Sonderdruck aus OBJEKTspektrum 02/2018

Kollaborative Testmethoden

Gemeinsam statt einsam

Kollaborative Methoden helfen Menschen in Teams, die Perspektive ihrer Teamkollegen auf die gemeinsame Software besser zu verstehen. Die Kommunikation über konkrete Aspekte des Sprint-Ergebnisses wird verbessert [Obe16]. In der agilen Welt haben sich kollaborative Ansätze wie Paarprogrammierung deswegen längst durchgesetzt. Und auch im Testing etablieren sich kollaborative Methoden zusätzlich zu bestehenden Ansätzen wie Testautomatisierung oder zu explorativen Testmethoden.

Kollaboration bedeutet Zusammenarbeit mehrerer Beteiligter, um ein gemeinsames Ziel zu erreichen. Kollaboration in der Softwareentwicklung oder im Testing führt Situationen herbei, in denen unterschiedliche Kenntnisse und Blickwinkel der Mitarbeiter auf die zu testende Software sichtbar werden. Durch diese Form der Zusammenarbeit kann die Kommunikation im Team verbessert und ein gleichmäßiger Kenntnisstand bei allen Projektmitgliedern erreicht werden.

Wenn Menschen Software gemeinsam testen, reden sie zudem sehr konkret miteinander: Die Kommunikation über konkrete Aspekte des Sprint-Ergebnisses ist unaufwendiger als das Sprechen über abstrakte Architekturkonzepte oder die Beschreibungen von Dialogoptionen.

Gemeinsames Tun zur selben Zeit und am gleichen Ort kann das Zusammengehörigkeitsgefühl des Teams stärken. Diese Form wird auch als synchrone Kollaboration bezeichnet; sie sorgt für wissensintensive Arbeitsbeziehungen. Durch die Kollaboration der unterschiedlichen Rollen erfolgt der Wissensaustausch über und mit dem Software Testing.

Das gemeinsame Anwenden einer kollaborativen Testmethode löst für die Teilnehmer den sogenannten „Group-Thinking-Effekt“ aus: Der einzelne Teilnehmer wird mutiger und experimentierfreudiger, als er es alleine wäre. Die Teilnehmer bauen zudem auf den Ideen ihrer Kollegen auf. Dadurch entstehen ungewöhnliche Ideen, was man testen könnte.

Die genannten Effekte – verbesserte Kommunikation, Wissen teilen, Team-Spirit aufbauen, Group-Thinking-Effekt auslösen – können sehr vielfältig sein. Je nach Teamzusammensetzung, angewandter Methode und verfolgtem Ziel werden sie unterschiedlich stark ausgeprägt sein.

Wir haben unsere Erfahrungen mit drei kollaborativen Testmethoden ausgewertet:

- Im *Pair Testing* arbeiten zwei Tester, oder ein Tester und ein Entwickler, in festen Rollen gemeinsam und gleichzeitig an einem Problem.
- Bei einer *Bug Bash* (auch Tester-Party genannt) nutzt man die Dynamik, die im Wettbewerb von Gruppen entsteht.
- *Mob Testing* nutzt Effekte der beiden anderen Methoden: Aus der Paarprogrammierung leiht sie sich die Methodik, aus der Bug Bash übernimmt sie den Impuls, die Dynamik von Arbeit in der Gruppe zu nutzen.

Wissen verteilen mit Pairing

„*For an idea to go from your head to the computer, it must go through someone else's hands.*“ (Llewelyn Falco, [Fal14])

Eine Softwaretesterin, nennen wir sie Natascha Wollner, arbeitet seit fünf Jahren bei einem deutschen Autobauer, die letzten beiden Jahre in einem agilen Team. Nun plant sie ein Sabbatical von sechs Monaten. Als einzige Testexpertin in ihrem Team übergibt sie ihre Aufgaben vorher an den neuen Kollegen Daniel – eine große Aufgabe: Daniel hat nur wenige Wochen Zeit, sich die Fachlichkeit des Projekts, Test-Artefakte und das Test-Design anzueignen.

