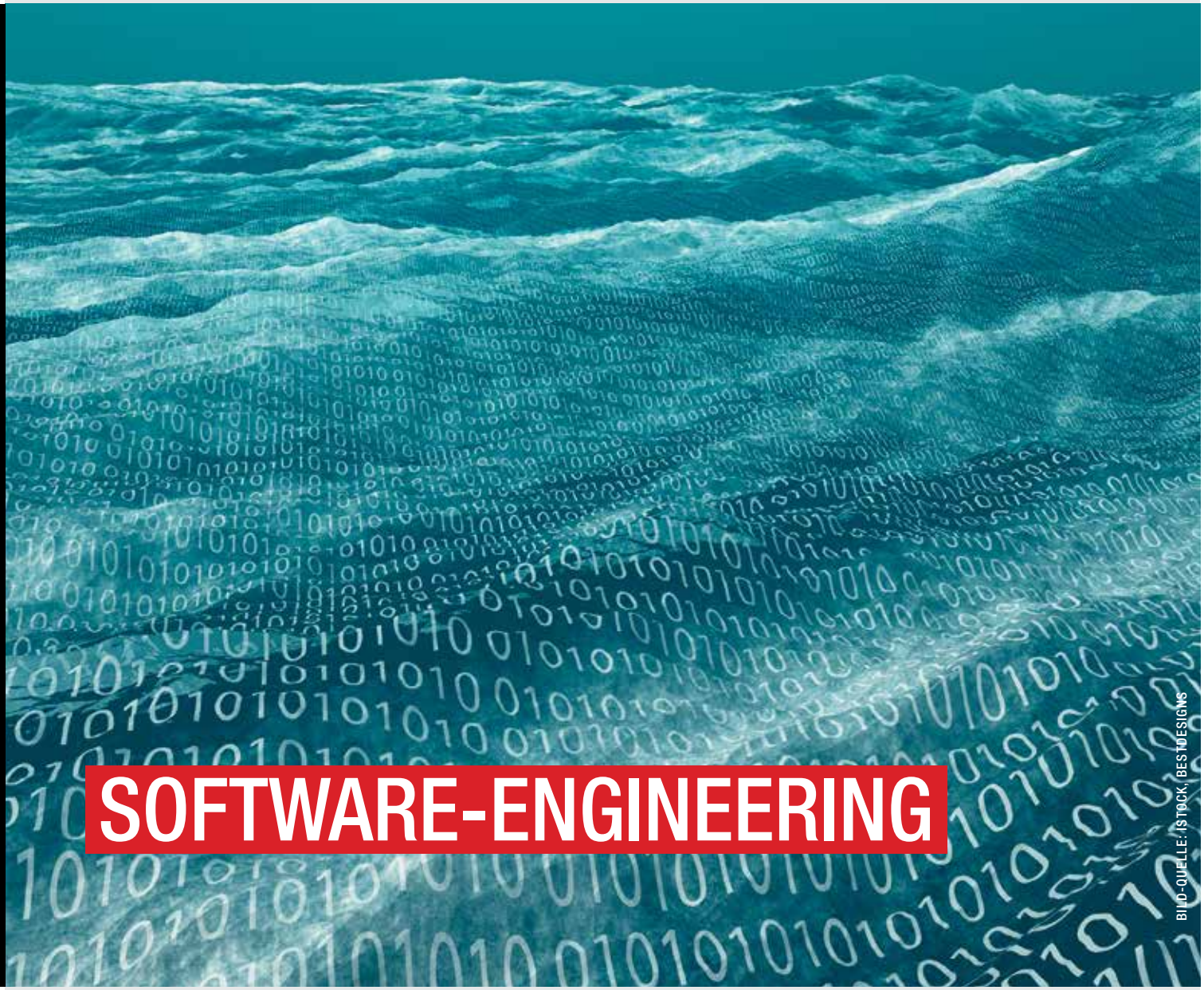
E&E FASZINATION ELEKTRONIK

publish industry verlag



SOFTWARE-ENGINEERING

BILD-QUELLE: ISTOCK, BESTVECTORS

Machine Learning

KI FÜR SELBSTFAHRENDE AUTOS

Eine der großen Schwierigkeiten für das autonome Fahren ist die Unstrukturiertheit des Verkehrs. Um sie zu bewältigen, sind gut trainierte KI-Systeme notwendig. Wie sich diese umsetzen lassen, erfahren Sie hier.

TEXT: Sorin Mihai Grigorescu, Markus Glaab, Joachim Schlosser, alle Elektrobit **BILD:** Elektrobit

In der Entwicklung softwarebasierter Systeme ist Machine Learning einer der wichtigsten Trends. Laut einer Studie des US-Technologieunternehmens Narrative Science nutzen schon heute 40 Prozent aller Unternehmen auf die eine oder andere Weise Künstliche Intelligenz (KI). Bis 2018 soll der Anteil sogar auf 60 Prozent steigen.

Machine Learning ist eine wichtige Teildisziplin der KI. Dabei treffen Algorithmen Entscheidungen oder Voraussagen auf Basis vorheriger Lernprozesse. Das Grundprinzip ist seit Jahrzehnten bekannt, jedoch sind seine Einsatzmöglichkeiten sowie die Qualität der Resultate in den vergangenen Jahren massiv gewachsen. Zu verdanken ist diese Entwicklung zum

einen der stark gestiegenen Rechenleistung und zum anderen der Weiterentwicklung der genutzten Algorithmen.

Erhebliche Fortschritte brachte das Konzept „Deep Learning“, bei dem die Interpretation von Daten nicht mehr durch Programmierer endgültig festgelegt wird. Stattdessen lernen entsprechende Systeme selbständig auf Basis geeigneter Trainingsdaten. So gelingt es Deep-Learning-Algorithmen, Muster in großen Datenmengen zu erkennen und daraus für sich selbst eine komplexere innere Struktur zu entwickeln. Diese Architektur ermöglicht bereits heute beeindruckende Ergebnisse. Zu den bereits auf dem Markt verfügbaren KI-Anwendungen zählen etwa das Erkennen natürlicher Sprache oder virtuelle Agenten für persönliche Assistenzfunktionen. KI-Systeme haben sich in jüngerer Zeit bereits sehr erfolgreich mit menschlichen Gegnern gemessen. Im März 2016 schlug zum Beispiel das KI-Programm AlphaGo einen professionellen Gegenspieler im Brettspiel Go. Das zwischenzeitlich von Google aufgekaufte KI-System DeepMind ist in der Lage, 49 klassische Atari-Videospiele erfolgreich zu bewältigen. Wie jede KI-Software lernen die genannten Systeme dabei aus Erfolg und Misserfolg.

Diese Fortschritte beweisen, dass Künstliche Intelligenz in der Lage ist, extrem komplexe Aufgabenstellungen erfolgreich zu bewältigen. Daher liegt der Gedanke nahe, sie auch im Kontext des hochautomatisierten Fahrens einzusetzen. Tatsächlich wurden hier im Rahmen von Forschungsprojekten schon vielversprechende Resultate erzielt. Für einen kommerziellen Einsatz müssen allerdings noch weitere Herausforderungen bewältigt werden – insbesondere in Hinsicht auf die funktionale Sicherheit. Dennoch gelten Machine Learning und Deep Learning als wichtige Werkzeuge bei der Entwicklung autonomer Fahrzeuge. Elektrobit konzentriert sich in diesem Zusammenhang derzeit auf die Erforschung und Entwicklung von zwei Anwendungsfällen: Die Erkennung von Verkehrszeichen und die Steuerung des Fahrzeugs.

