Janet Albrecht-Zölch

Testdaten und Testdaten-management

Vorgehen, Methoden und Praxis

dpunkt.verlag

Inhalt

4 Probleme mit Testdaten und Risiken

Dieses Kapitel bietet Ihnen einen Überblick über wesentliche Probleme in Bezug auf Testdaten und das Testdatenmanagement, die alle als Argumente für ein strukturiertes Testdatenmanagement dienen.

Nach der ausführlichen Beschreibung der im Zusammenhang mit Testdaten möglichen Probleme, betrachtet dieses Kapitel eher grundlegend die Bedeutung mangelhafter Testdaten als Risiko für das Test- bzw. Softwareentwicklungsprojekt sowie als Risiko für die Qualität des späteren Softwareproduktes.

Erfahrene Testdatenmanager und Testprojektmanager, die auch Testdatenmanagement betreiben, werden einiges aus der Praxis wiedererkennen.

4.1 Häufige Probleme mit Testdaten

Dieser Abschnitt beantwortet die Fragen, wo die häufigsten Probleme mit Testdaten liegen, wozu wir bessere Testdaten und ein Testdatenmanagement brauchen.

Einige Herausforderungen liegen in den Testdaten selbst (siehe Abschnitt 4.1.2), alle anderen lassen sich auf ein ungeeignetes Testdatenmanagement zurückführen. Die gute Nachricht: Ein Großteil der Probleme lässt sich durch strukturiertes Testdatenmanagement lösen oder zumindest handhaben.

4.1.1 Probleme mit Testdaten, die auf den Faktor Mensch zurückzuführen sind

Einige wesentliche Probleme in Bezug auf Testdaten sind auf den Faktor Mensch zurückzuführen.

Mangel an Aufmerksamkeit für Testdaten

Paradoxerweise schenken wir grundlegenden Dingen nicht immer die angemessene Beachtung. Insofern überrascht es nicht, dass Testdaten oft nicht die ihnen gebührende Aufmerksamkeit genießen. Dies schlägt sich auch auf das Testdatenmanagement nieder.

Über 10 % der Fehler, die in der Produktion gefunden wurden, treten nach [Muru13] wegen der Testdaten auf. Fehler, die leicht in einer der Testphasen hätten entdeckt werden können.

Die Gründe, warum es den Testdaten an Aufmerksamkeit mangelt, sind vielfältig. Ich sehe vor allem folgende Ursachen:

- Die Vorstellung, Testdaten gehen mich nichts an, denn Testdaten brauchen ja nur die Tester. (Wir wissen, dass diese Ansicht falsch ist, denn zahlreiche Projektbeteiligte lassen sich »mal schnell eine Benutzerkennung« geben, mit der sie »mal was ausprobieren« wollen. Und gelegentlich fallen dabei Ungereimtheiten oder gar Fehler auf.)
- Generell wird grundlegenden Arbeiten zu wenig Anerkennung gezollt. Dinge, die erledigt werden müssen, damit andere auch arbeiten können, sind für Außenstehende oft nicht sichtbar – und damit leider nicht geeignet, Lorbeeren zu ernten.
- Das Beschäftigen mit Testdaten artet schnell in Fleißarbeit aus. Damit lassen sich zwar Fleißbienchen sammeln, aber als der große Held wird man wohl eher nicht gesehen. (Es sei denn, man ist derjenige, der im Falle einer Eskalation durch »Hervorzaubern« von Testdaten die Kartoffeln aus dem Feuer holt.)
- Mangelndes Interesse am Thema allgemein. Um sinnvolle Testdaten gewinnen zu können, benötigt man einerseits einen guten fachlichen Überblick, d.h. Grundwissen über die Funktionalitäten des Systems, und andererseits Detailwissen. Beides ist nicht jedermanns Sache.
- Das Testdatenthema ist komplex und verursacht viel Arbeit, die von manchem sogar als unproduktiv betrachtet wird.

Allerdings ging es der Testdisziplin zu Beginn ihrer Zeit ähnlich. Die zunehmende Professionalisierung der Tester und ihrer Werkzeuge verschafften dem Testen inzwischen eine gute Akzeptanz. Ich gehe davon aus, dass wir Ähnliches bzgl. der Testdaten erleben werden. Zumal mit dem Zunehmen der in Softwaresystemen zu handhabenden Datenmengen ein wachsender Bedarf besteht.

Testdaten mit geringer Priorität behandelt

Dem Thema Testdaten und Testdatenmanagement wird nicht in allen Projekten die Priorität zugestanden, die erforderlich ist, um alle notwendigen Vorbereitungen zu geeigneter Zeit treffen zu können.

Eine Ursache ist sicher auch darin zu sehen, dass man als Tester auf Unverständnis stößt, wenn man bereits zu Beginn eines Projekts Testthemen anspricht. Manchmal drängt sich sogar der Eindruck auf, das Testen und die Tester werden als lästig empfunden. Schließlich ist ja noch nichts programmiert, also kann noch nichts getestet werden. Kommt Ihnen das bekannt vor?

Spätestens wenn die Situation eskaliert, weil das Projekt weit fortgeschritten ist und die Zeit für Testdurchführungen schwindet, erhalten Test und Testdaten Aufmerksamkeit und Ressourcen.

Unklares und/oder uneinheitliches Verständnis des Begriffs Testdaten

Soll gemeinsame Arbeit von Erfolg gekrönt sein, stellt ein gemeinsames Verständnis wesentlicher Begriffe eine notwendige Voraussetzung dar. Dies gilt auch für den Themenbereich Testdaten und Testdatenmanagement.

Oft habe ich beobachtet, dass Kollegen den Begriff Testdaten ausschließlich mit den Daten assoziieren, die unmittelbar zur Durchführung von Testfällen benötigt werden. Also Eingabedaten, Basisdaten u.Ä. Sie verstehen die Umgebungsdaten als zur Konfiguration zugehörig und daher nicht als Testdaten. Wie in Kapitel 2 dargestellt, zählen wir die Umgebungsdaten jedoch zu den Testdaten.

Existiert kein gemeinsamer Begriff, können ärgerliche und zeitraubende Missverständnisse entstehen, die dazu führen, dass Testdaten fehlen oder aber zu viel erstellt werden. Fehlende Testdaten verursachen Verzögerungen in der Testdurchführung und Weiterleitung der Testergebnisse, »zu viel«, also doppelt erstellte Testdaten bedeuten unnützen Zeitaufwand, verbrauchen unnötig Speicherkapazität und erschöpfen z.B. vorgesehene Nummernkreise zu früh (was besonders ärgerlich ist, wenn externe Stellen kontaktiert werden müssen, um weitere Nummernkreise zur Verfügung zu stellen).

Unklarheiten oder Missverständnisse bzgl. des Bedarfs an Testdaten

Solange kein einheitliches Verständnis über den Begriff der Testdaten herrscht, ist nicht sichergestellt, dass der Bedarf an Testdaten klar erkannt wird.

Wer braucht wann wofür Testdaten:

- Der Testumgebungsadministrator braucht Testdaten zum Aufsetzen und Warten der Testumgebungen. Diese Daten (Umgebungsdaten und Basisdaten; zum Begriff siehe Abschnitt 2.1) müssen bereits vor dem Einrichten der konkreten Testumgebung zur Verfügung stehen.
- Der/die Tester benötigen für die Durchführung der Testfälle aller geplanten Teststufen und Testarten Testdaten (Umgebungsdaten, Basisdaten; Eingabe- und Ausgabedaten; zum Begriff siehe Abschnitt 2.1).
- Der Softwareentwickler, der seinen Code testen möchte, sollte ebenfalls z.B. auf geeignete Testdaten (Eingabedaten) zugreifen bzw. diese ableiten (Ausgabedaten) können.
- Derjenige, der das System in der Produktionsumgebung installiert, benötigt ebenfalls eine Möglichkeit, zu prüfen, ob die Installation erfolgreich war und das System wie gewünscht läuft.