Natascha überlegt, wie sie die Einarbeitung gestaltet. Sie erinnert sich an einen Beitrag über Paarprogrammierung [Fal14]: Der Autor erklärt, wie zwei Entwickler gemeinsam ein Problem lösen. Dafür nehmen sie feste Rollen ein: Es gibt einen *Fahrer* (im englischen „Driver“) und einen *Navigator*.

Der Navigator gibt dem Fahrer Anweisungen fürs Fahren – in der Softwareentwicklung meint das das Umsetzen der Lösung. Er spricht auf dem höchsten Abstraktionsgrad, der für den Fahrer nach-

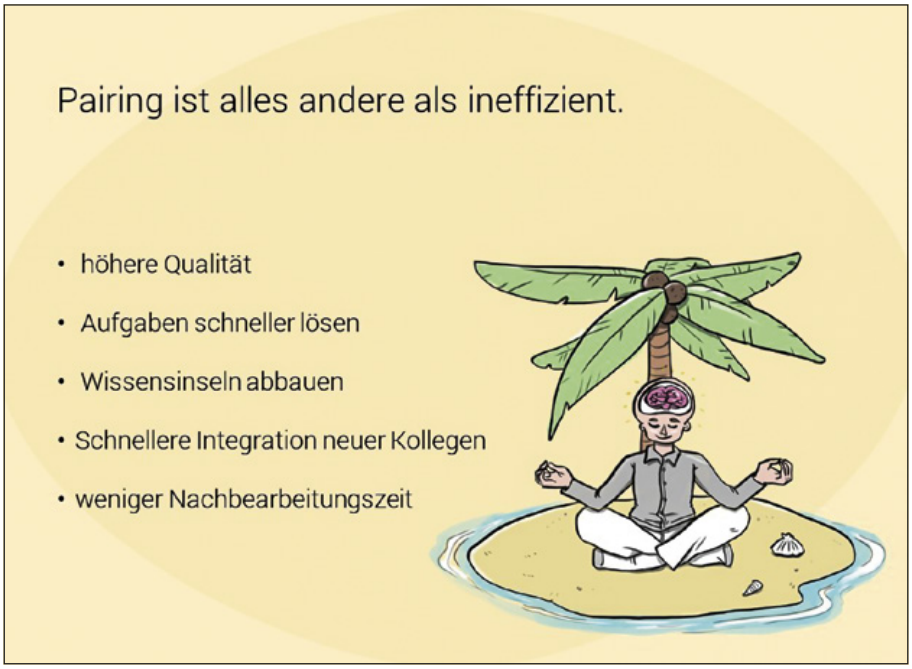

Abb. 1: Pair Testing verteilt Wissen zwischen Teammitgliedern

vollziehbar ist. Beim Programmieren setzt er die abstrakten Anweisungen in konkreten Code um.
Der Fahrer verlässt sich auf den Navigator und geht eine Lösung mit, auch wenn er den Lösungsweg zunächst nicht komplett nachvollzieht. Auf diesem Weg lernt der Fahrer vom Navigator.
Natascha passt diese Methode auf die Testing-Aufgaben in ihrem agilen Projekt an. Sie entscheidet sich dafür, Daniels Einarbeitung am Inkrement des aktuellen Sprints festzumachen. Zunächst wendet Natascha das Fahrer-Navigator-Prinzip an, wie der Blogbeitrag es beschreibt: Sie führt Daniel durch das Testing des aktuellen Sprint-Inkrements; nebenher erläutert sie Projekt, Test-Setup und Abläufe im Scrum-Team.
Nach den ersten beiden Wochen interpretieren die beiden Tester ihre Rollen um: Der Fahrer arbeitet selbstständiger; während der Eingabe kommentiert er seine Testideen und begründet sie. Die Navigatorin beobachtet die Eingaben und gibt Tipps oder bietet Hilfe an. Sie geht also von der steuernden Navigator-Rolle in einen Review-Modus. Fahrer Daniel kann weiterhin Fragen stellen, wenn etwas unklar ist. Er versteht die zu testende Anwendung Schritt für Schritt immer besser. Das zweite Augenpaar beugt Fehlern vor.
Das Fahrer-Navigator-Prinzip sagt im Kern: „For an idea to go from your head to the computer, it must go through someone else's hands" [Fal14]. Dahinter steckt die Erkenntnis, dass man Wert und Sinnhaftigkeit der eigenen Lösung hinterfragt, wenn man die Idee erklärt und eine andere Person sie umsetzt.
Das hier geschilderte Szenario von Natascha und Daniel ist fiktiv, aber erprobt. Pairing als kollaborative und explorative Testmethode funktioniert in einem festen Eins-zu-eins-Setting. Genauso gut eignet sich die Methode, um Wissen zwischen mehreren Kollegen zu verteilen – zum Beispiel, wenn sich ein agiles Tiger-Team auf mehrere Teams verteilt [Nie17].
Damit das Wissen im Team gleich verteilt ist, sollten die Paarkonstellationen regelmäßig wechseln. Bei regelmäßigem Wechsel wird das Pairing so lange durchgeführt, bis die Teammitglieder auf einem Stand sind. Nebenher bekommt man einen sehr guten Einblick in die Arbeitsweise der Kollegen im gleichen Team.
Eine andere Einsatzmöglichkeit ist die Ausbildung von Teamkollegen für spezielle Tools oder Methoden: Wir haben zum Beispiel Tester und Entwickler im Pair Testing zusammengebracht, um das Wissen über Testbarkeit von Code oder die Testabdeckung zu vermitteln.