Die größte Herausforderung des realen Straßenverkehrs ist, dass es sich um eine ausgesprochen unstrukturierte Umgebung handelt. Um dieses Problem zu lösen, zerlegen die Entwickler typischerweise die komplexe Gesamtsituation in einfacher strukturierte Teilaufgaben. Zwar erlauben Deep-Learning-Mechanismen grundsätzlich auch Ende-zu-Ende-Ansätze, bei denen eine Dekomposition der Problemstellung in kleinere Bausteine gar nicht mehr erforderlich ist. Jedoch schätzen wir diese Vorgehensweise zum gegenwärtigen Zeitpunkt als weniger geeignet ein, weil sie die Möglichkeiten für Testing und Validierung einschränkt. Das Zerlegen des hochkomplexen Gesamtsystems Straßenverkehr in Einzelbausteine mit klar definierten Schnittstellen vereinfacht die Analyse, die Entwicklung und den Test von KI-Systemen dagegen wesentlich.

KI verhält sich wie Blackbox

Dennoch braucht die Nutzung von Machine Learning auch bei der Entwicklung und beim Testing neue Ansätze. Während traditionelle Softwarekomponenten gegen konkrete Anforderungen getestet werden, verhalten sich KI-Systeme und ihre stochastischen Lernstrategien wie eine Blackbox. Eine Vorhersage, wie und in welcher Struktur das System lernt, ist praktisch nicht möglich. Auch beim Testing hilft eine Unterteilung in einfachere Teilaufgaben die Komplexität der KI-Lösungen und ihrer Aufgabenstellungen zu reduzieren.

Elektrobit nutzt bei der Entwicklung selbstlernender Fahrzeuge die neueste Generation von Deep-Learning-Technologien. Diese nutzen Belohnungsstrategien – sogenanntes Reinforcement Learning. Der Lernprozess der Software wird dabei durch ein Belohnungssystem gesteuert, welches es ermöglicht, für die gestellten Aufgaben eine unabhängige Lösungsstrategie zu entwickeln. Das Design solcher Systeme erfordert allerdings detailliertes Wissen über das jeweilige Anwendungsgebiet.

Die drei Klassen des Machine Learning

Grundsätzlich lässt sich Machine Learning nach den Lernbedingungen in drei verschiedene Klassen einteilen:

- Bei überwachtem Lernen steht das erwünschte Ergebnis im Vorfeld fest. Der Algorithmus lernt, aus den Input-Daten die korrekte Entscheidung abzuleiten.
- Bei unüberwachtem Lernen sind die Input-Daten nicht gekennzeichnet beziehungsweise strukturiert – Aufgabe des Algorithmus ist es, Strukturen und Eigenschaften der bereitgestellten Daten zu analysieren und zu identifizieren.
- Bei sogenanntem halbüberwachtem Lernen sind die Input-Daten ebenfalls nicht vorstrukturiert oder gekennzeichnet. In diesem Fall erhält der Algorithmus die Aufgabe, aus einer zur Verfügung stehenden Anzahl von Aktionen die bestmögliche auszuwählen und auf diese Weise mit seiner Umgebung zu interagieren. Dieses Konzept wurde auch bei dem eingangs erwähnten Beispiel eingesetzt, bei dem das KI-System DeepMind das erfolgreiche Spielen von Atari-Videogames erlernte. Auch für die „Fahrschule für Algorithmen“ kommt halbüberwachtes Lernen zum Einsatz.

Um dem Algorithmus das „Auto fahren“ in einem sicheren Umfeld beizubringen, realisierten Elektrobits Softwareingenieure ein Deep-Learning-System auf Basis des als Open-Source-Software angebotenen Autorenn-Simulators Torcs. Die Game-Engine generiert Bewegtbilder, die sich zur Objekterkennung nutzen lassen; und stellt gleichzeitig weitere Daten zur Verfügung, wie etwa die aktuelle Geschwindigkeit des zu steuernden Fahrzeugs, dessen relative Position auf der Straße und den Abstand zu vorausfahrenden Autos. Alle diese Bilder und Daten dienen als Input für das KI-System. Das System leitet daraus Fahrstrategien und die erforderlichen Steuerkommandos für das automatisiert fahrende Auto ab, um so mit der virtuellen Umgebung zu interagieren.

Der Simulator Torcs unterstützt eine Vielzahl von Fahrzeugen und Strecken, wodurch dem Deep-Learning-Algorithmus ein größeres Angebot an Einsatzumgebungen und Situationen zur Verfügung steht – und somit ein breiteres Spektrum an Input-Daten. Aus technischer Sicht handelt es sich bei dem gewählten Algorithmus um ein tief gefaltetes neuronales Netz (Deep Convolutional Neural Network, kurz DNN), das durch positive Belohnungssignale trainiert wird.