Mangelnde Akzeptanz des Testdatenmanagements

Wozu brauchen wir Testdatenmanagement? Wir ziehen einfach einen Datenbestand aus der Produktion ab oder die Tester sollen sich selbst um ihre Daten kümmern. Kommt Ihnen das bekannt vor?

[Ende12] führt an, dass sich im Zuge des neuen Testdatenmanagements für die am Projekt beteiligten Entwickler, Tester, Fachleute usw. die bisher gewohnte Datenbasis ändert. Im Extremfall kann das bedeuten, dass der gewohnte Echtdatenbestand nun durch völlig unbekannte synthetische Daten ersetzt wird. Dies erfordert von den Beteiligten eine gewisse Umorientierung und kann dazu führen, dass die Veränderungen auf geringe Akzeptanz stoßen.

Veränderungen in etablierten Abläufen können Unruhe unter den betroffenen Beschäftigten hervorrufen. Denn mit der Einführung eines neuen Werkzeugs gewinnt man nicht nur, sondern es verliert z.B. der Werkzeugexperte des bisher genutzten Werkzeugs seinen Status. In der Regel bedeuten Neuerungen zusätzlichen Aufwand für die Betroffenen, die sich nun in ein neues Werkzeug einarbeiten müssen.

Testausführung auf der falschen oder einer unsauber konfigurierten Testumgebung

Ist Ihnen das auch schon mal passiert? Sie führten Ihre Testfälle auf der falschen Testumgebung aus? Oder Sie beobachteten merkwürdige Reaktionen Ihres

Testobjekts?

Durch die Ausführung von Testfällen auf der falschen Testumgebung (z.B. andere Version des Testobjekts) oder auf einer schlicht unsauber konfigurierten Testumgebung können die Testergebnisse ungültig sein und die Testfälle wiederholt werden müssen.

Bleibt der Fehler unbemerkt und der Test gilt fälschlich als erfolgt, obwohl das eigentlich gemeinte Testobjekt ungetestet ist und ungetestet ausgeliefert wird – treten Fehler dann erst in der Echtumgebung zutage.

Welches Risiko mit der Testumgebung verbunden ist, betont Chace [Chac11]: Umgebungsdaten gehören zu den Testdaten.

Testmanagement und Testdatenmanagement tragen dazu bei, die richtige Testumgebung zu finden.

In den Themenbereich Konfiguration & Testdaten fällt auch das *Auf-meinem-Rechner-läuft's-Problem*.

Das Auf-meinem-Rechner-läuft's-Problem & Testdaten

Folgende Situation: Sie führen Testfälle durch. Die Testfälle schlagen fehl und der von ihnen dazu konsultierte Entwickler antwortet: »Auf meinem Rechner lief es.«

Dieses *Works-on-my-Machine-Phänomen* und seine Ursachen beschreibt Dave Nicolette in seinem Blogartikel vom März 2017 [Nico17].

Übriggebliebene Konfigurationen

Als ersten Fallstrick führt er das Problem übriggebliebener Konfigurationen an (»leftover configuration«). Diese verursachen auf dem Entwicklerrechner ein – im Vergleich zur Testumgebung oder zu anderen Umgebungen – abweichendes Verhalten des Testobjekts. »Leftover configuration from previous work enables the code to work on the development environment (and maybe the test environment, too) while it fails on other environments« [Nico17].

Konfigurationsdaten gehören zur Testdatenkategorie *Umgebungsdaten*. Das Testdatenmanagement muss also Entwicklertests in der Entwicklungsumgebung berücksichtigen.

Abweichende Konfigurationen

Darüber hinaus soll es sicherstellen, dass sich die Konfigurationen in der Entwicklungs- und Testumgebung sowie der Produktionsumgebung gleichen oder dass gewünschte Änderungen dokumentiert und in der Produktionsumgebung nachgezogen

werden. Denn Fallstrick 2 lautet: »Development/test configuration differs from production.« Die Ursachen sieht er darin, dass Entwickler mit dem Hinzufügen neuer Bibliotheken auch Konfigurationseinstellungen manuell anpassen oder konkurrierende Konfigurationen auftreten: »The environment grows from project to project, as more libraries are added and more configuration options are set. Sometimes the configurations conflict with one another, and teams/individuals often make manual configuration adjustments depending on which project is active at the moment.«

Dieses Phänomen lässt sich übrigens auch auf den Rechnern verschiedener Tester beobachten:

Praxisbericht: Fehlgeschlagene Testfälle aufgrund veralteter Konfiguration des Testwerkzeugs

Als Testerin in einem Team von Testern war ich mit der Durchführung automatisierter Testfälle befasst. Nach dem Deployment einer neuen Version des durch uns zu testenden Systems ließen wir Tester die geplanten Testfälle laufen. Einige der Regressionstestfälle auf meiner Liste schlugen fehl. Das konnte nicht sein, denn im letzten Durchlauf endeten sie erfolgreich und an der getesteten Funktionalität hatte sich nichts geändert. Was war passiert?

Wir verwalteten die Testfälle in einem Testfallmanagement-Werkzeug, das die Testfälle in der Testumgebung auch ausführte. Zwischen den beiden Testläufen wurde eine Konfiguration im Testfallmanagement-Werkzeug verändert. Das hatte sich im Testteam nicht herumgesprochen. Da die Bausteine, auf denen die Testfälle basierten, bereits auf die neue Konfiguration angepasst waren, schlugen die Testfälle auf den Rechnern aller Tester fehl, die diese Konfiguration in ihrem Testfallmanagementsystem noch nicht nachgezogen hatten.

Was lernen wir daraus? Konfigurationen für Testwerkzeuge sollten zentral verwaltet und umgesetzt werden und – wenn möglich – der Zugriff auf diese Parameter nicht für jeden Tester erlaubt sein.

Dave Nicolette [Nico17, Pitfall 2] empfiehlt zur Lösung des Problems der verschiedenen Konfigurationen z.B. den Einsatz virtueller Maschinen.

Mangelnde Kommunikation

Wie wichtig es ist, geplante oder bereits vorgenommene Änderungen an Testdaten zu kommunizieren, erfährt man spätestens dann, wenn man bisher erfolgreich

laufende Regressionstestfälle zu unveränderten Funktionalitäten plötzlich fehlschlagen sieht. Diese Situation erlebte ich durchaus:

Praxisbericht: Fehlschlagende Testfälle & Analyseaufwand wegen nicht kommunizierter Änderungen

Ein Kollege änderte »mal schnell« eine Umgebungsvariable, weil er etwas testen wollte. Leider vergaß er zwei wichtige Dinge: erstens den Testteamkollegen diese Modifikation mitzuteilen und zweitens die Variable wieder zurückzusetzen.

Diese zu Testzwecken editierte Variable führte leider dazu, dass die Regressionstestfälle einer anderen Testkollegin fehlschlugen. Deren Testfälle basierten auf einer korrekten Umgebungsvariablen und konnten mit der editierten nicht erfolgreich laufen.

Zum Glück fand sie recht schnell heraus, worin die Ursache dieses Testergebnisses lag.

Im schlechtesten Falle ziehen unbekannte Änderungen an Testdaten aufwendige Analysen nach sich, da fehlgeschlagene Testfälle ja auf Fehler in der Anwendung hinweisen. Womöglich wird ein Testfall fälschlich »korrigiert« – d.h. an die »neue« Funktionalität der zu testenden Anwendung angepasst. Auch das habe ich schon erlebt.

Änderungen an Testdaten – auch vorübergehende – müssen kommuniziert werden

Lessons Learned

Wenn Tester auf einer gemeinsamen Testumgebung arbeiten, dann müssen sie Änderungen an bestimmten Daten mitteilen. Eventuell benötigen sie die Testumgebung sogar für eine bestimmte Zeit exklusiv – um nämlich für ihren speziellen Testfall zentrale Testdaten ändern zu können, ohne die Testergebnisse der Kollegen zu verfälschen.