Anspruchsvolle Aufgaben besser lösen

Pair Testing eignet sich zudem, um besonders anspruchsvolle Aufgaben zu bearbeiten. Hier kommt der beschriebene Vorteil zum Tragen: Der Fahrer konzentriert sich darauf, die angeforderte Lösung im Detail möglichst gut umzusetzen. Der Navigator hat dagegen alle Aspekte im Blick, die später wichtig werden könnten, denkt an die nächsten Schritte und behält den Überblick über das große Ganze. Jeder der Partner denkt so konsistent auf seiner Abstraktionsebene. Das gemeinsame Ergebnis wird besser, als wenn eine Person ständig zwischen konkretem Doing und übergeordnetem Plan wechselt. Das gilt für zwei Tester wie für zwei Entwickler.
Gerade in diesem Szenario ist es wichtig, Langeweile oder einen festen Trott zu vermeiden. Deswegen wechseln Testpartner die Rollen im Zweier-Team regelmäßig. Je nach Konstellation rotiert man nach jeder Testidee, oder man tauscht nach einem festen Zeitplan. In einem unserer Projekte haben wir zum Beispiel alle vier Minuten die Rollen gewechselt. Wenn die Projektmitglieder sich an die Methode gewöhnt haben, entwickeln sie in der Regel ihren eigenen Rhythmus.
Auf den ersten Blick wirkt Pair Testing ineffektiv, arbeiten doch zwei Menschen an einem Problem, das sonst einer allein löst. Legt man die Erfahrung mit Paarprogrammierung zugrunde, ist das Gegenteil der Fall: Pairing spart Zeit, da die Nachbearbeitung vieler Aufgaben wegfällt [Nos98]. Demnach ist die Qualität, Lesbarkeit und Funktionalität von Softwarecode, der mit Paarprogrammierung entwickelt wurde, besser und hochwertiger als solcher, der von Individualentwicklern erstellt wurde. Ebenfalls sei das Vertrauen in den Code bei Pairing-Teilnehmern höher.
Diese Annahme wird von einer Studie an der University of Utah belegt [Wil00]: 41 Studenten sollten in sechs Wochen vier Programme entwickeln. 28 Teilnehmer arbeiteten mit Paarprogrammierung, die anderen entwickelten alleine und bildeten die Kontrollgruppe. Gemessen wurde die Qualität an der Länge des entstandenen Programms, an den bestandenen Tests und an der Arbeitszeit, die benötigt wurde. Die Studie belegt, dass die Qualität der Lösungen durch paarweise Entwicklung tatsächlich höher war als die der Solo-Entwickler. Ähnliche Effekte beobachten wir beim Pair Testing im Projekt.
Diese Vorteile erklären auch, warum Paarprogrammierung und Pair Testing bei einigen Projekten über die ganze Projektdauer einen festen Platz im Methodenkoffer hat: Sie profitieren von höherer Codequalität, sie lösen gerade anspruchsvolle Aufgaben schneller in besserer Qualität, es fallen weniger Nachbearbeitungen an, und das Wissen ist zwischen den Teammitgliedern verteilt.
Wir haben fürs Pair Testing gute Erfahrungen damit gemacht, Arbeitstage in Pairing-Zeit und autonome Zeit aufzuteilen: Am Vormittag arbeiteten die Projektmitglieder regulär im Projekt, sie beantworten E-Mails, treffen Absprachen und erledigen alle Aufgaben, für die in einer Pairing-Situation wenig Gelegenheit ist. Am Nachmittag treffen sich Paare an einem Laptop und testen als Fahrer und Navigator.
Diese Aufteilung durchzusetzen – und zwar in Kalendern und Köpfen – braucht Gewöhnung. Dafür braucht es die Rückendeckung der Projektleitung. Schließlich müssen Regeltermine des Scrum-Teams, geplante Meetings und kurze Absprachen am Vormittag stattfinden. Am leichtesten ist das, wenn es mehrere Pairing-Teams im Scrum-Team gibt – dann ergibt sich die Tagesaufteilung von selbst.
Wenn Pair Testing länger als ein paar Tage auf der Agenda steht oder gar zur festen Einrichtung im Team wird, sollten zudem die Umgebungsbedingungen systematisch aufgesetzt werden: Ein Raum, in dem das Pairing-Duo sich ungestört austauschen kann – und niemand anderen stört –, ist ein Muss. Ein großer Bildschirm ist ebenfalls hilfreich. Und last but not least: Beim Pairing lernt man seine Kollegen gut kennen. Das gefällt nicht allen. Möglicherweise funktioniert nicht jede Zweierkonstellation. Dann sollte man den Mut nicht aufgeben, sondern überlegen, ob eine Umbesetzung möglich ist.
Wir haben in unseren Pair Testings festgestellt, dass der Gruppenzusammenhalt stärker wurde. Neue Kollegen lernten schnell viel über Software und Projekt; sie profitierten vom Wissen der anderen Projektmitglieder; und sie wurden schneller ins Team integriert. Diese Kombination macht Pair Testing aus meiner Sicht sehr effizient: Teams erreichen mit wenig Aufwand viele Effekte.