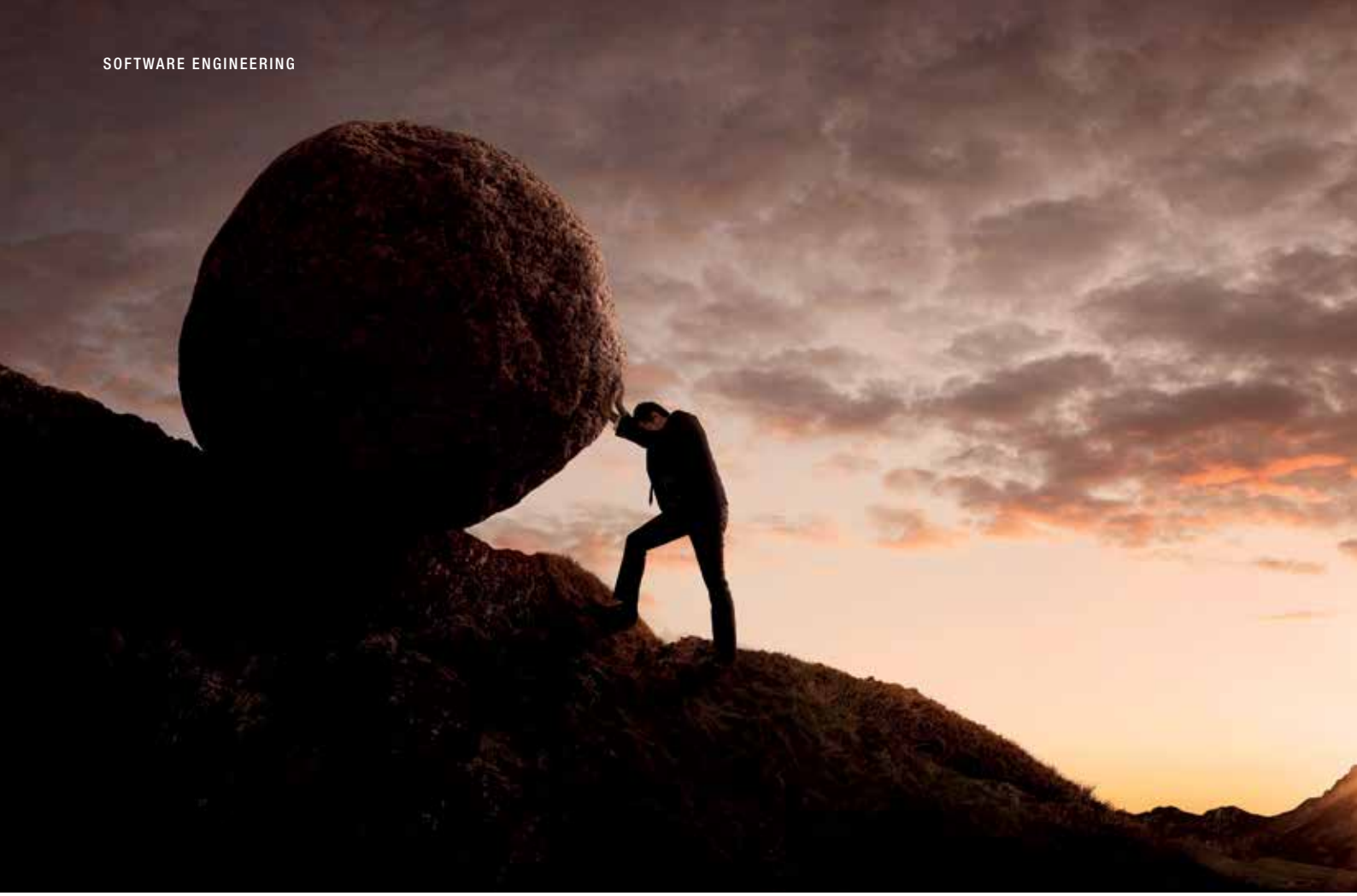
Seine Funktionsweise lässt sich an einem Beispiel verdeutlichen: In einer bestimmten Situation der Simulation stehen vier mögliche Aktionen zur Auswahl – beschleunigen, abbremsen,

nach links steuern oder nach rechts steuern. Das DNN berechnet, anhand einer sogenannten Q-Funktion, welche dieser vier Optionen es als die optimale Reaktionen auf die aktuelle vorliegenden Situation einschätzt. Die getroffene Entscheidung führt dann zu einer Zustandsänderung des Simulators - das Spiel reagiert auf die Steuerung durch den Algorithmus. Ist der so herbeigeführte Zustand positiv zu bewerten, erhält das KI-System eine entsprechende Rückmeldung mit Belohnung. Hat das Fahrzeug beispielsweise beschleunigt, ohne mit anderen Fahrzeugen zu kollidieren, wird diese Entscheidung belohnt und somit im neuronalen Netz verstärkt. Bei ungünstigem Ergebnis erfolgt dagegen keine Belohnung. Das System wird dadurch von dem gewählten Entscheidungsweg für die Zukunft abgeschreckt.

Passendes KI-Netz auswählen

Elektrobit setzt für diese Entwicklungsumgebung die hauseigenen Produkte EB Cadian und EB Robinos ein. Ersteres dient zur Analyse von Schwarmdaten aus weltweit verteilten Fahrzeugflotten, die mittels etablierten Standards wie beispielsweise ODX und UDS gesammelt werden. Im Zusammenhang mit Machine Learning können die Daten von EB Cadian beispielsweise für Predictive-Maintenance verwendet werden. EB Robinos ist eine Entwicklungsumgebung und ein Framework für komplexe Anwendungen im Bereich des autonomen Fahrens. Innerhalb seiner Module werden an verschiedenen Stellen Machine-Learning-Methoden genutzt, um komplexe Aufgaben zuverlässig zu lösen. Bislang entwickelte Prototypen haben mit den beschriebenen Mechanismen bereits sehr gute Erfolge bei der Lösung der komplexen Aufgabenstellung Straßenverkehr erzielt. Als entscheidend hat sich dabei erwiesen, das für die jeweilige Aufgabenstellung passende neuronale Netz auszuwählen.

Verbleibende Herausforderungen sind insbesondere die Anforderungen an die funktionale Sicherheit von Systemen, die auf Machine Learning basieren, sowie die Entwicklung geeigneter Test- und Validierungskonzepte. Trotz dieser Herausforderungen setzt Elektrobit darauf, dass Machine Learning in Zukunft für hochautomatisiertes Fahren eine entscheidende Rolle spielen wird. Die auf diesem Gebiet bereits erzielten Ergebnisse sind jedenfalls vielversprechend. ☐



SOFTWARESCHUTZ

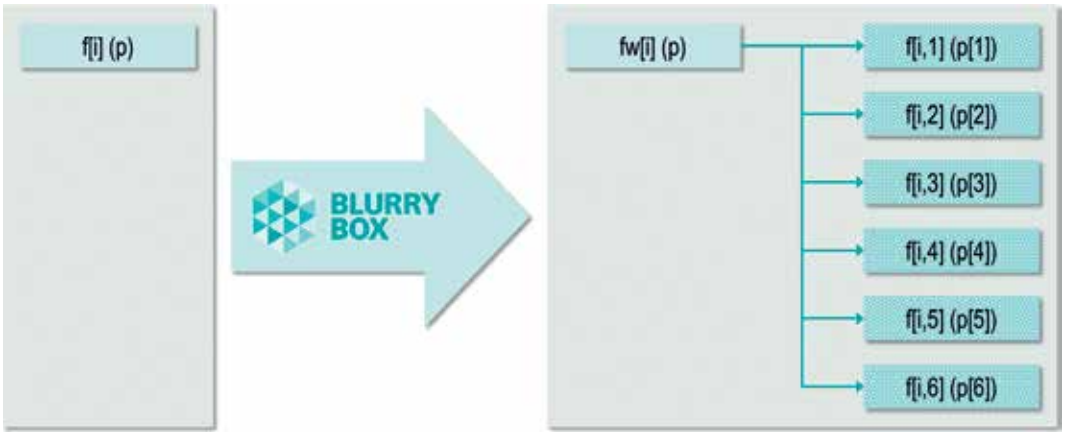
Zu viel Aufwand für den Hacker

Mit der Blurry-Box-Technologie lässt sich Software gegen Reverse Engineering und das Raubkopieren schützen. Das funktioniert so gut, dass sich bei einem Hackers Contest selbst Meister ihres Fachs die Zähne daran ausgebissen haben.