Demzufolge sollten sich in einem Testdatenkonzept Hinweise dazu finden, wer unter welchen Umständen welche Testdaten bearbeiten oder löschen darf und wer darüber – am besten vorab – zu befragen und zu informieren ist.

Gemeldeter Softwarefehler nicht nachvollziehbar, da zu nutzende Testdatenkonstellation nicht ausreichend beschrieben

Testdaten können auch in Fehlerberichten eine tragende Rolle spielen. So erhöht es die Nachvollziehbarkeit von Fehlerbeschreibungen, wenn darin die

verwendeten Testdaten angegeben sind. Dies gilt insbesondere für diejenigen Fehlerwirkungen, die nur unter Verwendung bestimmter Testdatenkonstellationen auftreten.

»Bei mir tritt der Fehler nicht auf. Wie hast du das getestet?« oder »Mit welchen Daten hast du das getestet?« Kommen Ihnen diese Aussagen eines Entwicklers bekannt vor?

Wenn ja, wissen Sie aus eigener Erfahrung, dass auch die im Test verwendeten Testdaten im Fehlerbericht erwähnt werden müssen. Da sich der komplette Datenbestand nicht beifügen lässt – was ja auch nicht sinnvoll wäre –, bleibt es eine Herausforderung, genau die richtigen Angaben zu den Testdaten zu notieren, die für das Nachstellen des beobachteten Fehlverhaltens der Anwendung relevant sind.

Manchmal findet man erst dann heraus, auf welche Daten es ankommt, wenn der Entwickler dem Tester zeigt, wie er den Fehler nachzustellen versucht – und die Testschritte auf *anderen* Testdaten ausführt als der Tester im ursprünglichen Testfall. So geschehen im nachfolgend geschilderten Erlebnis.

Praxisbericht: Fehler tritt nur auf, wenn Attribut X abgefragt – im Fehlerbericht notieren

Einmal stellte ich beim Testen einer Anwendung fest, dass der am System angemeldete Kunde A auf der Seite mit seinen Adressdaten fälschlich auch die Adressdaten eines Kunden B angezeigt bekam. In der Produktion hätte dies eine Datenschutzverletzung dargestellt.

Ich besprach die Fehlerwirkung mit einem Testkollegen. Gemeinsam grenzten wir die mögliche Fehlerursache auf ein bestimmtes Attribut der beiden Kunden ein. Im zugehörigen Fehlerbericht notierte ich die notwendigen Testdaten, mit denen sich die Fehlerwirkung reproduzieren lassen sollte. Wenig später stellte ich fest, dass der Entwickler den Fehlerbericht zurückwies – er konnte die Fehlerwirkung nicht nachstellen.

Da es sich um einen gravierenden Fehler handelte, suchte ich den Entwickler sofort persönlich auf und bat ihn, den Fehler in meiner Gegenwart noch einmal nachzustellen. Schließlich kamen wir an den Punkt, an dem der Entwickler für eine Abfrage das Attribut X verwendete – eines mit vermeintlich einzigartigem Wert. Doch die Abfrage lieferte entgegen den Erwartungen nicht nur einen, sondern zwei Treffer: nämlich Kunde A und Kunde B! Daher wurden auf der Seite des Kunden A auch die Daten von Kunde B angezeigt.

Im Fehlerbericht hatte ich nicht deutlich genug betont, dass es darauf ankommt, dass zwei Kunden für ein bestimmtes Attribut in der Historie denselben Wert aufweisen müssen, um die Fehlerwirkung nachvollziehen zu können.

Für die Praxis zog ich folgenden Schluss:

Gelegentlich tritt die Situation auf, dass ein Fehler mit genau dieser verwendeten Kundenkennung nicht nachgestellt werden kann, weil der Account für weitere Testfälle verwendet und dessen Historie damit verändert wurde. In solchen Fällen empfiehlt es sich, auf das weitere Benutzen dieser Testdaten nach Auftreten des zu meldenden Fehlers zu verzichten und stattdessen andere Daten zu verwenden. Manchmal weiß man das aber erst danach …

Verwendete Testdaten oder relevante Datenkonstellation im Fehlerbericht angeben Lessons Learned

Grundsätzlich empfiehlt es sich, im Fehlerbericht die für den Testfall verwendeten Testdaten anzugeben. Sofern der Fehler bereits dahingehend eingegrenzt werden konnte, sollte der Verfasser des Fehlerberichts deutlich machen, auf welche Testdaten oder Datenkonstellationen es beim Nachstellen der Fehlerwirkung ankommt. Denn womöglich muss der Entwickler diese in der ihm zur Verfügung stehenden Entwicklungs- und Testumgebung zunächst nachbilden.

So steigt die Wahrscheinlichkeit, dass der Entwickler den Fehler nachvollziehen, ggf. weiter analysieren und vor allem beheben kann.

Testdaten in der Produktionsumgebung

Obwohl Testen in Produktionsumgebungen nicht erlaubt ist, so muss nach der Produktionsinstallation des ehemaligen zu testenden Systems überprüft werden, ob die Installation erfolgreich lief und ob das System nun für den Gebrauch freigegeben werden kann. Auch sind manche Integrationstests nur in der Echtumgebung möglich, da die zu integrierenden Nachbarsysteme in keiner Testumgebung zur Verfügung stehen.

Beim Testen in Echtumgebungen müssen besondere Vorsichtsmaßnahmen ergriffen werden. Denn auf keinen Fall dürfen echte Daten und echte Kunden durch Testaktivitäten beeinträchtigt werden. Demzufolge müssen auch die Daten getrennt gehalten werden (z.B. indem der Test mit einer Benutzerkennung durchgeführt wird, die Zugriff nur auf einen eingeschränkten Bereich der Daten hat, während die Benutzerkennungen des Echtbetriebs genau diesen Bereich

aussparen). Dann treten Irritationen wegen Testdaten in Produktionsumgebungen (wie z.B. James Bachs Tester-Knöllchen; siehe Praxisbericht auf Seite 96) gar nicht erst auf.

4.1.2 Probleme mit Testdaten, die in den Testdaten selbst liegen

Die nachfolgend dargestellten Schwierigkeiten ergeben sich direkt aus den Eigenschaften der Testdaten.

Komplexität der benötigten Testdaten

Eine große Herausforderung besteht in der Komplexität der Testdaten. Um Testdaten erstellen zu können, benötigt man sowohl fachliches als auch technisches Wissen – und zwar für alle Systeme oder Systemteile, in denen Anteile der Datensätze gespeichert bzw. verarbeitet werden. Dies gilt für komplexe Daten umso mehr (siehe dazu auch die Ausführungen von [Held16] und die Erfahrungsberichte).

Die Ursache des Problems ist Testdaten-immanent, daher lässt sich das Problem für bestehende zu testende Systeme nicht lösen.

In den folgenden Fällen benötigt man für das Erstellen von Testdaten detailliertes Wissen über diejenigen Systeme oder Systemteile, die die Daten liefern:

- Bei Last- und Performance-Tests: Es reicht nicht aus, nur Stammdaten und Bewegungsdaten zu haben. Die Daten müssen auch zueinander passen, d.h., die richtigen Stammdaten müssen mit den richtigen Bewegungsdaten (Konto und Kontobewegungen desselben Kontoinhabers) eingesetzt werden. Oft kommen diese Daten aus verschiedenen Systemteilen, sodass ein synthetisches Erzeugen der Daten kein triviales Problem ist.
- Beim Testen von DWH- und BI-Systemen: In diesen Systemen fließen üblicherweise Daten aus verschiedenen Systemen mit unterschiedlichen Datenstrukturen zusammen. Um das System überhaupt testen zu können, benötigt man eine gewisse Menge an Daten aus den verschiedenen Systemen.