Bug Bash motiviert Teams zum Testen

"All the brilliant people working on the same thing, at the same time, in the same space, on the same computer." (Woody Zuill)
Szenenwechsel: Timo und Alex wollen die Usability für das neu entwickelte Zeiterfassungssystems in ihrer Firma testen.

Crowd Testing fällt nach gründlicher Sicherheitsabwägung aus – selbst mit Testdaten würde man externen Usern zu viel Einblick ins eigene Unternehmen geben.

Alex hat bei ihrem vorherigen Arbeitgeber gute Erfahrungen mit einer Bug Bash gemacht: Dabei finden Teams in einer konzentrierten Test-Session möglichst viele Fehler. Die Teams treten gegeneinander an; das spornt die Teilnehmer an, viele Fehler zu finden. Die Gruppen haben dabei vollen Freiraum, die Software explorativ zu erkunden. Gefundene Fehler werden durch bereitgestellte Tools aufgenommen. Die Bandbreite der möglichen Werkzeuge reicht von ausgedruckten Handzetteln über Notizen auf einer Intranetseite bis zu JIRA. Am Ende wertet die Jury die gefunden Fehler aus. Die Teams mit den meisten gefunden Fehlern sind die Gewinnerteams. Die Gewinner werden bei der Siegerehrung ausgezeichnet. Die Methode ist auch als Tester-Party bekannt.

Alex und Timo planen einen Nachmittagstermin von 15 Uhr bis 18 Uhr und laden ihre Kollegen aus der Fachabteilung ein. Bei 30 Einladungen sagen 20 Teilnehmer zu. Das Entwicklerteam ist ebenfalls vor Ort und macht sich ein Bild von gefundenen Fehlern. Nun geht es an die Vorbereitung: Die beiden Tester kaufen Snacks und Getränke für das Ende der offiziellen Testphase um 17 Uhr. Während die Teilnehmer essen, wertet die Jury die Ergebnisse der Teams aus, so der Plan. Für die Siegerehrung bestellen sie Medaillen.

