Software testen im agilen Entwicklungsprozess

Permanente Kontrolle

Tilo Linz

Testen ist auch im agilen Entwicklungsprozess das wichtigste Instrument zur Qualitätssicherung von Software. Vor allem Systemtests sind hier unverzichtbar. Und das Erproben der Funktionen muss den Prinzipien des agilen Manifests folgen.

Zentrale Anforderung an agile Tester ist die Forderung nach schnellem Feedback. Alle Maßnahmen sind dahingehend ausgerichtet. An die Stelle sequenzieller Testphasen (mit langer Feedbackzeit) tritt kontinuierliches Testen in jedem Sprint unter parallelem Einsatz von Tests jeder Stufe: Test nonstop mit täglichem Feedback. Damit dies gelingt, sind folgende Erfolgsfaktoren entscheidend:

Testautomatisierung: Dauerhaft lässt sich schnelles Feedback nur erreichen und aufrechterhalten durch umfassende Testautomatisierung auf jeder Testebene – von Unit Tests über Integrationstests bis zu Systemtests. Dieses Netz aus automatisierten Tests erlaubt das kontinuierliche Refactoring von Programmcode und ist Voraussetzung für die sichere Anwendung dieser Clean-Code-Technik.

Exploratives Testen: Da es nicht möglich ist, jeden Testfall sofort zu automatisieren, muss die Testautomatisierung durch schnelles manuelles Testen ergänzt werden. Das Mittel dazu ist exploratives Testen. Es verzichtet weitgehend auf eine vorbereitende Testspezifikation und gibt dem Tester Freiraum, intuitiv zu arbeiten. Dadurch ist er in der Lage, kurzfristig neue oder geänderte Features zu überprüfen, auch wenn das Sollverhalten auf der Feature-Taskkarte nur knapp skizziert ist. Dies bedeutet, dass der Tester in der Lage sein muss, sich die nötige Zusatzinformation durch Kommunikation mit allen potenziellen Wissensträgern und Stakeholdern zu beschaffen . Nicht jeder kann das gleich gut, aber das lässt sich erlernen und üben.

Testwissen im Team: Die Verantwortung für das Testen liegt beim agilen Team. Es plant und steuert die Testaktivitäten in der gleichen Weise wie alle anderen Aufgaben im Sprint. Jedes Teammitglied kann und soll, abhängig von seinem Wissen, Testaufgaben übernehmen. Den Test als Teamaufgabe zu organisieren, setzt voraus, dass ausreichend Testerfahrung vorhanden ist. Ein klassisches Entwicklungsteam, das auf Scrum umstellt, besitzt sie in der Regel nicht ausreichend. Daher müssen externe Tester und solche aus der eventuell existierenden Systemtestabteilung in den Teams dauerhaft mitarbeiten. Das heißt, dass jetzt Experten, die sich ausschließlich um Testaufgaben kümmern, nicht mehr getrennt und unbeeinflusst testen, sondern als Mitglied des agilen Teams. Das unabhängige Testen mit getrennten Rollen und getrennter Organisationsstruktur entfällt.

Dieser durchaus risikobehaftete Schritt gelingt nur, wenn parallel sichergestellt wird, dass Testexperten im Team dabei sind und wenn der Scrum Master oder ein Testexperte die Aufgaben eines Testmanagers mitübernimmt. Aber auch wenn Experten im Team vorhanden sind, kann die enge Zusammenarbeit den Test schwächen, weil hauptamtliche Tester ihr Projekt immer mehr aus einer Entwicklerperspektive betrachten und Ergebnisse zunehmend weniger kritisch beurteilen, als ein unabhängiger Tester es tun würde.

Teamübergreifende Sicht: Wenn in großen Projekten mehrere Scrum-Teams (als Feature-Teams) parallel arbeiten, braucht es eine übergeordnete Sicht, die das korrekte Zusammenspiel der Features über die Scrum-Teams hinweg im Blick hat. Auch wenn jedes Team für sein Teilprodukt Unit Tests, Integrations- und Systemtests erstellt und durchführt, besteht die Gefahr, dass team- beziehungsweise featureübergreifende Testszenarien zu wenig Beachtung finden.