TEXT: Rüdiger Kügler, Wibu-Systems **BILDER:** Wibu-Systems; iStock, Nastco

Dem Kerckhoffs'schen Prinzip zufolge sollte die Sicherheit eines kryptographischen Verfahrens auf der Geheimhaltung des Schlüssels und nicht auf der Geheimhaltung des Algorithmus basieren. Diese Maxime aus dem Jahre 1883 ist heute aktueller denn je. Öffentlich bekannte Verschlüsselungsverfahren sind in der Regel besser untersucht als geheime Algorithmen. Haben sie einer Überprüfung standgehalten, sind sie deshalb vertrauenswürdiger. Aber auch andere Gründe sprechen für das Prinzip. Es ist zum Beispiel schwieriger, einen Algorithmus geheim zu halten als einen Schlüssel, da sich Algorithmen durch Reverse Engineering der Soft- und Hardware analysieren lassen. Algorithmen können außerdem kompromittiert werden, wenn Insider, meist unzufriedene Mitarbeiter, Firmengeheimnisse verkaufen. Zudem ist es in der Praxis deutlich aufwendiger, den Algorithmus auszutauschen als lediglich einen Schlüssel. Wird ein Algorithmus offengelegt, wird er auch stärker nach Hintertüren, Implementierungsfehler und Sicherheitslücken überprüft. Diese können von den Programmierern dann beseitigt werden.

In der Vergangenheit basierte Softwareschutz meist auf Geheimniskrämerei. Jeder Anbieter hatte den vermeint-



Bei Blurry Box werden die einzelnen Methoden einer Software zu Varianten vervielfältigt, um Hackern den Zugang zu erschweren.

lich sichersten und einfachsten Schutz implementiert. Marketing-Prospekte überschlugen sich mit Superlativen. Ein Vergleich verschiedener Anbieter war auf Grund der Geheimhaltung jedoch nicht einmal ansatzweise möglich, geschweige denn eine neutrale Prüfung.

Ein beliebtes Verfahren ist zum Beispiel die Verwendung von Kopierschutzsteckern, auch Dongles genannt. Dabei wird ein Algorithmus im Dongle implementiert und aus der Software heraus aufgerufen. In der Software sind Tabellen mit Anfragen und den dazu passenden Antworten abgelegt. Wenn die vom Dongle gelieferte Antwort mit der erwarteten Antwort übereinstimmt, wird die Software korrekt ausgeführt, andernfalls wird eine Fehlermeldung angezeigt und die Software beendet. Im Wesentlichen reduziert sich der Schutz also auf eine simple Ja/Nein-Entscheidung. Auf Assembler-Ebene ist dies ein JNZ oder ein JZ (Jump Not Zero oder Jump Zero) mit den entsprechenden binären Codes 74 beziehungsweise 75. Ein Hacker muss also nur an der richtigen Stelle eine 74 durch eine 75 ersetzen oder umgekehrt, und schon läuft die Software ohne Lizenz. In den meisten Fällen wurden Dongles deshalb bereits nach kurzer Zeit geknackt.

Heute setzen alle großen Anbieter auf Standard-Algorithmen im Dongle. In der Regel wird AES verwendet mit einer Schlüssellänge von 128 oder 256 Bit. Für die Sicherheit ist die Länge des Schlüssels aber nicht das entscheidende Kriterium. Wichtiger ist, mit welcher Technik der Algorithmus in der Software abfragt wird. Die meisten Anbieter geben zwar an, dass die Software verschlüsselt ist und nur mit dem passenden Dongle entschlüsselt werden kann. Damit hört das offengelegte Wissen aber in den meisten Fällen auf.

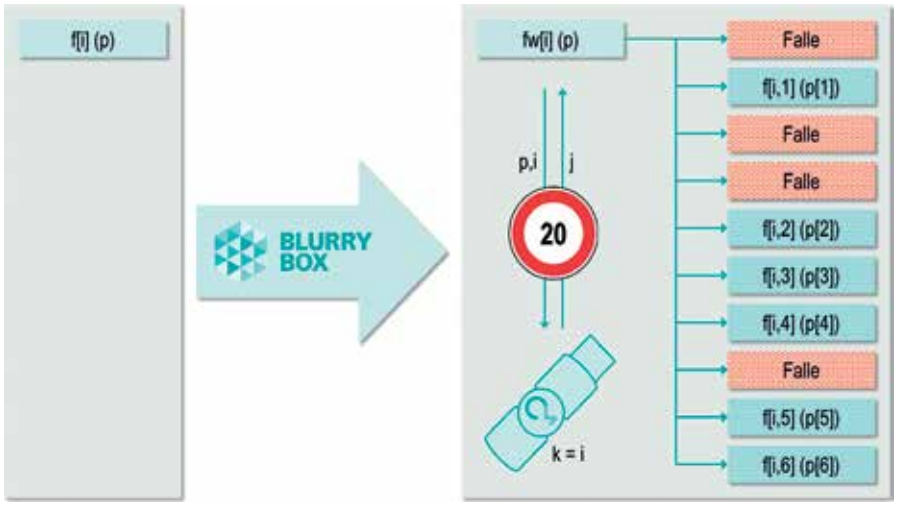
An dieser Stelle beginnt die Erfolgsgeschichte des Softwareschutzes Blurry Box. Entworfen wurde er von Forschern des Karlsruher Institut für Technologie (KIT) und Entwicklern des Securityunternehmens Wibu-Systems. Ihr Ziel war es, einen Softwareschutz zu entwickeln, bei dem auch die Integration in die Software offengelegt werden kann. Das Ergebnis dieser Kooperation erwies sich als so überzeugend, dass Blurry Box 2014 auf Anhieb den 5. Deutschen IT Sicherheitspreis gewann.

Blurry Box funktioniert folgendermaßen: Hacker, die einen Softwareschutz entfernen wollen, gehen entweder statisch oder dynamisch vor. Bei der statischen Analyse wird der Code nur angesehen, verschlüsselte Teile werden gefunden, manuell entschlüsselt und wieder zusammengesetzt. Bei der dynamischen Analyse hingegen wird die ablaufende Software beobachtet und Teile aus dem Speicher, vor allem bereits entschlüsselter Code, werden entnommen. Die Blurry-Box-Technologie soll beide Angriffsformen sowie beliebige Kombinationen daraus verhindern. Ein hundertprozentiger Schutz ist unmöglich, da jede Software irgendwann einmal irgendwo ausgeführt wird. Sinnvoll ist es deshalb den Aufwand für das Hacken der Software soweit zu vergrößern, das er gleich oder höher ist als der Aufwand für die Neuerstellung einer vergleichbaren Software.

Folgende Grundannahmen wurden für Blurry Box außerdem getroffen:

- Die Software ist hinreichend komplex.
- Der Hacker versteht sein Handwerk.
- Der Hacker kann die Funktionsweise der Software nicht im Detail beurteilen.

Vor allem die erste Annahme ist wichtig, um das Schutzziel zu erreichen. Eine Anwendung, die lediglich eine Ausgabe in der Kommandozeile ausspuckt,



Entschlüsselt der Hacker eine Falle, wird ihm automatisch der Zugang zu den nachfolgenden Methoden verwehrt.

ist nicht komplex genug, um mit Blurry Box geschützt zu werden. Schließlich braucht es nur wenige Minuten, um die Anwendung neu zu schreiben. Bei einer realen Anwendung verwendet ein normaler Anwender circa 10 Prozent der verfügbaren Funktionen. Testprogramme schaffen ungefähr eine Codeabdeckung von 80 Prozent. Anhand dieser Zahlen lassen sich die Herausforderungen bei einer dynamischen Analyse erahnen. Der Angreifer müsste alle Funktionen einmal aufrufen, um diese beobachten zu können. Das ist eine Aufgabe, die selbst die interne Qualitätssicherung des Softwareherstellers nicht zu 100 Prozent erfüllen kann! Wenn die Anwendung hinreichend komplex ist, wird ein rein dynamischer Hack deshalb immer unvollständig sein.