Am Tag vor der Bug Bash installieren sie den aktuellen Entwicklungsstand der Zeiterfassung in einer Testumgebung, und spielen anonymisierte Testdaten ein. Die Test-Accounts bekommen unterschiedliche Rechte. So kann jede Gruppe als Geschäftsführer, Projektleiter und Mitarbeiter testen. Sie bestücken den Meeting-Raum mit kleinen Tischen mit je vier Stühlen. Dadurch ergeben sich automatisch Teams.

Zu Beginn stellen Alex und Timo das Verfahren vor: Die Teilnehmer testen 90 Minuten lang wild drauf los. Sie protokollieren alle Fehler auf vorgedruckten Handzetteln. Die Software stellen Alex und Timo nicht vor. Damit würde der Blick der Teilnehmer gelenkt. Die Jury besteht aus den Kollegen des Entwicklungsteams sowie Alex und Timo selbst. Ihre Aufgabe ist, die Bug-Reports nach der Session einzusammeln und die gefundenen Bugs auszuwerten.

Party mit Nachwirkung

Am Nachmittag erschienen 18 Kollegen zur Bug Bash, die in sechs Dreiergruppen aufgeteilt werden. Die Stimmung ist gut, die Teilnehmer testen ungehemmt drauf los. Während der Auswertung unterhalten sich die Teilnehmer weiter. Bei der Siegerehrung um 17.45 Uhr werden drei Teams ausgezeichnet: Das Team mit den meisten Fehler findet zwölf Bugs, auch die anderen Teams kommen auf sechs bis zehn Fehler. Zusätzlich vergibt die Jury Medaillen an das Team mit dem kreativsten Fehler und für den kritischsten Fehler.

Am nächsten Tag bittet Timo per E-Mail um Feedback zum Format. Konkret fragt er, wie den Teilnehmern die Bug Bash gefallen hat, wie sie den Termin und die Gruppengröße fanden. Viele Teilnehmer antworten, dass sie Spaß hatten, einige loben den Termin am Nachmittag innerhalb ihrer Arbeitszeit. Andere hätten einen Abendtermin bevorzugt: Sie versprechen sich eine lockere Atmosphäre. Hier ist es am besten, sich an die Gepflogenheiten im eigenen Unternehmen zu halten. Einige Teilnehmer hätten sich etwas größere Teams gewünscht. Alex und Timo beschließen, beim nächsten Mal mit Vierer- oder Fünfer-Teams zu arbeiten.

Ein weiteres hilfreiches Feedback: In jeder Gruppe sollten Teilnehmer mit mehr und mit weniger Wissen über die Software gemischt werden. Der Hinweis kam aus einer Gruppe, die es schwierig fand, Fehler zu erkennen; sie hatten sie oft als Anwenderfehler interpretiert.

Die Veranstalter sind zufrieden mit dem Ergebnis: Die 18 Teilnehmer haben in relativ kurzer Zeit eine Menge Fehler gefunden, die sie mit ihren Kollegen aus dem Entwicklungsteam auswerten können. Auswerten bedeutet, tatsächliche Fehler und unerwartete Verhaltensweisen zu identifizieren. Diese können die Entwickler vor dem Live-Gang beheben. Durch Beobachten der Teilnehmer, die das Zeiterfassungssystem noch nicht kennen, haben sie zudem neue Blickwinkel auf die Software gewonnen. Und: Der Aufwand für eine Tester-Party – eine längere Meetingdauer, Essen und Getränke, Medaillen – lohnt sich nicht für jedes Release. Es bietet sich an, diese Methode als Highlight einzusetzen, zum Beispiel für Usability-Tests mit dem Fachbereich.

Abb. 2: Eine Bug Bash, auch Tester-Party genannt, bringt viele Stakeholder an einen Tisch

Mob Testing schult Key-User

Sales-Manager Justus Kreibel öffnet die Tür zum Meetingraum im zweiten Stock – und bleibt erstaunt stehen: Er erwartet die Powerpoint-Präsentation über die neuen Funktionen im CRM-System, die regulär alle drei Monate stattfindet. Statt der Frontalbeschallung arbeiten seine Kollegen gemeinsam an einem Laptop.