Als Gegenmaßnahme sollten sich (wie die Scrum Master im Scrum of Scrums) die Testmanager oder die hauptamtlichen Tester der Teams regelmäßig untereinander austauschen und koordinieren, zum Beispiel in einem „Scrum of Testers“. Als weitere Maßnahme kann es sinnvoll sein, ein übergeordnetes System-

Systemtests: Nahe an der Realität

Scrum verlangt, dass am Ende eines jeden Sprints ein potenziell funktionstüchtiges Softwareprodukt vorliegt. Damit es tatsächlich ausgeliefert werden kann, muss es außerhalb der Continuous-Integration-Umgebung lauffähig sein, über eine Bedienschnittstelle verfügen und meist mit anderen Applikationen und Systemen des Kunden zusammenarbeiten. Um diese Bedingungen zu prüfen, sind Testfälle nötig, die das System aus Nutzerperspektive und über dessen Bedienschnittstelle testen. Und es bedeutet, dass diese Tests in einer Testumgebung stattfinden müssen, die der späteren Einsatzumgebung nahekommt (siehe Kasten „Testumgebung: Teuer, aber notwendig“). Weder Unit Tests noch Integrationstests leisten das. Deshalb sind zusätzlich Systemtests notwendig. Sie prüfen Aspekte, die sonst nicht abgedeckt werden.

Die Systemtestfälle können grundsätzlich direkt aus den im Product Backlog formulierten Anforderungen und Akzeptanzkriterien oder aus eventuell zugehörigen, detaillierteren Use-Case-Beschreibungen abgeleitet werden. Das Team soll aber hier nicht haltmachen, sondern kritisch nach Lücken in diesen Dokumenten suchen und weitere Varianten und Sonderfälle sowie zusätzliche absehbare Use Cases identifizieren und in Form von Testfällen notieren. Mit dem Product Owner ist zu klären, ob und wo Anforderungen im Backlog aufgrund des Inputs der Tester zu präzisieren sind und ob zusätzliche Anforderungen im Backlog zu ergänzen sind.

Der Systemtest soll prüfen, ob das Produkt beim Endanwender funktionieren wird und nicht nur im Labor. Je realitätsnäher die Testumgebung, desto sicherer kann aus einem fehlerfrei laufenden Systemtest auf eine korrekte und reibungslose Funktion im Betrieb geschlossen werden. Umgekehrt erhöht eine zu einfache Testumgebung das Risiko, dass die Systemtests Probleme nicht aufdecken und die erst im Betrieb beim Kunden auftreten.

Produktionsnähe ist gefordert

Den Grad, mit dem die Testumgebung die benötigten Merkmale der späteren Produktivumgebung abbildet, bezeichnet man mit „Produktionsnähe“. Die Forderung nach weitgehender Produktionsnähe bedeutet, dass externe Bausteine, mit denen das Produkt in der Endanwenderlandschaft zusammenarbeiten muss, in der Testumgebung durch reale Komponenten oder durch realitätsnahe Simulationen solcher Komponenten dargestellt werden. Statt Testtreiber und Platzhalter sollten auf allen Ebenen die später tatsächlich zum Einsatz kommenden Hard- oder Softwareprodukte in der Testumgebung installiert werden (Rechner, Systemsoftware, Treibersoftware, Netzwerk, Fremdsysteme). Eine Systemtestumgebung kann deshalb aufwendig, komplex und teuer sein.

Die hohe Zahl zusammengeschalteter Komponenten bringt außerdem eine Vielzahl unterschiedlicher Aufbau- und Konfigurationsmöglichkeiten mit sich, in denen sich die Umgebung betreiben lässt. Bestimmte Konfigurationen sind dabei typisch für bestimmte Anwendergruppen, Einsatzszenarien oder Kompatibilitätseigenschaften des zu prüfenden Produkts.

testteam einzurichten (siehe Kasten „Systemtests: Nahe an der Realität“). Es stellt nicht nur sicher, dass es teamübergreifende Testszenarien gibt, sondern es kann gleichzeitig die Pflege der Testframeworks übernehmen und diese den anderen Scrum-Teams als Service zur Verfügung stellen.