Blurry Box besteht aus sieben Einzelschritten, die teilweise bereits im Feld erprobt, im Rahmen von Blurry Box verbessert oder neu erfunden wurden.

Schritt 1 - Bilden von Varianten: Jede Software besteht aus einzelnen Methoden, die zur Laufzeit ausgeführt werden. Blurry Box erschafft Varianten dieser Methoden. Beispielsweise werden aus einer Methode sechs neue Methoden erzeugt. Zusätzlich wird eine Auswahlmethode (Wrapper) eingefügt. Der Wrapper wird anstelle der originalen Methode aufgerufen und wählt abhängig von den Eingabeparametern eine der neuen Methoden aus, die dann aufgerufen wird. Die Varianten erhöhen die Komplexität einer dynamische Analyse. Um nämlich alle Varianten beobachten zu können, müsste der Angreifer die Wertebereiche aller Parameter kennen und die Programmdurchläufe so modifizieren, dass sie alle Varianten einmal aufrufen.

Schritt 2 - Modifikation der Varianten: Gemäß dem Kerckhoffs'schen Prinzip kennt der Angreifer Blurry Box, das heißt er weiß, dass eine Auswahlmethode eingefügt wurde. Daher könnte er den Wrapper finden und so einstellen, dass immer die Variante 1 verwendet wird. Das wird verhindert, indem die Varianten so modifiziert werden, dass sie nur im entsprechenden Wertebereich der Eingabeparameter korrekt funktionieren. Dadurch liefert ein potentieller Variante-1-Hack fehlerhafte Ergebnisse. Die Modifikationen reichen von einfachen Wertebereichsprüfungen über Hinzufügen von „* 1“ und „+ 0“, wenn die Parameter genau diese Werte annehmen, bis zur Optimierung der Methoden. Es werden etwa Operationen weggelassen oder Variablenlängen verkürzt. Eine Rücknahme der Modifikation erfordert eine detaillierte Analyse jeder Funktion sowie ein fundiertes Wissen über die Funktionsweise der Software. Das erschwert die Arbeit des Hackers deutlich.

Schritt 3 - Verschlüsselung der Varianten: Jede einzelne Variante wird mit AES verschlüsselt. Eine Analyse des Codes in der Variante ist damit ohne Schlüssel überhaupt nicht und mit Schlüssel erst nach Entschlüsselung möglich. Diese Technologie kommt im Rahmen der Codemeter Protection Suite von Wibu bereits seit vielen Jahren zum Einsatz. Neu an Blurry Box ist, dass sie auf die einzelnen Varianten angewendet wird, während im klassischen Fall einfach die originale Methode verschlüsselt wurde.

Schritt 4 - Einfügen von Fallen: Der Codemeter-Dongle (CmDongle) besitzt die Eigenschaft, dass eine entsprechende Sperrsequenz die AES-Schlüssel im Dongle ungültig machen kann. Bei Blurry Box werden weitere Methoden in die Anwendung eingefügt, die als Fallen dienen. In der obigen Abbildung sind zum Beispiel vier Fallen im Code platziert. Bevor die eingefügten Methoden verschlüsselt werden, wird an deren Anfang eine Sperrsequenz geschrieben. Die Auswahlfunktion wird so modifiziert, dass sie für ungültige Eingabeparameter

Mit sieben Maßnahmen macht Blurry Box Hackern das Leben schwer. Sie erhöhen den Aufwand, der für die Analyse einer Software nötig ist. Dadurch ist es uninteressant die Software zu kopieren oder zu cracken.

diese Fallen-Methoden aufruft. Ohne ein tieferes Verständnis der Programmlogik kann der Hacker nicht entscheiden, ob es sich um eine Falle oder eine benötigte Variante handelt. Entschlüsselt der Hacker eine dieser Fallen, so wird der CmDongle gesperrt und eine Entschlüsselung der nachfolgenden Methoden ist unmöglich. Jetzt müsste er sich einen neuen CmDongle mit der passenden Lizenz besorgen, was bei einer hinreichend großen Anzahl von Fallen nicht im Verborgenen erfolgen kann.

Schritt 5 - Variantenauswahl im Dongle: Durch eine Codeanalyse der Auswahlfunktion wäre es jedoch noch möglich, Wertebereiche und entsprechende Fallen zu erkennen. Um das komplett zu verhindern, wird die Variantenauswahl in den CmDongle verlagert. Die Auswahlfunktion schickt eine ID und einen Parameter an den CmDongle. Dieser entscheidet dann, welche der Varianten ausgewählt werden soll. In der Grafik auf der nebenstehenden Seite trifft er beispielsweise die Wahl zwischen 10 Varianten.

Schritt 6 - Dongle als State-Engine: Die ersten fünf Schritte von Blurry Box erschweren die Analyse der Software. Dennoch wäre es denkbar, dass ein Hacker durch systematisches Abfragen des CmDongles die Zuordnung der einzelnen Varianten zu den Wertebereichen der Parameter herausfindet. Um das zu verhindern, dient der CmDongle als State-Engine. Indem Ketten von erlaubten Zuständen gebildet werden, weiß der Softwareentwickler, welche Methoden vor beziehungsweise nach einer bestimmten Methode aufrufbar sind. Diese Zustände werden im CmDongle gespeichert. Dieser antwortet nur noch auf die Anfrage der Variantenauswahl, wenn vorher die richtige Vorgängerfunktion aufgerufen wurde. Für den Hacker ist es damit nicht mehr möglich, jede Auswahlfunktion einzeln zu analysieren.

Schritt 7 - Geschwindigkeitsbeschränkung: Wie auch im Straßenverkehr ist beim Entschlüsseln die maximale Geschwindigkeit manchmal eher gefährlich als nützlich. Der Entwickler weiß genau, wie seine Software funktioniert und in welchen zeitlichen Abständen den geschützte Methoden entschlüsselt werden. Daraus ergibt sich eine sinnvolle Geschwindigkeit für die Entschlüsselung im CmDongle. Indem der Entwickler bei Blurry Box eine durchschnittliche Pausenzeit zwischen zwei Entschlüsselungsoperationen festlegt, dauert es für einen Hacker nochmals deutlich länger, alle Variantenkombinationen durchzuprobieren.

Blurry Box sorgt somit dafür, dass die Analyse für den Hacker so verkompliziert und die Zeit für die Erstellung eines Hacks so erhöht wird, dass es wirtschaftlicher ist, die Software komplett neu zu schreiben. Dabei kann der Hacker alle von Blurry Box verwendeten Strategien kennen, ohne dass ihm daraus ein Vorteil erwächst. Damit wird Blurry Box nicht nur dem Kerckhoffs'schen Prinzip gerecht, sondern macht es endlich auch möglich, Softwareschutz zu messen und zu vergleichen.