Testmanagerin Elisabeth Bauer kommt auf ihn zu: „Heute findet ihr gemeinsam die neuen Features heraus. Dafür haben wir uns was Neues ausgedacht: Mob Testing.“ Schnell erklärt sie ihm die Regeln: Eine Gruppe von mindestens drei Personen arbeitet gemeinsam an einem Rechner, zusätzlich gibt es pro Gruppe einen Facilitator. Die Gruppenmitglieder wechseln durch festgelegte Rollen: Der Navigator sagt, was er testen möchte und wie das erreicht werden soll. Der Fahrer setzt

die Ideen des Navigators um. Dabei darf er oder sie Verständnisfragen stellen; das Vorgehen des Navigators darf er jedoch nicht hinterfragen. Die anderen Teilnehmer sind die Mob-Mitglieder: Sie beraten den Navigator zu seinen Test-Ideen, geben Tipps oder stellen Fragen, wenn etwas unklar ist. Nach vier Minuten wechseln die Rollen.

Elisabeth Bauer achtet als externer Facilitator auf die Zeiten. Sie protokolliert die Session und nimmt gefundene Abweichungen und Irritationen auf. Sie wird diese später mit dem Entwicklungsteam besprechen. Außerdem achtet sie darauf, dass sich alle Teilnehmer an ihre Rolle halten. Justus setzt sich als Teil des Mob dazu. Gleich beim nächsten Rollenwechsel wird er Navigator.

In dieser Rolle möchte er ein neues Unternehmen im System anlegen. Bisher ging das nur umständlich über das Menü. Eine neue Funktion ermöglicht es ihm nun, den Datensatz direkt aus der Kontaktmaske heraus zu erstellen; dabei kann er sogar bei Google direkt nach der Adresse forschen. Er führt seine Kollegin Antje durch seinen Fall hindurch. Seine Kollegen im Mob stellen ihm zudem Fragen über seine Testidee, dadurch tauschen sie sich über ihre Erfahrungen mit dem System aus.

Nach weiteren vier Minuten wird er zum Fahrer und bekommt Anweisungen von Heinz, dem neuen Navigator. Heinz möchte die Umsatzsteuernummer eines Debitoren eintragen. Schnell stellt er fest, dass er die Nummer ja bei allen Rechnungsempfängern des Unternehmens ändern muss, nicht nur beim Unternehmensdatensatz selbst. Nun weist er Justus an, die Daten in den nächsten Datensatz zu kopieren, dann in den nächsten. Nach dem fünften Kontakt-Aufruf fängt das System an zu stottern und zeigt Latenzen. Dann sind die vier Minuten schon wieder vorbei und Justus wechselt zurück in den Mob.

Am nächsten Tag berichtet er seinen Kollegen beim Mittagessen von der Session: Er hat die Arbeitsroutinen seiner Kollegen kennengelernt, da alle den eigenen Fokus, eigene Ideen und Vorerfahrungen eingebracht haben. Auch er als alter Hase hat noch Neues im System entdeckt. Das begeistert ihn. In der Rolle des Fahrers für Heinz musste er sich allerdings sehr zusammenreißen: Er hätte die Änderung der Umsatzsteuernummer über die Funktion „Adressdaten auf alle Kontakte übertragen“ ausgerollt; als Fahrer durfte er das Vorgehen des Navigators – hier Kopieren der Nummer und manuelles Einfügen – aber nicht hinterfragen. Im Nachhinein ist er froh über die Regelung: Nur dadurch hat sein Team herausgefunden, dass das CRM-System beim zu schnellen Aufrufen von Kontakten ins Stocken gerät. Das ist ein wichtiges Feedback an die Entwickler. Die neuen Funktionen des Releases hat er übrigens ganz nebenbei kennengelernt – und mit deutlich mehr Spaß als in einer Powerpoint-Vorstellung.

