»Agiler Integrationstest mit Mehrwert«

Eine halb agile Insel im Ozean des Wasserfallkonzerns: Für viele Fundamentalisten der Agilität ein Unding, ist sie für die meisten Teams gelebter Projektalltag. Storys als vollintegriertes Inkrement im selben Sprint in den Status „DONE" zu bringen, wird durch verschiedene Dienstleister, Systeme und Prozesse für viele Teams im Enterprise-Umfeld nahezu unmöglich. Ein spezieller Integrationstest kann hier viele Probleme der skalierten agilen Delivery abfangen und erheblichen Mehrwert schaffen.

Beim Begriff *Integrationstest* denkt man unwillkürlich an die rechte untere Ecke des V-Modells, Testfallzahlen und Bürokratie. Wie kann das etwas mit agiler Methodik zu tun haben?

Für dieses Relikt ist doch seit Scrum & Co. und CI/CD kein Platz mehr in unseren agilen Teams. Allenfalls als eine Aktivität von vielen, die während eines Sprints mit einer Story passieren – aber doch nicht als dedizierte Phase mit Team! Oder doch?

Der agile Idealfall

Ein Team konzipiert, entwickelt, integriert, testet und demonstriert ein Inkrement innerhalb eines Sprints und schiebt am Ende ein Sticky Note von „IN PROGRESS" auf „DONE".

Natürlich findet auch hier ein Integrationstest statt. Niemand nennt es aber so, denn es gibt keine Rolle namens *Integrationstester*, keine speziellen *Integrationstestfälle* und keine Spalte *Integrationstest* auf dem Scrumboard (gehört zu „IN PROGRESS").

Die Realität – „agile delivery"

Was im Mikrokosmos von Produkt/Feature-Innovation oder Start-up-Inselleben funktioniert, scheitert spätestens im Konzernumfeld kläglich. Hier fehlen elementare Voraussetzungen, damit ein agiles Team erfolgreich ist. Dies ist keine neue Erkenntnis und gerade bei Fixed-Time-Fixed-Price-Dienstleistern oder Werklieferverträgen ist daher das Modell der *agile delivery* weit verbreitet.

Klingt modern und fancy. Die Idee dahinter ist, dass der Auftraggeber problemlos in seinem Konzernwasserfall und eingefahrenen Prozessen bleiben kann, bis alle Requirements (Lasten- und Pflichtenheft lassen grüßen) fertig sind. Dann wird ein Dienstleister mit der Implementierung beauftragt, der agil-inkrementell das Produkt herstellt. Dank agiler Methodik und Befreiung vom Konzernballast geht es schneller. Das fertige Produkt wird in die Konzernlandschaft implantiert. Soweit die Theorie.

In der Praxis ist das Team nun auf einer Insel des agilen Optimismus gestrandet, umgeben vom Ozean der Widrigkeiten: Feste Releasetermine, autoritäre Prozesse, Schnittstellenpartner mit anderen Releasezyklen, fehlende Product Owner, unterschiedliche Dienstleister und Zuständigkeiten. Eine Story in nur einem Sprint abzuliefern, ist hier fast unmöglich, der Sprint wird zum Hindernisparcours.

Die Herausforderungen

Der Status „DONE" einer Story bedeutet, dass nichts mehr daran zu tun ist: Alles – inklusive *jeglicher* Tests – ist erledigt, das Inkrement produktionsbereit. In der *agile delivery* lässt sich das nicht so schnell feststellen: Ob eine Story wirklich DONE ist, kann erst final beantwortet werden, wenn der Nachweis unter Einbeziehung aller beteiligten Systeme, Komponenten, Daten und Konfigurationen erbracht wurde, die aber oft fehlen.