Umfang der Testdaten

Eine der Anforderungen an den Testdatenbestand insgesamt lautet, dass dieser weder zu groß noch zu klein bemessen sein soll. Wie umfangreich dürfen die

Testdatenmengen sein, um diese Anforderung zu erfüllen? Wie groß ist der ideale Testdatenbestand?

Diese Frage lässt sich schwer allgemeingültig beantworten. Denn die benötigten Testdaten werden durch die spezifizierten Testfälle bestimmt. Welche und wie viele Testfälle benötigt werden, ergibt sich aus der angestrebten Testabdeckung.

Testdaten und Testfälle wiederum hängen eng zusammen mit der Testart, für die sie entworfen wurden, und mit dem Testziel, dem sie dienen.

Ein Testdatenbestand für funktionale Tests (z.B. Testfälle der Art »Anlegen eines Neukunden«) wird sich wesentlich unterscheiden von einem Testdatenbestand für einen Lasttest, der z.B. misst, wie viele Neukunden sich gleichzeitig registrieren können. In letzterem Falle geht es darum, zu zeigen, dass das System eine bestimmte hohe Anzahl an Aufrufen desselben Service verkraftet. Dabei spielt die Varianz der Testdaten im Gegensatz zum oben genannten funktionalen Test eine untergeordnete Rolle. So kann Liesbeth Müller im Rahmen des Lasttests durchaus unzählige Male ihre Daten ins System schreiben. Für den funktionalen Test sollten in jedem Testfall andere Eingabedaten Verwendung finden.

Größer ist nicht besser: Ein zu großer Testdatenbestand verlangsamt die Testdurchführung

Ein zu großer Bestand an Testdaten beansprucht nicht nur Speicherplatz, sondern kostet auch wertvolle Testzeit. So kann z.B. die Suche nach einem geeigneten Datensatz, der für einen bestimmten Testfall verwendet werden soll, sehr lange dauern, da zahlreiche Datensätze zu durchsuchen sind.

Gleiches gilt für das Prüfen von Testergebnissen: Stellen Sie sich eine Datenbank vor, die ein Vielfaches der benötigten Daten enthält. Eine Abfrage für eine Tabelle läuft dann unter Umständen sekundenbis minutenlang. Wenn mehrere Tabellen involviert sind, steigt die Wartezeit weiter.

Diese Suchzeiten können im automatisierten Testen dann zum Problem werden, wenn ein Testfall mit dem Ergebnis der Suche weiterarbeitet, bevor er vom System die Daten erhalten hat. Dieser Testfall schlägt fehl und führt unter Umständen zum Abbruch des gesamten Testlaufs oder zum Fehlschlagen der folgenden Testfälle in diesem Testlauf. Solche Abbrüche lassen sich ggf. mit dem Einbauen von Wartephasen (»waits«) vermeiden – in jedem Falle aber erhöhen sie die Testlaufzeit.

Chace [Chac11, S. 2] verweist auf ein Testprojekt ihres Arbeitgebers, bei dem die Durchführungsdauer eines Testzyklus von einer Woche auf einen halben Tag reduziert werden konnte – allein dadurch, dass die Testdatenmenge auf eine geeignete Teilmenge gekürzt wurde. (Es ging konkret um die Nachteile von Echtdaten als Testdaten, unter anderem um den ungeeigneten Umfang der Daten.)

Aus Gründen der Nachvollziehbarkeit oder zur Analyse von Fehlern werden u.a. Logdateien und andere Systemdateien in Testumgebungen gespeichert und über einen bestimmten Zeitraum aufgehoben. Je größer der Bestand wird oder je größer einzelne Dateien sind, desto zeitaufwendiger ist es, diese zu durchsuchen.

Begrenzte Lebensdauer von Testdaten: Testdaten altern

Wie wir in Abschnitt 3.1 bereits erfahren haben, weisen nicht alle Testdaten dieselbe Lebensdauer auf. Einige Testdaten verbrauchen sich und müssen fortlaufend neu erstellt werden. Alternativ wird der gesamte Testdatenbestand zurückgesetzt.

Das Testdatenmanagement regelt die Details zu diesem Problem. Eventuell lassen sich z.B. Testdatenpakete derart schnüren, dass die regelmäßig verbrauchten und neu zu erstellenden Testdatensätze in einem eigenen Paket zusammengefasst sind. Sodass diese Daten dann als Aktualisierung des Pakets automatisiert neu erstellt werden können. Steuerungsinformationen dazu liefern die Metadaten.

Testdaten altern

Testdaten altern. Was das bedeutet, illustriert das folgende Beispiel:

Beispiel: Mein Onlineshop – »bisher gekauft«

Sie haben einen Kunden im System angelegt, mit diesem ein paar Einkäufe getätigt und diverse Änderungen an dessen Stammdaten getestet. Nun prüfen Sie mit diesem Kunden seit einigen Monaten nur noch, ob z.B. seine getätigten Einkäufe in der Seite »bisher gekauft« korrekt angezeigt werden. Das System ist so spezifiziert und implementiert, dass es die Einkäufe für einen bestimmten Zeitraum anzeigt.

Seit dem letzten Einkauf verging einige Zeit (die Testdaten alterten). Daraus ergibt sich, dass der letzte Einkauf *vor* dem anzuzeigenden Zeitraum getätigt wurde.

Ein automatisierter Test mit diesem Kunden, der nach dem Öffnen der Seite sofort die angezeigten Einkäufe prüft, schlägt nach einiger Zeit fehl, wenn er bestimmte Einkäufe in der Liste erwartet, der Zeitraum dafür aber abgelaufen ist. Also müssen entweder der Testfall oder die Testdaten angepasst werden.

In diesem Falle erscheint es sinnvoll, für den Test der Funktionalität »Zeige bisher gekaufte Produkte an« verschiedene Testkunden zu verwenden: ein Kunde mit keinem Einkauf, ein Kunde mit genau einem Einkauf, ein Kunde mit zahlreichen und aktuellen Einkäufen.

Darüber hinaus empfiehlt es sich, zumindest den Kunden mit mehreren Einkäufen aktuell zu halten, d.h. diesen immer wieder Einkäufe tätigen zu lassen, sodass beim Öffnen der Seite immer Treffer erzielt werden, ohne dass der Anzeigezeitraum verändert werden müsste.

Testdaten sind umgebungsspezifisch (Pakete)

Stellen wir uns erneut folgende Situation vor: Im Testkonzept ist vorgesehen, verschiedene Teststufen auf jeweils eigenen Testumgebungen durchzuführen. So sollen z.B. Entwicklertest und Systemtest auf der Entwicklungsumgebung stattfinden. Wogegen für den Systemintegrationstest eine Systemintegrationstestumgebung bereitgestellt werden soll. Außerdem soll gelten, je höher die Teststufe, desto produktionsähnlicher die Testumgebung.

Das kann dazu führen, dass sich die Testumgebungen nicht nur bzgl. ihrer Softwareversion, sondern auch hinsichtlich ihres Aufbaus unterscheiden. Während in der Entwicklungsumgebung vielleicht nur ein Applikationsserver und ein Webserver existieren, besteht die Systemintegrationstestumgebung aus einem Applikationsserver und mehreren Webservern, die per Load Balancer bedient werden.

Daraus ergeben sich Unterschiede in den Umgebungsdaten: In den Daten für die Entwicklungsumgebung sind dann weniger und andere Parameter enthalten als in den Daten für die Systemintegrationstestumgebung. Die Testdatenbestände beider Umgebungen sind also nicht gleich, sondern umgebungsspezifisch.

Daher bietet sich an, die Testdaten auch dahingehend zu unterscheiden, ob sie für alle Testumgebungen verwendet werden können oder ob sie – weil umgebungsspezifisch – nur für eine bestimmte Testumgebung passen. Dieses Wissen sollte zum einen im Testdatenkonzept festgehalten sein und zum anderen bei der Organisation der Testdaten berücksichtigt werden. Die Testdaten sollten z.B. in verschiedene Pakete unterteilt werden, sodass unabhängig voneinander auf die Umgebungsdaten und auf Stammdaten oder andere Eingabedaten zugegriffen werden kann.