Nach dem Gewinn des 5. Deutschen IT Sicherheitspreises wurde die Blurry-Box-Technologie in die Codemeter Protection Suite von Wibu-Systems integriert. Außerdem hatte das Unternehmen die Idee, Blurry Box im Rahmen des Hackers Contest 2017 von echten Profis auf Herz und Nieren testen zu lassen. Als Beispielanwendung für den Contest wurde ein Reisespiel mit Fragen zu deutschen Städten ausgewählt. Den 315 Teilnehmern wurde kostenfrei das CmDongle mit allen für die vollständige Ausführung des Spiels notwendigen Schlüsseln zur Verfügung gestellt. Am Ende des Contests wurden zwei Einsendungen registriert. Beide waren Replay-Attacken auf den CmDongle, die aber nicht zu einem ohne CmDongle korrekt lauffähigen Spiel führten. ☐

Open-Source-Software

Freiheit bei der Programmierung

Open-Source-Software (OSS) hat die Software-Entwicklung in den letzten zehn Jahren grundlegend verändert. OSS erlaubt Entwicklern mehr Agilität und spart Unternehmen Zeit und Kosten. Bei der Auswahl von Komponenten gibt es jedoch einige Faktoren zu beachten – insbesondere wenn es um Fragen der Lizenzierung und Sicherheit geht.

TEXT: Jeff Luszcz, Flexera **BILDER:** Flexera; iStock, Anyaberkut

Es ist noch nicht so lange her, dass Software-Entwicklungsteams darüber entschieden, ob eine bestimmte Funktionalität selbst entwickelt oder eingekauft werden sollte. Die Entscheidung war nicht immer nur eine Kostenfrage. Auch das fehlende Wissen über kommerzielle Komponenten Dritter hatte zur Folge, dass Software-Entwickler den größten Teil des Quellcodes – mehr als 90 Prozent – in ihren Softwareprodukten letztlich selbst schrieben.

Mit dem Internet hat sich das grundlegend geändert. Die weltweite Vernetzung erlaubt einen engen Austausch zwischen Software-Entwicklern und bietet eine Plattform, auf der Millionen kostenloser Softwarekomponenten zur Verfügung stehen. Heute liegt der Anteil der OSS-Produkte bei 50 bis 90 Prozent und umfasst Hunderte, wenn nicht Tausende von OSS-Komponenten.

Mehr Freiheit braucht mehr Kontrolle

Der Einsatz von OSS-Komponenten ermöglicht Unternehmen, große und anspruchsvolle Projekte zu stemmen, die letztlich auf „Best of Breed"-Komponenten beruhen. Das hat auch Folgen für die Entwicklungsprozesse. Um zum Beispiel Vertragsbedingungen auszuhandeln, Lizenzen zu erwerben und

Vertraulichkeitserklärungen oder Service Level Agreements abzuzeichnen, mussten früher alle Softwarekomponenten von Drittanbietern durch IT, Einkauf und Rechtsabteilungen abgesegnet werden. Heute hingegen finden abteilungsübergreifende Absprachen nur noch selten statt. Zudem wird der Einsatz von OSS in vielen Fällen nicht mit geltenden Richtlinien abgeglichen. Sind die Komponenten erst einmal ausgewählt und integriert, werden sie ohne regelmäßige Überprüfung einfach weiter verwendet.

Die Risiken solcher nicht verwalteter Open-Source-Software sind groß, werden aber von vielen Unternehmen nicht ernst genommen. In den meisten Fällen fehlt dafür schlichtweg eine Übersicht der im Unternehmen genutzten OSS-Komponenten. Bei Konzernübernahmen – wenn Transparenz eine zentrale Rolle spielt – kann häufig kein einziges OSS-Projekt genannt werden. Liegt ein Verzeichnis mit Open-Source-Komponenten vor, führt dieses für gewöhnlich nur einen Bruchteil auf – im Schnitt 20-mal weniger Open-Source-Komponenten als tatsächlich im Einsatz sind.

Wird Open-Source-Software nicht ordnungsgemäß verwaltet, kommt es zu zwei grundlegenden Problemen: Zum einen laufen Unternehmen Gefahr, gegen die Compliance-Richtlinien der von ihnen genutzten Lizenzen zu verstoßen. Zum anderen setzen sie sich dem Risiko unentdeckter Softwareschwachstellen aus, die sich in der OSS befinden.

Lizenzierungen prüfen

Geht es um die Auswahl geeigneter Komponenten sollten Software-Entwickler daher bestimmte Aspekte berücksichtigen – ganz gleich, ob es sich um Open Source oder um ein kommerzielles Angebot handelt. An erster Stelle stehen die legalen Rahmenbedingungen bei der Verwendung der Komponente und die damit verbundenen Verpflichtungen. Bei kommerziellen Komponenten gilt es, Zahlungs- und Nutzungsbedingungen zu verhandeln. Aber auch Open-Source-Komponenten müssen hinsichtlich ihrer Lizenzierung geprüft werden. Wer beispielsweise eine Komponente in seinem Softwareprodukt nutzt, die unter die GNU General Public License fällt, verpflichtet sich unter anderem dazu, den Quellcode für das Produkt sowie den Quellcode für die ursprüngliche Komponente bereitzustellen. Komponenten mit MIT-Lizenz machen es erforderlich, einen Urheberrechts- und Genehmigungshinweis in die Software aufzunehmen.

Ist die Lizenzierung einer Open-Source-Komponente nicht eindeutig, ist Vorsicht geboten. Entweder wird die Komponente nicht häufig genutzt oder der Kreis der Benutzer ist re-

Absprachen, beispielsweise zwischen dem Engineering- und Sicherheits-Team, sind für ein effektives OSS-Management entscheidend.

lativ klein. So oder so ist es denkbar, dass ein Mitentwickler im Rahmen des ursprünglichen Entwicklungsprojekts nachträglich einen Antrag stellt, um den Code unter eine Lizenz zu stellen oder die lizenzrechtliche Lage neu festzulegen. Wird die Komponente bereits in einem Softwareprodukt verwendet, kann das für Softwarehersteller zu einem ernsten Problem werden. Die Entscheidung, ob eine profitable Software vom Markt genommen werden muss, hängt dann oft allein von der Reaktion des Autors/Urhebers der OSS oder des Projektverantwortlichen ab.

Alter und Wartung berücksichtigen

Neben der Lizenzierung lohnt es sich zudem, bei der Auswahl einer OSS-Komponente auf das Alter und den Wartungszustand des zugrundeliegenden Projekts zu achten. Gab es in den vergangenen Jahren keine weiteren Beiträge? Oder beschränkt sich die Mitwirkung am OSS-Projekt auf einen einzigen Autor? Die Zahl der aktiven Entwickler an einem Projekt und die aktive Lebensdauer eines Projekts sind wichtige Indikatoren für dessen Vitalität. Es gibt jedoch auch eine Reihe von OSS-Projekten, die ausgereift sind und nur von einer einzigen Person getragen werden. Ihr Vorteil: Änderungen sind seltener notwendig als in Komprimierungs- und Bildverwaltungsbibliotheken. Hier findet man üblicherweise wenige Mitwirkende, aber viele Nutzer.