Wie schon mehrfach erwähnt, werden Testaktivitäten in Scrum in der gleichen Weise geplant und gesteuert wie alle anderen Aufgaben innerhalb eines Sprints: über Tasks, die im Rahmen der Sprint-Planung vom Product Backlog ins Sprint Backlog und auf das Taskboard wandern. In Bezug auf Testaufgaben gilt es hierbei, folgende Punkte zu beachten:

Checkliste abarbeiten

Definition of Ready: Die DoR ist eine Checkliste, die bei der Erstellung der User Stories durch den Product Owner sowie bei deren Qualitätssicherung und spätestens bei der Übernahme von Stories vom Product ins Sprint Backlog Anwendung findet. Ob eine User Story „Ready“ ist, erkennt das Team, wenn es sie aus Testsicht betrachtet. Wenn es nicht gelingt, Testfälle zu entwerfen, oder unklar ist, wann ein Ergebnis als „passed“ oder „failed“ einzustufen wäre, bietet die Story offenbar zu viele Freiheitsgrade. Sie sollte als nicht „ready“ vom Team zurückgewiesen werden. Alternativ kann das Team die Lücken in der Story (im Sinne von Test First) füllen, indem es Testfälle ergänzt, die die fehlenden Aussagen liefern. Ein Pairing von Tester und Product Owner ist hier sinnvoll.

Definition of Done: Die DoD als weitere Checkliste beschreibt, welche Ziele das Team beim Umsetzen der Story erreichen muss, bevor sie als „fertig zur Vorlage im Sprint-Review“ gelten. Die DoD legt Testziele wie die notwendigen Testarten, die zu erreichende Testabdeckung und die Testende-Kriterien (im Allgemeinen die Entfernung aller gefundenen Fehler) fest. Sie dient damit unmittelbar der Sicherstellung der Produktqualität und der Kundenzufriedenheit.

Eine Testaufgabe lässt sich explizit als eigenständiger Task formulieren, aber auch implizit als Done-Kriterium eines Programmiertasks. Wenn alle nötigen Tests automatisiert vorliegen, ist die implizite Variante problemlos. Wenn eine Story oder ein Feature erstmalig getestet wird oder wenn Tests zu automatisieren sind, ist es ratsam, diese Aufgabe(n) explizit als separate Task(s) zu behandeln.

In den vergangenen zehn Jahren hat sich Softwaretesten zu einem professionellen Berufsbild entwickelt. Mit Test- und Qualitätssicherungsaufgaben werden heute in vielen Projekten speziell ausgebildete und zertifizierte Softwaretester betraut, sie leisten einen ganz wesentlichen Beitrag zum Projekterfolg.

Testen als unverzichtbare Disziplin

Viel Wissen über Softwaretest und Softwarequalitätssicherung ist angesichts immer komplexerer Softwaresysteme in jedem Team unverzichtbar, vor allem im agilen Projekt. Denn agiles Testen bedeutet nicht, dass weniger, laxer oder schlechter getestet wird als früher. Im Gegenteil, Testen erhält im agilen Projekt eine vorher nicht gekannte, zentrale Funktion: Testfälle definieren das System und lösen Spezifikationen ab (Test First), Tests werden umfassend automatisiert und kontinuierlich durchgeführt (Test Nonstop). Das Entwerfen guter Testfälle ist eine anspruchsvolle Tätigkeit, die ohne Ausbildung in entsprechenden Entwurfstechniken nicht gelingen kann. Testautomatisierung erfordert Know-how in der Programmierung und Toolkenntnisse. Analoges gilt für Codeanalyse, Continuous Integration, Reviewtechniken und so weiter.