Eine solche Unterscheidung könnte als positiven Nebeneffekt die Möglichkeit bieten, auch die Produktionsumgebung bei der ersten Auslieferung automatisiert

initial mit Daten zu versorgen. Dazu müsste das Datenpaket die realistischen Daten enthalten; vielleicht wurden diese für eine Schulungsumgebung ohnehin erstellt.

Zeitreisen

Zeitreisen simulieren zu Testzwecken das Altern der Daten. Die Testdaten sollen aussehen, als seien sie auf »natürliche Art« durch den Gebrauch des zu testenden Systems entstanden bzw. genutzt worden.

Dazu führt man z.B. regelmäßig Batchläufe durch, manipuliert gezielt und koordiniert Uhrzeiten und Datum der Server. Zeitreisen stellen insofern eine Herausforderung für das Testdatenmanagement dar, als sie sich als sehr aufwendig erweisen können (vgl. [DaGl16]).

4.1.3 Probleme aufgrund fehlerhafter, ungeeigneter oder vergessener Testdaten

Dieses Unterkapitel umfasst diejenigen Probleme, die sich daraus ergeben, dass die im Test verwendeten Testdaten schlicht ungeeignet waren, den Testzweck zu erfüllen.

Verspätete Bereitstellung der Testdaten

Wenn die Testdaten zu spät bereitgestellt werden, d.h. erst nach dem geplanten Testbeginn, dann verzögert sich naturgemäß die Testdurchführung. Das führt dazu, dass nicht alle geplanten Testfälle oder Testszenarien ausgeführt werden können. Eine Verlängerung des Testzeitraums ist nicht immer möglich.

Praxisbericht: Fehlende Testdaten

Vor einigen Jahren führte ich Systemintegrationstests gegen eine Webanwendung durch. Die Anwendung bezog Inhalte für ihre Internetseiten zum Teil aus einem Content-Management-System (CMS). Ein eigenes Content-Team pflegte die Inhalte in das CMS ein.

Leider fand das oft zu Beginn des Zeitraums statt, in dem wir laut Projektplan bereits Tests durchführen sollten.

Der fehlende Content hielt uns in zweierlei Weise auf: Zum einen standen die für die Tests benötigten Seiten nicht zur Verfügung, d.h., unser Testzeitraum verkürzte sich. Zum anderen waren wir gezwungen, die Reihenfolge der einzupflegenden Inhalte abzustimmen bzw. den Fortschritt des Content-Teams laufend abzufragen, um sofort mit dem Test der

fertiggestellten Seiten und der durch sie verkörperten Funktionalitäten beginnen zu können.

Ausgabedaten oft vergessen

Ganz ehrlich, wenn Sie an Testdaten denken, kommen Ihnen dann auch die entstehenden Ausgabedaten in den Sinn?

Praxisbericht: Testdaten legen Testumgebung lahm

Ich erlebte einmal, dass während einer Testdurchführungsphase eine Testumgebung für mehrere Stunden nicht zur Verfügung stand. Die Ursache dafür war bald gefunden: Die Platten waren voll. Das zu testende System schrieb während seiner Laufzeit nicht nur Daten in die Datenbank, sondern füllte auch diverse Logdateien. Unsere Testaktivitäten hatten dazu geführt, dass wir mit den durch den Test erzeugten Daten den Speicher der Testumgebung so vollgeschrieben hatten, dass wörtlich nichts mehr ging.

Natürlich wurde sofort eine Erweiterung des Speichers in die Wege geleitet. Doch als Zwischenlösung mussten Daten gelöscht werden, die wir eigentlich aufheben wollten. Sonst hätten wir nicht weitertesten können, bis Tage später die Plattenerweiterung eingebaut war.

Dieser Erfahrungsbericht zeigt, dass man sich im Rahmen eines Testdatenkonzepts auch Gedanken über den Platzbedarf der Testdaten machen muss.

Auch für Ausgabedaten muss man Speicherplatz einplanen Lessons Learned

Im Testdatenkonzept muss auch der Platzbedarf der Testdaten betrachtet werden. Dabei sind nicht nur Umgebungs- und Stammdaten, sondern eben auch die Ausgabedaten zu berücksichtigen. Bezüglich der Ausgabedaten (z.B. Logdateien, Datendateien) ist zu überlegen, wie lange diese aufbewahrt werden sollen. Denn wenn Logdateien zahlreich und detailliert erstellt werden, belegen sie über die Dauer der Testperiode sehr viel Speicherplatz. Dem muss beim Planen der Testumgebung Rechnung getragen werden.

Fehlende Testdaten

Testdaten können auf zweierlei Arten fehlen: entweder beim Spezifizieren von Testfällen oder bei deren Durchführung.

Zur ersten Möglichkeit: Uns liegen Anforderungen vor, zu deren Test wir Äquivalenzklassen und Grenzwerte identifizieren. Üblicherweise wählen wir pro

Äquivalenzklasse einen Repräsentanten aus und bilden damit in Kombination mit einem abstrakten Testfall einen konkreten Testfall. Nehmen wir an, uns wäre ein Fehler unterlaufen und eine Äquivalenzklasse entgeht uns. Für diese können wir weder einen Repräsentanten benennen noch einen konkreten Testfall erstellen.

Diese Testdaten fehlen und führen dazu, dass es dazu auch keinen Testfall gibt. Das stellt man im schlechtesten Fall erst fest, wenn in der Produktion genau diese ungetestete Ausprägung der Anforderung zu einem Fehlverhalten der Anwendung führt.

Die zweite Möglichkeit besteht darin, dass Testdaten fehlen, wenn Testfälle nicht wie geplant durchgeführt werden können, weil z.B. benötigte Stammdaten oder Historiendaten, mit denen der Testfall arbeiten würde, nicht in der Testumgebung existieren. Das fällt relativ schnell auf, wenn deshalb Testfälle falsch negativ ausgehen.

Zur zweiten Möglichkeit gehört auch die Situation, dass Konfigurationsdaten fehlen, die Testumgebung also unvollständig eingerichtet ist. Bekommen wir das mit? Ja, wenn Testfälle fehlschlagen oder die Anwendung »merkwürdig« reagiert (z.B. beim explorativen Test).

Das folgende Praxisbeispiel zeigt, welche Auswirkungen fehlende Testdaten (oder fehlende Testfälle) im Produktionsbetrieb entfalten können.

Beispiel: Ausfall der Gepäckanlage am Flughafen

Die Software der Gepäckförderanlage hatte den 29. Februar nicht als Datum erkannt. Dieses dient jedoch als Sortierkriterium. Nachdem es wiederholt zu Ausfällen kam, stellte die Software ihren Dienst am Vormittag ganz ein. Die Koffer wurden am Check-in manuell sortiert, 1.200 Gepäckstücke blieben jedoch liegen. Ein Software-Update ermöglichte gegen Mittag den weiteren Betrieb der Gepäckförderanlage (siehe [Berk16], [Anon16a]).

Dies zeigt, dass Tests allein mit Werten wie erster Tag oder letzter Tag des Jahres nicht ausreichen.

Lessons Learned

Schalttage nicht vergessen!

Alle Felder, die Kalenderdaten verkörpern, sollten im Test auch mit einem Datumswert gefüllt werden, der einen Schalttag darstellt.

Fehlerhafte Testdaten führen zu fehlerhaften und unbrauchbaren Testergebnissen

Fehlerhafte Testdaten können zu fehlerhaften Testergebnissen führen. Diese Ergebnisse sind wertlos und stellen eine Verschwendung von Ressourcen dar.

So kann ein Testfall aufgrund der fehlerhaften Testdaten falsch positiv oder falsch negativ ausfallen.