Sicherheit groß schreiben

Eine zentrale Rolle spielen auch Sicherheitsaspekte – in erster Linie mögliche, in OSS-Komponenten enthaltene Schwachstellen. Entwickler können hier auf Datenbanken und Feeds zurückgreifen, in denen aktuelle Listen bekannter Schwachstellen für die entsprechenden Komponenten geführt werden. Eine der bekanntesten Quellen mit Informationen zu OS-Schwachstellen ist zum Beispiel die National Vulnerability Database. In dieser Datenbank können Entwickler nach Komponentennamen suchen und die damit verbundenen Schwachstellen versionsspezifisch einsehen.

Trotz aller Vorsicht sollte man dabei jedoch bedenken, dass eine Schwachstelle nicht zwangsläufig ein Hinweis auf eine qualitativ minderwertige Komponente ist. Je beliebter oder wichtiger eine Komponente ist, desto wahrscheinlicher wird sie von Sicherheitsexperten mit dem Ziel überprüft werden, die Qualität zu verbessern. Eine Komponente, für die bereits Schwachstellen gemeldet sind, ist oft sicherer als eine Komponente, die diesbezüglich eine augenscheinlich „weiße Weste“ besitzt.

Mit der Auswahl einer Komponente ist das OSS-Management jedoch noch nicht zu Ende. Eine regelmäßige Überprüfung auf bekanntgewordene Schwachstellen ist zentral, um die Sicherheit der Anwendungen gewährleisten zu können. Neue Angriffe über Schwachstellen werden typischerweise erst dann aufgedeckt, nachdem eine Komponente freigegeben und für gewisse Zeit verwendet wurde. Behält man hier den Überblick, lassen sich nach dem Bekanntwerden neuer Schwachstellen die Anwendungen über Updates schützen und Patches bereitstellen.

Schwachstellen, Lizenzierung und Projektverlauf gehören beim Einsatz von Open Source auf die Checkliste und stellen sicher, dass bei der Auswahl die beste Komponente für die jeweilige Anwendung zum Einsatz kommt. Für Entwickler wie Unternehmen lohnt es sich zudem Feedback, Bug Reports und Prüfungsergebnisse an die betreffenden Open-Source-Projekte zurückzuspielen und so effektiv und langfristig zur gesamten Open-Source-Community beizutragen. Diese Aktivitäten zahlen sich aus: Durch hochwertige OSS-Komponenten, sicherere Produkte, besser gesteuerte Upgrade-Zyklen und weniger Nacharbeit. ☐

Software-Entwicklung

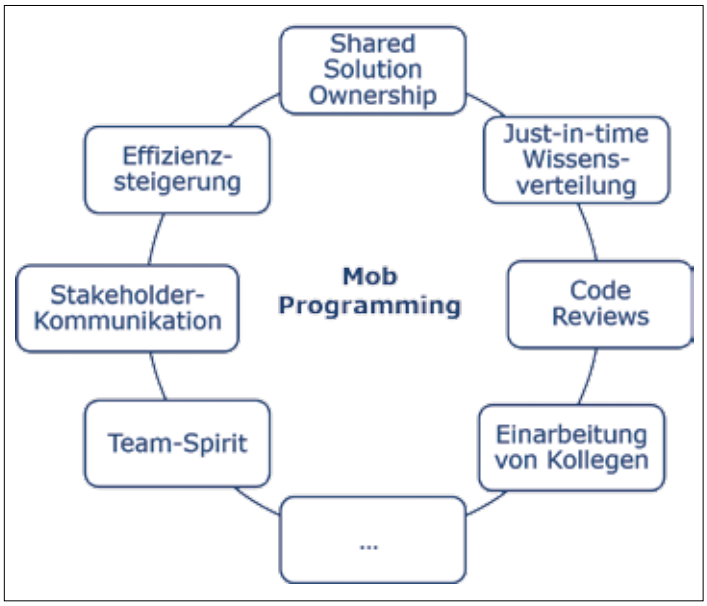
GEMEINSAM GESTALTEN

Mob Programming zählt zu den jüngsten Ansätzen in der Software-Entwicklung. Die Methode bündelt Teamkräfte, um so die bestmögliche Lösung für eine Aufgabenstellung zu erhalten. Dadurch wird nicht nur die Entwicklungsqualität erhöht, sondern auch der Zusammenhalt des Teams gestärkt.

TEXT: Christian Hoffmann, Method Park **BILDER:** Method Park; iStock, Alashi

Die aktuelle Software-Entwicklung ist geprägt von agilen Methoden, selbstorganisierenden Teams und dem Gedanken des Shared Code Ownerships. Um in derartigen Konstellationen qualitativ hochwertige Produkte zu entwickeln, ist ein stetiger Erfahrungsaustausch innerhalb der Teams auf verschiedenen Abstraktionsebenen unumgänglich. Diese reichen von projektübergreifendem Wissen bis hin zum konkreten Programmcode-Level. Domänenübergreifend haben sich in der Praxis hierfür bereits geeignete Methoden wie kontinuierliche Code Reviews und Pair Programming etabliert.

Im Projektalltag sind neben dem geteilten Wissen jedoch auch die Speziali-



Effizienz-Steigerung, Einarbeitung von Kollegen - Mob Programming bietet viele Vorteile für ein Unternehmen.

sierungen der einzelnen Teammitglieder wertvoll und notwendig. Hier setzt die Methode des Mob Programmings an. Sie verspricht gemäß des Slogans „All the brilliant people working on the same thing, in the same space, on the same computer" Lösungswege durch die Bündelung des gesamten, im Team verteilten Know-hows auf eine konkrete Aufgabe. Indem die Teilnehmer eine Aufgabe gemeinsam bearbeiten, verbreitet sich direkt und fortlaufend das Wissen sowohl über den Problem als auch über den Lösungsraum. Dazu gehören neben der jeweils gewählten Lösung auch die zugrundeliegenden Abwägungen sowie generelle Konzepte und Herangehensweisen. Durch die gemeinsame Arbeit werden zudem die vorhandenen Freiheitsgrade zur Lösung der jeweiligen Aufgabe häufig deutlicher als bei reinen Pair-Programming-Szenarien.

Working on the same thing

Das grundlegende Prinzip des Mob Programmings besteht darin, an eine Aufgabe gemeinsam im Plenum – dem namensgebenden „Mob" – heranzugehen. Dieser Mob umfasst das gesamte Entwicklungsteam. Dieses Team nutzt einen einzelnen Computer via Beamer oder TV-Bildschirm als zentrale Arbeitsplattform. Um diesen Computer versammeln sich alle Teilnehmer, diskutieren, programmieren und entwickeln so im Hands-on-Format den für die Aufgabenstellung angemessenen Lösungsweg.

Zwei Teammitglieder nehmen hierbei spezielle Rollen ein. Der Driver bedient die Programmierumgebung und übersetzt die Ideen des Plenums in konkreten Programmcode. Der Navigator leitet ihn dabei an und kanalisiert als Zwischenschritt die Diskussionen des Plenums in direkt programmiertechnisch umsetzbare Ergebnisse. Ziel ist, die Anleitung so abstrakt wie möglich zu halten, sodass der Driver sie ohne größere eigene Interpretationen flüssig in Programmcode umsetzen kann.