Fehler finden und Wissen vermitteln

Mob Testing basiert wie das oben beschriebene Pair Testing auf dem „Strong Style Pair Programming“-Pattern. Im Unterschied zur Paarprogrammierung ist der Fahrer in einer rein ausführenden Rolle: Er tippt die Anweisungen des Navigators ein und stellt sie nicht infrage.

Damit führt der Navigator letztendlich die Tests durch. Der Navigator muss auf einer sehr hohen Abstraktionsebene sprechen; so kann jeder dem Thema folgen. Die Mob-Mitglieder überprüfen Anweisungen des Navigators. Sie helfen ihm, wenn es notwendig ist, und geben ihm Einblicke in die jeweiligen Kenntnisse. Diese Rollen werden nach der vorgegebenen Zeit neu verteilt.

Es bietet sich an, einige Testideen für Neulinge vorzubereiten und als ersten Navigator eine Person zu wählen, die die Software gut kennt. Der Facilitator kann den Navigator ermuntern, seine Testidee Schritt für Schritt zu erklären. So finden die Teilnehmer schnell in den Arbeitsmodus.

Das Zeitmanagement übernimmt der Facilitator; diese Rolle wechselt als einzige nicht durch. Er erläutert zu Beginn einer Session Abläufe, Grundsätze, Spielregeln und die zu testenden Aufgaben und greift ein, wenn über ein Thema zu lange diskutiert wird – und stärkt die Autonomie des aktuellen Navigators. Der Facilitator springt für Teilnehmer ein, wenn diese zum Beispiel kurz telefonieren. Und schließlich protokolliert er den Verlauf der Session, Schwierigkeiten oder gefundene Fehler.

Die Anzahl an Personen in einem Mob umfasst zwischen drei und zwölf Mitglieder. So bleibt das Team produktiv [Zui14]. Bei zwei Mitgliedern ist es Pair Testing; bei mehr als zwölf Teilnehmern rotiert nicht mehr jedes Mitglied durch alle Rollen hindurch. Außerdem wird die Kommunikation unübersichtlich und Teilnehmer können sich in der großen Gruppe verstecken.

Wir setzen Mob Testing in zwei Szenarien ein: Fehler finden und Wissen vermitteln. Der große Vorteil für das Fehlerfinden ist die Balance zwischen kreativ sein und gleichzeitig Wissen und Ideen strukturiert teilen. In einem unserer Mobile-App-Projekte testen die Teammitglieder zum Beispiel neue Funktionen parallel auf Android und auf iOS. Dadurch fallen Versionsunterschiede zwischen den beiden Plattformen schnell auf und werden systematisch erfasst. Nebenher kennt jedes Teammitglied immer alle neuen Features. In Schulungskontexten zahlt sich der Vorteil aus, dass Teilnehmer Gelerntes sofort in der Gruppe ausprobieren können. Durch den Austausch über die Versuche