Im ersten Fall liegt eine Fehlermaskierung vor: Das Fehlverhalten der Anwendung wird aufgrund ungeeigneter Testdaten nicht erkannt. Passiert das in der letzten Teststufe, »rutscht« der Fehler durch und kann im Echtbetrieb zu erheblichen Problemen und Kosten führen.

Im zweiten Fall, falsch negatives Testergebnis, wird fälschlich ein Fehler gemeldet (und weiterbearbeitet), der auf Testen mit falschen Daten zurückzuführen ist. So etwas wirkt sich eher negativ auf das Verhältnis zwischen Entwicklung und Test aus.

Unterm Strich schmälern beide fehlerhaften Testergebnisse das Ansehen des Tests. Das Vertrauen in Testergebnisse geht verloren, das Ansehen der Tester sinkt.

Mangelnde Qualität der Testdaten und mangelnde Datenintegrität

Ein schlechter oder fehlender Testdatenmanagementprozess wirkt sich ebenso negativ auf die Qualität der Testdaten aus wie ein inkonsistentes Vorgehen beim Aktualisieren des Testdatenbestands [Muru13]. Das kann zu schwerwiegenden Problemen bzgl. der Datenqualität und zu Integritätsverletzungen führen. Das gilt gerade im Umfeld komplexer und heterogener Systeme.

Ein Mangel an Qualität der Testdaten kann z.B. in Folgendem bestehen: in Redundanzen in Datenbeständen, schlechter Auffindbarkeit eines bestimmten Datensatzes, der Eignung der benutzten Daten für die durchgeführten Testfälle oder in vernachlässigter Pflege der Testdaten.

Testergebnisse, die auf »schlechten Testdaten« basieren, lassen keine verlässliche Aussage über die Qualität der getesteten Software zu.

Ein weiteres Risiko, das sich aus der Verwendung der Testdaten ergibt, besteht darin, dass die Qualität der Testdaten mit zunehmender Nutzungsdauer sinkt: Der »Datenbestand ist kaputt getestet«. Daten, die sich »verbrauchen« (z.B. neu erstelltes Kundenkonto ohne Transaktionen), müssen erneut angelegt werden. Manipulationen direkt auf der Datenbank (Daten zurücksetzen) können fehlerhaft oder unvollständig ausgeführt worden sein.

Wenn die Ursache ein unreifer Testdatenmanagementprozess ist, lautet die Lösung, den TDM-Prozess zu verbessern.

Unrealistische Testdaten als Fehlerursache von Fehlern in der Produktionsumgebung

Welche Bedeutung das Testen mit realistischen oder realitätsnahen Daten hat, zeigt u.a. folgendes Umfrageergebnis:

Auf Platz 3 der häufigsten Ursachen für in Produktion gefundener Fehler nannten die Teilnehmer einer Studie (siehe [Ferr16] und [Neum16]) das Testen gegen unrealistische Daten, bevor ein System in Produktion geht. 26 % der Teilnehmer sahen die unrealistischen Testdaten als ursächlich. Mit 27 % auf Platz 2 landete übrigens die gegenseitige Abhängigkeit externer Systeme.

Das Debuggen dieser in der Produktionsumgebung vorhandenen Fehler nimmt bei 43 % der Antwortenden zwischen 10 und 25 % ihrer Zeit ein, stolze 18 % der Teilnehmer müssen sogar zwischen 25 und 50 % fürs Debuggen von Fehlern aufwenden. Nur 34 % der Umfragebeteiligten investieren bis zu 10 % darauf.

Wir wissen, je später ein Fehler gefunden wird, desto höher sind die Kosten zum Beheben des Fehlers.

Fachexperten beim Testdesign und Spezifizieren von Testdaten nicht involviert

Wie sehr es sich empfiehlt, beim Testdesign und vor allem beim Spezifizieren von Testdaten die Fachexperten zu beteiligen, zeigt der Erfahrungsbericht von Stephan Grünfelder (siehe Abschnitt 13.4.3).

Wenn Tester, Entwickler und Vertreter der Fachabteilung z.B. im Rahmen eines Scrum-Teams zusammenarbeiten, dann findet ohnehin ein regelmäßiger Austausch und demzufolge auch Reviews verschiedener Artefakte statt.

In anderen Vorgehensweisen oder Teststufen, die außerhalb von Scrum Teams durchgeführt werden, sollten Tester darauf bestehen, Kontakt zur Fachabteilung bzw. zu den Verfassern der Anforderungsspezifikation zu halten. Im Gespräch finden oft hilfreiche Informationen ihren Weg zum Tester, die er sonst nicht bekommen hätte (»Wieso aufschreiben, ist doch selbstverständlich!«, »Wenn ich das alles aufschreibe, werde ich ja nie fertig!«).

Vertreter des Fachbereichs sollten Angaben zu späteren Nutzern des zu testenden Systems machen können. Anhand dieser lassen sich für den Test

(zumindest in höheren Teststufen wie dem Systemtest) Personas definieren und daraus direkt oder indirekt weitere Testdaten ableiten.

Wenn eine intensive Zusammenarbeit nicht möglich ist, sollten zumindest Worst-Case-Szenarien (Welche schweren Fehler können auftreten?) und die Teststrategie mit dem Fachbereich besprochen werden. Ein gemeinsames Überfliegen und Besprechen einzelner Testfälle kann ebenfalls nicht schaden. Auf diese Art und Weise gewann ich gelegentlich interessante Informationen, die ich verwendete, um Testfälle zu eliminieren oder durch sinnvollere zu ersetzen.

4.1.4 Herausforderungen bei Gewinnung, Herstellung und Wartung von Testdaten

Der Umgang mit Testdaten weist einige Herausforderungen auf, die sich auch auf die Gewinnung, die Herstellung, aber auch auf die Wartung von Testdaten beziehen.

Testdaten aktuell halten

In einer Studie im Bereich DevOpsTest [Ferr16] gaben über 20 % der Teilnehmer an, dass die größte Herausforderung im Umgang mit Testdaten für sie darin besteht, die Testdaten aktuell zu halten. Die Notwendigkeit ergibt sich z.B. daraus, dass Testdaten altern oder sich verbrauchen.

Eine Möglichkeit ist das Zurücksetzen des Datenbestands auf einen »sauberen« Stand. Das heißt, nach Beendigung der Testdurchführung wird der zuvor gesicherte Testdatenbestand eingebracht und ersetzt so den »verbrauchten«.

Aufwand aufgrund mangelnder oder ungeeigneter Werkzeugunterstützung

Je komplexer das zu testende System, desto aufwendiger ist auch das Gewinnen und Bereitstellen von Testdaten. Der Aufwand wächst auch mit der Menge der benötigten Daten und der Anzahl der zu versorgenden (Test-)Umgebungen.

Für die Analyse und Anonymisierung von Echtdaten sowie beim Erzeugen von synthetischen Testdaten leisten geeignete Werkzeuge gute Dienste.

Wo immer möglich, sollte man aufwendige Handarbeit vermeiden und eine Automatisierung anstreben – sofern der Aufwand dafür akzeptabel ist. Der Automatisierungsaufwand ist am ehesten gerechtfertigt für häufig wiederholte, manuell sehr zeitintensive Schritte der Testdatenerzeugung und -bereitstellung.

Abwägung zwischen Testdaten direkt in die Datenbank schreiben oder lieber über Schnittstellen des Systems eingeben

Dieses Problem ist eher ein Dilemma. Der einfache Weg ist unsauber, der saubere eventuell teurer.

Testdaten können auf verschiedenen Wegen in das zu testende System gebracht werden. Für einige Daten ist es z.B. möglich, diese direkt in die Datenbank zu schreiben (Insert) oder über eine systemeigene Schnittstelle zu gehen.