In the same place

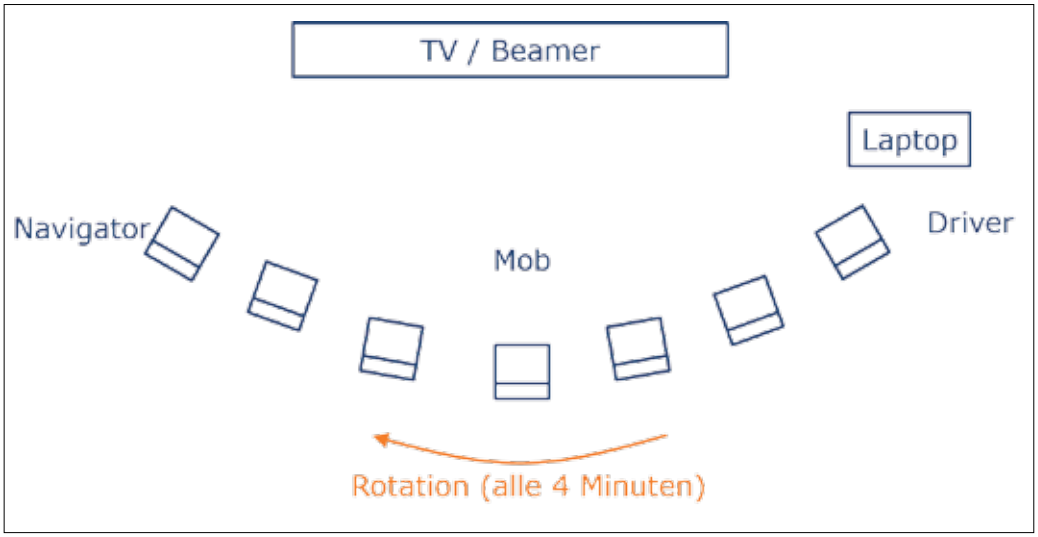
Die Teammitglieder verteilen sich im Halbkreis um den Beamer. Dem Driver und Navigator stehen dabei die beiden äußeren Sitzplätze im Mob zu. Alle Beteiligten besetzen diese beiden Rollen im Wechsel. Hierzu rotieren sie nach Ablauf eines vorgegebenen Zeitintervalls einen Sitzplatz weiter. Ziel ist es, Aufmerksamkeit, Konzentration und Dynamik des Mobs zu fördern. Dazu wird ein kurzes Rotationsintervall von vier bis sechs Minuten gewählt. Längere Intervalle bis zu fünfzehn Minuten bieten sich an, wenn codierungsfreie Diskussions- und Designphasen zu erwarten sind.

On the same computer

Für den Erfolg des Mob Programmings spielt die Umgebung eine wichtige Rolle. Alle Mob-Teilnehmer müssen eine bequeme, blendfreie Sicht auf den Beamer haben und ohne räumliche Hindernisse häufig rotieren können. Nachdem alle Teilnehmer ein- und denselben Computer nutzen, kann es sich lohnen, Zeit für eine für alle zufriedenstellende Konfiguration zu investieren. Gegebenenfalls müssen mehrere unterschiedliche Mäuse und Tastaturen bereitgestellt werden.

Praxistipps

Anwendbar ist die Methode ab einer Teamgröße von drei Personen. Als praktikable Obergrenze hat sich eine Mob-Größe von zehn Personen herausgestellt. Ist



Der Mob gruppiert sich um die zentrale Arbeitsplattform und arbeiten dort an einem einzelnen Computer via Beamer oder TV-Bildschirm.

der Mob größer, wird es mühsam und zeitintensiv, gemeinsam zu einer Entscheidung zu gelangen.

Neben den fachlichen Spezialisierungen berücksichtigt das Mob Programming auch die persönlichen Leistungskurven der Entwickler über den Tagesverlauf. Es besteht jederzeit die Möglichkeit, sich besonders aktiv in den Mob einzubringen oder auch ihn für eine kurze Pause zu verlassen. Gleiches gilt für Projektleiter, die nicht direkt an der Codierung beteiligt sind, oder für Vertreter benachbarter Teams – auch diese Stakeholder können sich zeitweise der Runde anschließen oder beratend zur Seite stehen. Hierdurch lassen sich Schwierigkeiten und Verständnisfragen direkt adressieren, sodass lange Kommunikationswege vermieden werden.

Eine sinnvolle Ergänzung kann die Rolle des Facilitators darstellen. Das gilt insbesondere, wenn sich ein Mob neu formiert. Der Facilitator wird zu Beginn aus den Reihen des Mobs gewählt und unterstützt richtigen Anwendung der Methode. Zu den weiteren Aufgaben des Facilitators gehört es, alle Mitglieder des Mobs aktiv in die Diskussion einzubeziehen. Dadurch erhält jeder die Chance, sein Wissen und seine Fähigkeiten in die Lösungsfindung einzubringen. Auch organisatorische Belange können in der Verantwortung des Facilitators liegen: Er initiiert Mob Programming Sessions, sucht passende Räumlichkeiten, stellt das benötigte Equipment bereit und moderiert Retrospektiven.

Mob Programming gegen Effizienz?

Das räumliche Set-up beim Mob Programming legt den Fokus auf den Beamer. Dies verleitet dazu, schnell mit einer zumindest prototypischen Codierung zu beginnen. Gerade bei innovativen und herausfordernden Aufgabenstellungen sollte jedoch durch vorgelagerte Diskussionen sichergestellt werden, dass alle Teilnehmer das gleiche Verständnis von Ziel und Randbedingungen haben. Dafür können im Vorfeld mehrere Zeitintervalle reserviert werden. Auch während einer bereits laufenden Mob-Programming-Session bietet es sich an, codierungsfreie Intervalle einzuschieben. Die entsprechende Planung des Vorgehens obliegt dem Teammitglied, das zum jeweiligen Zeitpunkt die Rolle des Navigators innehat.

Das gesamte Team auf nur eine Aufgabe zu fokussieren, mag aus Effizienzsicht zunächst abschreckend wirken. Wäre es nicht produktiver, das mehrköpfige Team mit mehreren Aufgaben gleichzeitig zu betrauen? Bestehen die zu bearbeitenden Aufgaben aus leichten und zugleich langwierigen Tätigkeiten, kann das sicher richtig sein. Bei komplexeren Fragestellungen ist es hingegen einfacher, im Mob Programming eine Lösung zu finden, da alle Teammitglieder mit ihren persönlichen Fähigkeiten einbezogen sind.

Das während des Mob Programmings geteilte Wissen und die gemeinsamen Diskussionen führen nicht nur zu einer Shared Code Ownership, sondern auch zu einer Shared Solution Ownership. Diese vereinfacht die Bearbeitung der nachfolgenden Aufgaben und stärkt zudem den Zusammenhalt des Teams. Außerdem wird durch das Mob Programming auch die Produktqualität gesteigert - schließlich sind in die Lösungen die Fähigkeiten des gesamten Teams gleichermaßen eingeflossen. Zusätzlich dazu hat bereits ein implizites Code Review durch den Mob während der Implementierung stattgefunden. ☐