Lisa Crispin und Janet Gregory haben in *Agile Testing* ([Cri14], Seiten 229, 459) beschrieben, was das für die Integration bedeutet. Frei übersetzt:

- „Wenn Ihr System mit anderen zusammenarbeiten muss, kann es sein, dass Sie diese nicht alle in Ihren Testumgebungen darstellen können. [...] Dies ist eine der möglichen Situationen, in denen es unvermeidbar ist, dass Sie mit dem Test erst starten können, wenn die Entwicklung schon vollständig abgeschlossen ist."
- „Simulatoren, Mocks und andere Hilfsmittel für das Testen während der Entwicklung können manche Risiken verringern,

aber je früher Sie Ihr Produkt in Verbindung mit den externen Anwendungen testen können, umso geringer wird das Risiko."

- „Ihr Team mag agil sein, aber andere Produktteams in Ihrer Organisation oder Drittanbieter, mit denen Ihr Team zusammenarbeitet, sind es vielleicht nicht."
- „Beginnen Sie frühzeitig damit, sich mit anderen Produktteams oder externen Partnern zu koordinieren, deren Produkte mit Ihrem integriert werden sollen."

Zusammengefasst bedeutet das *Risiko* ...

- ... aufgrund fehlender End-to-End-Integration während der Entwicklung.
- ... durch das Auffinden von Fehlern nach Abschluss des Sprint/Release.
- ... durch Reibungsverluste an den Übergängen zwischen agiler und klassischer Welt.
- ... durch Änderungen innerhalb der Partnersysteme oder deren Schnittstellen.

Wie geht man damit um?

Ein praktisches Beispiel

In einem aktuellen Projekt sah die *agile delivery* so aus: Mehrere Scrum-Teams mit unterschiedlicher Erfahrung in agilen Prozessen, unzureichendes Ownership durch den Kunden bei Anforderungen und Lieferantenmanagement sowie überlastete kundenseitige IT.

Dass unsere Teams dennoch „abliefern" konnten, lag zu einem Großteil an unseren Integrationstestern, die als Mitglieder ihrer jeweiligen Scrum-Teams an den Sprints beteiligt waren und gemeinsam als Service-Unit teamübergreifend agierten:

Im Scrum-Team

Die Testaktivitäten innerhalb des Sprints wurden so weit getrieben, wie es unter Einsatz von Mocks/Simulatoren sinnvoll möglich war. Alles Weitere (wir nannten es „Testschuld") wurde in die Service-Unit geschoben.

In der Service-Unit

Hier bestand die Hauptaufgabe darin, die team- und sprintübergreifenden Testschulden und deren Ursachen zu verfolgen: fehlende, fehlerhafte oder unvollständige Komponenten, nicht verfügbare Schnittstellen oder Partnersysteme. Wenn sich aus den Sprints aller Teams die Testbarkeit einer Schuld ergab, wurde diese getestet und die Erkenntnisse daraus wurden zurück in das Entwicklungsteam oder an den zuliefernden Partner gegeben (IT-Abteilung des Kunden, Drittanbieter).