Die Variante »direkt in die Datenbank schreiben« geht (eventuell) schneller, ist aber fehleranfälliger. Denn hier greifen nur Prüfmechanismen der Datenbank (z.B. Pflichtfelder).

Die Variante »über Systemschnittstellen eingeben« hat dagegen den Vorteil, dass in der Schnittstelle implementierte Prüfungen wirken können (z.B. zulässige Wertebereiche, Pflichtfelder) und so bereits einige Fehleingaben auffallen. Der Nachteil: Steht die Schnittstelle wegen eines aktuellen Fehlers nicht zur Verfügung, können die Testdaten nicht erstellt werden.

Welche Möglichkeit sich wann empfiehlt, ist im Einzelfall abzuwägen. Grundsätzlich kann man sich daran orientieren, was für Daten und wie viele davon wann und zu welchem Zweck eingegeben werden sollen. In höheren Teststufen empfiehlt es sich jedoch grundsätzlich, auf die vom zu testenden System implementierten Funktionalitäten zurückzugreifen und »unsaubere« Methoden, wie direktes Manipulieren der Datenbank, zu unterlassen.

Abwägung zwischen Verwenden von Produktionsdaten und Erstellen von synthetischen Daten

Grundsätzlich stellt sich die Frage, ob man Daten aus der Produktion kopieren darf und will oder ob man – möglicherweise etwas aufwendiger – künstlich Testdaten erzeugt. Beide Möglichkeiten haben Vor- und Nachteile und nicht immer stehen Produktionsdaten zur Verfügung.

Eine Entscheidung fällt neben der Berücksichtigung rechtlicher Aspekte auch unter Beachtung des Anwendungszweckes und natürlich der mit dem Gewinnen der Testdaten verbundenen Kosten (siehe dazu auch Abschnitt 5.1.1).

Auch hier handelt es sich weniger um ein Problem als vielmehr um ein Dilemma. Die einfache Lösung, nämlich Echtdaten unverändert zu verwenden, ist gesetzeswidrig, wenn es bedeutet, dass personenbezogene oder

personenbeziehbare Daten zum Einsatz kommen. Alle anderen Lösungen stellen sich aufwendiger dar (Anonymisierung oder synthetische Testdaten).

Das Henne-Ei-Problem des Testdatenmanagements

Das *Henne-Ei-Problem des Testdatenmanagements* [Albr14] tritt bei Neuentwicklungen, seltener bei Weiterentwicklungen auf.

Es tritt auf, weil

- aufeinander aufbauende oder sich bedingende Funktionalitäten (z.B. User Stories) nicht gleichzeitig implementiert und getestet werden können und
- voneinander abhängige Funktionalitäten »gegenseitig« darauf warten, getestet zu werden, und
- wenn Softwareentwicklung und vorbereitende Testaktivitäten oder gar Testläufe parallel stattfinden (z.B. in agilen Softwareentwicklungsprojekten, wenn Tester und Entwickler gemeinsam, also gleichzeitig, dieselben User Stories bearbeiten).

Es besteht darin,

- dass der Test für die Testdurchführung einige Testdaten (z.B. Basisdaten, Stammdaten) aus Systemteilen benötigt, die noch nicht implementiert (oder noch nicht getestet und freigegeben) sind, oder
- dass wir für das Anlegen von Testdaten in höheren Teststufen die Funktionalitäten verwenden wollen, die das zu testende System für diese Daten vorsieht – die Funktionalitäten aber noch nicht zur Verfügung stehen.

Das kann dazu führen, dass wir für testvorbereitende Arbeiten bereits auf Funktionalitäten zugreifen müssten, die es im System noch nicht gibt (oder die es offiziell noch nicht gibt, weil sie noch nicht getestet und daher nicht freigegeben wurden). In diesen Fällen muss man den Test dieser Funktionalitäten zunächst zurückstellen und ein oder zwei Sprints später testen; bis dahin bleiben die User Stories offiziell offen.

Gerade in höheren Teststufen wie dem Systemtest sind wir darauf bedacht, die Testdaten mit Mitteln des zu testenden Systems zu erzeugen. Das heißt, wir wollen – wann immer möglich – die Funktionalitäten nutzen, die die zu testende Anwendung für die Eingabe der von uns als Testdaten anzulegenden Daten bietet.

Beispiel: Artikel und Kunden für Mein Onlineshop

Mein Onlineshop bietet dem Betreiber des Onlineshops zwei Schnittstellen, über die die

anzubietenden Artikel in das System eingegeben werden können. Für das Erfassen einzelner Artikel steht eine Eingabemaske zur Verfügung. Größere Mengen werden aus einer Datei eingelesen (z.B. beim Neuanlegen des Shops).

Im Systemtest pflegen wir in der Rolle des Shopbetreibers die initiale Menge an Artikel-Testdaten daher per Datei ein. Weitere Artikel erstellen wir bei Bedarf über die Eingabemaske.

Gleiches gilt für die anzulegenden Kunden. (Hier jedoch mit dem Unterschied, dass die Dateischnittstelle für den Shopbetreiber vorgesehen ist, die Eingabemaske zum Registrieren eines Neukunden dagegen für die Benutzung durch Kunden des Onlineshops.)

Soweit zum Idealfall. In der Realität kann es zum Beispiel in agilen Softwareentwicklungsprojekten durchaus vorkommen, dass Funktionalitäten noch nicht fertig entwickelt sind. Diese stehen uns dann im Test auch nicht für das Erzeugen von Testdaten zur Verfügung.

Beispiel: Artikel für Mein Onlineshop

Mein Onlineshop wird unter Verwendung agiler Methoden Schritt für Schritt entwickelt. Aufgrund enger Terminpläne und Verzögerungen im Projektablauf passt das Team die Prioritäten der User Stories an. In der Folge stellen sie die Eingabemaske für das Erfassen einzelner Artikel zugunsten anderer kundenbezogener Funktionalitäten zurück. Für Testzwecke benötigte einzelne Artikel werden vorerst über die Dateischnittstelle ins System eingegeben (Workaround).

Das Entwicklungsteam, das die o.g. Schnittstellen zum Erfassen der Kundendaten entwickeln will, kann seine User Stories krankheitsbedingt nicht fertigstellen. Somit stehen keine spezifizierten Funktionalitäten zum Einpflegen von Kunden zur Verfügung. Die Tester müssen die Kunden also anderweitig ins System bekommen.

Welche Lösung gibt es für das *Henne-Ei-Problem?*

Die einfachste, aber sicher nicht immer umsetzbare Variante: Wir warten, bis alle für den Test bzw. die Testdatenerstellung benötigten Funktionalitäten fertiggestellt sind, und testen alle diese voneinander abhängigen Funktionalitäten gemeinsam.

Ein anderer Aspekt dieses *Henne-Ei-Problems* bezieht sich auf die Umgebungsdaten. Die Testumgebungen der verschiedenen Teststufen unterscheiden sich meist in ihrer Realitätsnähe (z.B. weniger Server als in der Produktion, kein Load Balancer u.Ä.). So finden sich häufig erst im

Systemintegrationstest Umgebungen, die der Produktionsumgebung gleichen oder zumindest nahekommen. Damit unterscheiden sich jedoch die in den verschiedenen Testumgebungen benötigten und verwendeten Umgebungsdaten. Jedes Einrichten und Installieren einer neuen Testumgebung stellt damit quasi einen kleinen Installationstest dar – und ein Herantasten an korrekte Konfigurationen, d.h. auch an die korrekten Umgebungsdaten.

Aufwand für Konsistenz der Testdaten, vor allem in Integrationstests und über Systemgrenzen hinweg

Testdaten müssen in sich konsistent sein. Dies ist insbesondere bei komplexen Datenmodellen von großer Bedeutung. Konsistenz sicherzustellen verursacht einen hohen Aufwand. Besonders dann, wenn sich der Testdatenbestand über Systemgrenzen hinweg erstreckt.