Da Scrum die Rolle des Testers nicht explizit beschreibt und die Rolle des Testmanagers nicht erwähnt, sind leider oftmals Scrum-Teams anzutreffen, denen das nötige Wissen fehlt, um die Testaufgaben fachgerecht zu erledigen. Das ist riskant, denn Scrum verlässt sich stark auf die Wirksamkeit von Feedbackschleifen, und eines der wichtigsten Mittel, um in der Softwareentwicklung Feedback zum Produkt zu generieren, ist Testen.

Im interdisziplinären Team ist hiervon jedes Mitglied betroffen. Zumindest in den grundlegenden Testtechniken sollten daher alle Beteiligten, auch die Programmierer, geschult werden. Umgekehrt müssen Tester Grundlagen der Programmierung erlernen. Testautomatisierung und Codeanalyse-Aufgaben erfor-

Testumgebung: Teuer, aber notwendig

Der Systemtestaufwand skaliert nicht nur mit der Anzahl der Systemtestfälle, sondern auch mit der Anzahl abzudeckender Testumgebungskonfigurationen. Was bedeutet das für agile Projekte?

Stückliste und Konfiguration der Systemtestumgebung müssen bewusst geplant und definiert werden. Nur dann ist klar, welche Aussagekraft die Tests haben. Dabei ist abzuwägen zwischen Aussagekraft und Kosten der Umgebung. Das initiale Setup einer Systemtestumgebung kann aufwendig, teuer und fehlerträchtig sein. Mit Tasks zum Aufbau der Systemtestumgebung muss man sich schon in frühen Sprints beschäftigen. Wird das versäumt, läuft das Team Gefahr, zu spät belastbares Feedback aus den Systemtests zu erhalten.

In jedem Sprint muss das Team Tasks zur Pflege der Systemtestumgebung reservieren. Andernfalls veraltet die Testumgebung, was schwerwiegende Hindernisse mit sich bringt: Die eingerichteten Konfigurationen halten nicht mit der Entwicklung des Produkts Schritt und werden unbrauchbar und notwendige Konfigurationsvarianten werden nicht aufgebaut. Die Systemtests verlieren Sprint für Sprint an Aussagekraft oder sind gar nicht mehr ausführbar.

Wegen der vielen beteiligten, oft heterogenen Komponenten ist es schwer, Setup und Konfiguration der Systemtestumgebung zu automatisieren. Dennoch ist es ein lohnendes Unterfangen. Hier ist ebenfalls darauf zu achten, dass regelmäßig entsprechende Tasks eingeplant werden.

In den Retrospektiven sollte der Scrum Master immer wieder nach Möglichkeiten zur Optimierung der Testumgebung und insbesondere der Systemtestumgebung fragen. Allerdings sind die Optimierungsziele unterschiedlich: Unit- und Integrationstestumgebungen werden für eine maximal hohe Ablaufgeschwindigkeit der Tests optimiert. Jeder Testfall prüft einen kleinen Funktionsausschnitt unter einfachen Randbedingungen (etwa ein vorab festgelegtes Schaltkommando senden und in einer Datei aufzeichnen). Externe Bausteine sind dazu durch Platzhalter zu ersetzen. Natürlich ist im Systemtest eine hohe Ablaufgeschwindigkeit wünschenswert. Vorrangig ist hier jedoch, die Aussagekraft der Tests zu maximieren – im Zweifel zulasten der Testlaufzeit. Wegen dieser unterschiedlichen Ziele ist es notwendig, für die Systemtests eine eigene Umgebung aufzubauen.

Wegen hoher Kosten ist es in vielen Projekten unvermeidbar, dass die Systemtestumgebung dem Team nicht uneingeschränkt zur Verfügung steht, sondern mit anderen Teams oder anderen Produktentwicklungseinheiten geteilt werden muss. Eine solche Situation behindert die Taskplanung des Teams und führt unter Umständen dazu, dass das Team seine Systemtests zu bestimmten Zeiten durchführen muss. Sie sind vielleicht extern vorgegeben oder müssen zumindest mit anderen Teams koordiniert werden. Es ist Aufgabe des Scrum Master, die Sache zu koordinieren. Das Forum dazu sind die sogenannten Scrum of Scrums.

dern es. Die Anwendung all dieser Techniken muss eingeübt werden. Die Personalverantwortlichen im Unternehmen müssen die veränderten Anforderungen berücksichtigen und sollten gemeinsam mit dem Scrum Master und QM-Stab ein entsprechendes Aus- und Weiterbildungsprogramm für agile Teams zusammenstellen und anbieten.