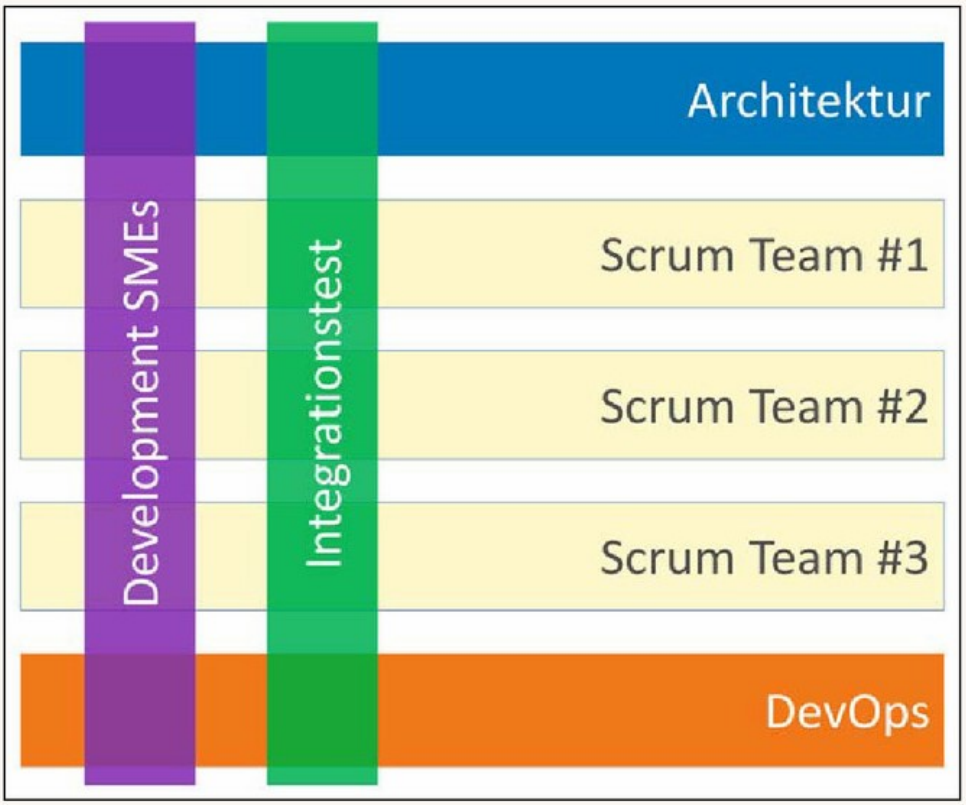

Abb. 1: Schnittstellen zwischen den Teams

Organisation/Prozesse

Unsere Integrationstester als eigenes Team in einem Raum: Das war zu Beginn eine logistische Notlösung, die wir zugunsten integrierter Scrum-Teams abschaffen wollten, entpuppte sich dann aber als Glücksgriff:

Da sie bei sämtlichen Scrum-Events und häufig während eines Sprints bei ihrem jeweiligen Scrum-Team saßen, konnten sie Erkenntnisse zentral sammeln und wieder in die Teams verteilen. Der Integrationstest entwickelte ein team- und scopeübergreifendes Verständnis, quasi das „big picture".

Der tägliche Umgang mit mehreren Umgebungen, den builds, Partnersystemen und deren Schnittstellen ergab eine enge Verbindung zu DevOps und den Drittparteien. Der direkte Zugang zu Informationen, Problemen und Lösungen in den Teams führte dazu, dass die Tester auch exzellente Ansprechpartner für die Entwicklungs-SMEs waren (SME: Subject Matter Expert, hier die „lead"-Entwickler), die ihrerseits über die Scrum-Teams verteilt waren **(siehe Abbildung 1)**.

Um das Potenzial ausschöpfen zu können, wurden organisatorische Maßnahmen getroffen und Prozesse definiert oder angepasst:

Es war klar, dass wir Fehler finden würden, nachdem die Entwicklung des entsprechenden Codes längst abgeschlossen wäre. Also machten wir aus der Not eine Tugend und passten die Workflows so an, dass die Testschulden fester Bestandteil unserer Backlogs wurden: Statt auf frühe Fehlerfindung zu hoffen, planten wir späte Fehler fest ein, da sie sich ohnehin nicht vermeiden ließen.

Wir verzichteten darauf, den Integrationstest als Bestandteil der Definition of Done zu schreiben. Sobald Unittests und Code-Review erledigt waren, galt „DONE". War etwas davon direkt testbar, feature oder bugfix, wurde es sofort so tief wie möglich getestet. Alle fehlenden Testanteile wurden auf den Parkplatz der Service-Unit geschoben.

Natürlich entstanden daraus auch Nachteile. Diese wurden in Kauf genommen, denn die Vorteile waren wichtiger: Die Entwickler hat-

Referenzen

- [Coh09] M. Cohn, The Forgotten Layer of the Test Automation Pyramid, 2009, siehe: **https://www.mountaingoatsoftware.com/blog/the-forgotten-layer-of-the-test-automation-pyramid**
- [Cri14] L. Crispin, J. Gregory, Agile Testing – A practical guide for testers and agile teams, Addison-Wesley, 2014

ten mehr Bandbreite für Backlog Grooming oder Bugfixing und die Tester hatten mehr Bandbreite, um sich um Themen zu kümmern, die mehr als nur ihr jeweiliges Team betrafen: den Mehrwert.