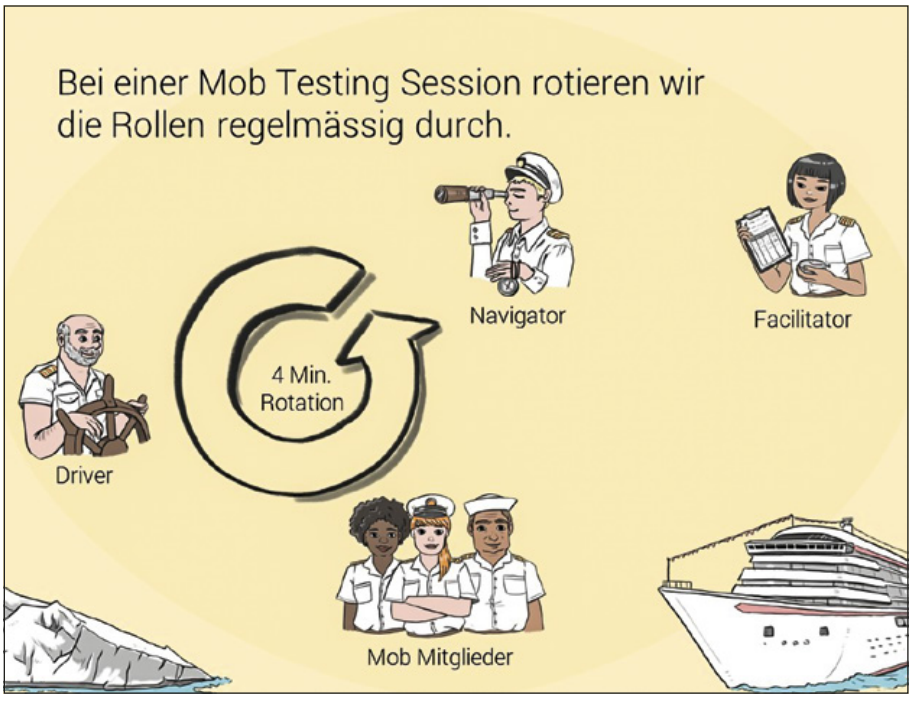

Abb. 3: Beim Mob Testing lernen Stakeholder die neuen Funktionen in der Gruppe kennen

Literatur & Links

[Fal14] L. Falco, Llewelyn's Strong-Style Pairing, 2014, siehe: **http://llewellynfalco.blogspot.de/2014/06/llewellyns-strong-style-pairing.html**

[Nie17] Ch. Niemann, T. Becker, Sieben Lösungspatterns für große agile Projekte, in: OBJEKTspektrum, 2/2017, siehe auch: **https://www.sigs-datacom.de/fachzeitschriften/objektspektrum/archiv/artikelansicht/artikel-titel/scrum-mit-zwei-drei-vier-teams-sieben-loesungs-patterns-fuer-grosse-agile-projekte.html**

[Nos98] J. T. Nosek, The Case for Collaborative Programming, 1998, siehe: **https://www.researchgate.net/publication/27295641_The_Case_for_Collaborative_Programming**

[Obe16] K. Obermüller, J. Campbell, Mob Programming – the Good, the Bad and the Great, The official meltwater engineering blog, 2016, siehe: **http://underthehood.meltwater.com/blog/2016/06/01/mob-programming/**

[Wil00] L. A. Williams, The Collarborative Software Process, Dissertation an der University of Utah, 2000, siehe: **https://collaboration.csc.ncsu.edu/laurie/Papers/dissertation.pdf**

[Zui14] W. Zuill, Mob Programming – A Whole Team Approach, agilealliance, ExperienceReport, 2014, siehe: **https://www.agilealliance.org/wp-content/uploads/2015/12/ExperienceReport.2014.Zuill_.pdf**

lernen viele Teilnehmer nachhaltiger und mehr als durch Einzelaufgaben.

Ein Team für jedes Anliegen

Im kollaborativen Testing nutzen Menschen leichtgewichtige Methoden, um Wissensinseln abzubauen, unterschiedliche Blickwinkel auf die Software zu entwickeln und durch den Group-Thinking-Effekt kreativere Lösungen zu finden.
Nach unseren Erfahrungen mit Pair Testing, Mob Testing und Bug Bashes spielen die Formate ihre Vorteile neben dem klassischen Testanliegen – Fehler finden – in der Wissensweitergabe im Team aus: Pair Testing eignet sich besonders gut zum systematischen und intensiven Wissenstransfer. Bei einer Bug Bash oder Tester-Party fühlen sich viele Teilnehmer durch ein kompetitives Setting angefeuert, das ihre Kreativität beim Fehlerfinden steigert.
Mob Testing verbindet die Effekte der beiden anderen Methoden: Aus der Paarprogrammierung leiht sie sich die Fahrer-Navigator-Methodik, aus der Bug Bash übernimmt es den Impuls, die Intelligenz der Gruppe zu nutzen. Damit eignet sich die Methode, wenn beide Anliegen wichtig sind: Bugs oder Unstimmigkeiten finden und Wissen im Team verteilen. ||

Der Autor

Benedikt Wörner
(benedikt.woerner@maibornwolff.de)
ist Agile Testing Expert und Test Coach bei MaibornWolff mit zehn Jahren Erfahrung im Testmanagement und Coaching. In seinen Projekten öffnet er neue Perspektiven für das Testing, derzeit zum Beispiel mit kollaborativen und explorativen Testmethoden.