Der internationale Maßstab ist die Ausbildung zum Certified Tester nach ISTQB-Standard (siehe „Alle Links“). Der Foundation-Level-Lehrplan bietet alle grundlegenden Techniken: von Äquivalenzklassenanalyse über Grenzwertanalyse bis zum zustandsbasierten Testen. Der Stoff deckt alle Teststufen von Unit Test bis Abnahmetest ab, mit denen ein Scrum-Team unweigerlich konfrontiert ist. Die ISTQB-Advanced- und Expert-Level-Lehrpläne vertiefen die Lehrinhalte und eignen sich für Teammitglieder, die ihren Aufgabenschwerpunkt bei Test- und Qualitätssicherungsaufgaben haben. Die deutschsprachigen Lehrpläne werden in Kooperation herausgegeben durch das Austrian Testing Board, das German Testing Board und das Swiss Testing Board. Diese Boards organisieren das Prüfungswesen und überwachen die Ausbildungsangebote im deutschsprachigen Raum.

Für Personen, die schon über eine Certified-Tester-Ausbildung verfügen, sind Trainingsangebote geeignet, die Grundlagen in Scrum vermitteln, verbunden mit der Schulung agiler Testtechniken (etwa „Testen in Scrum-Projekten“ oder „Certified Agile Tester“. Trainings, die auf agile Testtechniken fokussieren und beispielsweise nur auf den Aspekt „exploratives Testen“ eingehen, können eine Ergänzung zu Grundlagentrainings nach ISTQB-Foundation-Level sein, aber sie nicht ersetzen. Wie oben erwähnt, erfordert Testen im agilen Projekt die gesamte Bandbreite an Testmethoden. Explorative Techniken sind nur ein Baustein innerhalb dieses Instrumentariums.

Die Umstellung auf Scrum bringt Veränderungen im Arbeitsstil mit sich. Aus dem Blickwinkel von Qualitätsmanagement und -sicherung sind hier vor allem zwei Veränderungen zu nennen: Das Qualitätsmanagement wandelt sich von einem top-down zu einem bottom-up getriebenen Prozess. Qualitätsmanagement wird zum Dienstleister der agilen Teams (siehe Artikel „Alles geregelt auf S.).

Die operative Qualitätssicherung verliert ihre Unabhängigkeit. Nicht mehr eine externe Testgruppe kümmert sich darum, sondern das Ganze passiert im Team. Damit verbunden sind veränderte Wertvorstellungen von Testern und Qualitätssicherungs-Experten, die in agilen Teams mitarbeiten:

- Ein konstruktives Verhältnis zwischen allen Teammitgliedern zählt mehr als Testprozesse und Testwerkzeuge.
- Getestete Software ist wichtiger als umfassende Testdokumentation.
- Die kontinuierliche Zusammenarbeit mit dem Kunden ist wichtiger als formale Abnahmetests am Projektende.
- Das Reagieren auf Veränderung hat höhere Priorität als das Befolgen eines Testplans.

Gefragt sind Zusammenarbeit, Pairing, Kommunikation statt Dokumentation, Selbstorganisation, Eigeninitiative und aktive Informationsbeschaffung.

Für Test- und Qualitätssicherung-Experten bedeutet dies, dass ihr methodisches Rüstzeug wichtiger und wertvoller ist als jemals zuvor. Ihr Arbeitsstil wird sich jedoch wandeln. (jd)

Quelle

Der Artikel ist ein überarbeiteter und gekürzter Auszug aus diesem Buch:

Tilo Linz

Testen in Scrum-Projekten, Leitfaden für Softwarequalität in der agilen Welt

dpunkt.Verlag 2013

Alle Links: www.ix.de/ix1516