Der Mehrwert

Automatisierung

Dadurch, dass die Integrationstester direkt mit den Schnittstellen und Partnersystemen in den jeweiligen Umgebungen zu tun hatten, lag es nahe, dies zu instrumentalisieren. Automatisierte Schnittstellentests verifizierten die Funktionsfähigkeit der Partnersysteme in den integrierten Umgebungen.

Dies war für alle Beteiligten ein enormer Gewinn: Fehler wurden aufgedeckt, bevor auch nur der erste Code unserer Teams involviert war. Die ansonsten komplexe Fehleranalyse (und Ressourcenbindung unserer Entwickler) entfiel größtenteils.

Qualitätsgesicherte Testware

Während der ersten Sprints existiert noch keine *qualitätsgesicherte* „Testware" (user, configs, content, …), die für die Entwickler von elementarer Wichtigkeit ist. Ihr Fehlen führt dazu, dass notgedrungen rasch unterschiedliche Packages ad hoc kreierter Daten kursieren. Extrapoliert auf mehrere Scrum-Teams führte das zu einem Datenwildwuchs, in dem derselbe Code funktionierte oder auch nicht funktionierte.

Hier konnten unsere Integrationstester aus dem Vollen schöpfen: Sie saßen bei Problemen im jeweiligen Team direkt an der Quelle. Andererseits hatten sie als Service-Unit den gesamten Überblick. Der Integrationstest hatte den „Rundumblick", auf bestimmten Umgebungen, die jeweils gültige und korrekte Kombination von Software, Konfiguration und Content zu bestimmen.

Das funktionierte so gut, dass gemeinsam mit dem Kunden beschlossen wurde, die Rolle des Release-Managements in unserer Integrationstest-Service-Unit zu verankern.

Im Zusammenhang mit Automatisierung sei auf den Beitrag von Mike Cohn „Die vergessene Schicht der Testautomatisonspyramide" [Coh09] verwiesen: Hier liegt erhebliches Potenzial, das leider im Kreuzfeuer agiler Methodik und dem Silodenken einer umgebenden Plattformwelt oft brach liegt.

Der Integrationstest stellte fortan den Entwicklungsteams abgestimmte Pakete zur Verfügung – seien es Konfigurationen, Workflows, Testuser oder JSON/XML-snippets.

Learnings

Der Mehrwert entstand durch zusätzlichen Aufwand. Also Aufwand, der *zusätzlich zur Testaktivität im Sprint* anfällt. Dieser Aufwand verteilte sich zu 50 Prozent auf das In-sprint-Testen (inklusive manueller Regressionstestanteil) und 50 Prozent auf den Anteil der Service-Unit.

Bei uns hat dieses Modell funktioniert, weil es durch alle Beteiligten getragen wurde, wir die Prozesse entsprechend gestalten konnten, und weil wir bei Bedarf auf einen erfahrenen Testautomatisierer zurückgreifen konnten, der primär im kundenseitigen Abnahmetest tätig war. Und zu guter Letzt, weil wir mit den richtigen Leute arbeiteten. Denn es braucht einen guten Mix aus technischen Skills, persönlichen, diplomatischen und planerischen Fähigkeiten, etwas Erfahrung und Fingerspitzengefühl und eine dicke Haut.

Thomas Klein

thomas.klein@redbots.de
ist Senior Consultant bei der redbots. Als Testmanager und Scrum Master betreut er die Projekte der redbots in den Bereichen Softwarequalitätssicherung und agiler Methodik.

Carsten Negrini

carsten.negrini@redbots.de
ist Geschäftsführer der redbots und arbeitet daran, die Effizienz von IT-lastigen Organisationen zu optimieren.