Der Ersteller und auch die Benutzer der Testdaten, müssen sich fachlich und technisch gut auskennen. Sie müssen wissen, welche Daten »zusammengehören« und welche der zu testenden Systeme oder Systemteile darauf zugreifen.

Des Weiteren kann sich die Suche nach geeigneten Testdatensätzen als sehr aufwendig erweisen, wenn komplexe Datenstrukturen benötigt werden. Denn in diesen Fällen ist es schwierig, genau den passenden zu finden.

Das Problem wird noch interessanter, wenn man Echtdaten verwenden will und diese dazu anonymisiert. Gehören als Schlüssel verwendete Daten zu den zu anonymisierenden, dann müssen diese in allen Systemen in derselben Art und Weise verändert werden. Was für eine Herausforderung an die jeweiligen Systemverantwortlichen …

Abgrenzung zwischen Testsystem und Produktionssystem

Probleme mit Testdaten können sich daraus ergeben, dass absichtlich oder unabsichtlich eine Verbindung zwischen der Testumgebung und der Echtumgebung besteht.

Worin das Problem liegt, verdeutlichen die folgenden Beispiele.

Das zu testende System ist an die Infrastruktur aus dem Produktionssystem angebunden, z.B. eine Druckstraße. Denn in Testumgebungen spart man sich üblicherweise den unverhältnismäßig hohen Aufwand und verzichtet auf eine extra Test-Druckstraße.

So kann es passieren, dass Ausdrucke, die im Rahmen von Tests erstellt wurden (d.h. Ausgabedaten), tatsächlich mit der Post versandt werden. Im besten Fall sind diese Schreiben aufgrund der gewählten Testdaten als solche erkennbar und werden nicht an echte Personen zugestellt.

Beispiel Systemintegrationstest: Testdrucke in der Echtumgebung, da Infrastruktur in Testumgebung nicht verfügbar

Mein Onlineshop bietet seinen Kunden die Möglichkeit, Online-Gutscheine zu erwerben. Die Gutscheine werden von der Fachabteilung erstellt und im System gespeichert. Sie enthalten bereits Verlinkungen auf konkrete Angebotsseiten des (echten) Shops.

Beispiel Systemintegrationstest: Hyperlinks auf Echtumgebung für Test umlenken

Erwirbt ein Käufer in der Testumgebung einen Gutschein, so enthält der Gutschein also Hyperlinks, die auf die Produktionsumgebung verweisen. Da Testumgebung und Produktionsumgebung getrennt gehalten werden und auch über verschiedene Datenbestände verfügen, »funktioniert« der Link aus der Testumgebung in die Echtumgebung nicht.

Bei der Testdurchführung muss der Link daher auf die Testumgebung umgelenkt werden, bevor der Test fortgesetzt werden kann. Andernfalls würde der Tester feststellen, dass der Link nicht wie gewünscht funktioniert, er also nicht auf die Angebotsseite weitergeleitet wird. Fällt nicht auf, dass der Tester irrtümlich den Link auf die Produktionsumgebung verwendete, meldet er möglicherweise einen Fehler, der kein Fehler ist (Beispiel für falsch negatives Testergebnis).

Wäre der Gutschein jedoch von vornherein mit einem Link auf die Testumgebung ausgestattet, liefe der Test durch. Wenn dann aber versäumt würde, diese Verzweigung für das Produktionssystem umzustellen, tritt im Echtbetrieb ein Fehler auf, obwohl im Test alles in Ordnung war.

Vernachlässigung der Metadaten

In Abschnitt 2.7 hatten wir uns kurz mit Metadaten befasst. Hier geht es nun darum, welche Folgen das Vernachlässigen der Metadaten nach sich zieht.

Ohne Metadaten besteht laut [Ende12] die Gefahr, dass verschiedene Parteien auf dieselben Daten zugreifen und sich dadurch in ihren Testaktivitäten gegenseitig stören. Wir erinnern uns: Metadaten sollen Informationen über die Eigentümer der Daten beinhalten.

Testdaten ohne Metadaten sind nur von begrenztem Nutzen. Enderlein drückt es drastischer aus: Sie sind »ohne Metadaten wertlos« [Ende12]. Rufen wir uns in

Erinnerung, dass wir die Testdaten benötigen, um bestimmte fachliche Szenarien mittels Testfällen durchzuspielen. Dann sollten wir für das jeweilige Szenario auch die passenden Daten verwenden können. Diese fachlich relevanten Informationen sollten aus den Metadaten hervorgehen (und das Big Picture aus dem Testdatenkonzept).

Metadaten werden jedoch auch als Steuerungsinformationen für Testwerkzeuge benötigt. Ohne oder mit nicht ausreichend gepflegten Metadaten sinkt der Nutzen, der sich aus der Verwendung der Werkzeuge ergeben soll.

Sind die Daten einmal nicht mehr auf dem aktuellen Stand, wird es schwer, das wieder aufzuholen.

Die Qualität von Testdaten lässt sich schwer messen

Wir wissen, was wir unter guten Testdaten verstehen. Wie aber können wir die Qualität von Testdaten nachvollziehbar messen?

Ein guter Testdatenbestand enthält realistische bzw. realitätsnahe Daten und umfasst genau die benötigte Menge an Testdaten – nicht mehr und nicht weniger. Darüber hinaus erfüllt ein guter Testdatenbestand alle in Kapitel 3 genannten Anforderungen. Zusätzliche Informationen über die Testdaten liefert eine geschickt definierte Menge an Metadaten.

Diesem nicht trivialen Problem widmet sich Johannes Held in seiner Dissertation [Held16]. Wir gehen in Kapitel 12 detaillierter darauf ein.

Im Falle fehlender oder unzureichender Datensicherung droht Datenverlust (Hardware)

Testdatenbestände müssen vor Verlust durch kaputte Hardware oder versehentlichen Löschungen bewahrt werden. Daher sollten regelmäßig Sicherungen des Testdatenbestands angelegt werden.

Dies ist nötig, um im Notfall schnell einen sauberen Testdatenbestand wieder bereitstellen zu können. Eine Sicherung könnte ggf. auch verwendet werden, um mehrere Testumgebungen mit demselben Datenbestand versorgen zu können.

Testdaten müssen vor Zerstörung geschützt werden (fehlerhafte Handhabung)

Besondere Aufmerksamkeit sollte den Testdaten zukommen, die für automatisierte Testfälle benötigt werden. Schützen Sie diese Testdaten mittels

geeigneter technischer und/oder organisatorischer Maßnahmen vor versehentlicher Zerstörung.

Folgende Lessons Learned belegt auch die Notwendigkeit, Testdaten hinsichtlich ihrer Verwendung zu trennen und ggf. zu reservieren.

Trennen von Testdaten zwecks Vermeidung von Beeinträchtigungen insbesondere automatisierter Tests Lessons Learned

Ein Kundenlogin, mit dem nach dem Einloggen das Kaufen eines Buches getestet wird, sollte nicht gleichzeitig in einem Testfall verwendet werden, der das Ändern des Passwortes prüft. Wieso nicht? Scheitert das Ändern des Passwortes wegen eines Fehlers im zu testenden System, ist dieser Kunde für das Testen verloren. Denn das System akzeptiert das alte Passwort nicht mehr, ein neues ist aber nicht gespeichert. Das führt dazu, dass alle nachfolgenden automatisiert ausgeführten Testfälle mit diesem Kunden bereits beim Login fehlschlagen.

Für die Tests von Anwendungsfällen wie »Ändere Passwort« u.Ä. sollten eigens dafür erstellte Accounts verwendet werden.

4.1.5 Organisatorische Problemstellungen

Diese Testdatenprobleme sind genau genommen Probleme, die weniger an den Daten als vielmehr in deren Management liegen.

Steigende Anforderungen an Testdaten aufgrund gesetzlicher Regelungen

Testdaten müssen neben inhaltlichen Anforderungen auch gesetzliche erfüllen. Der Datenschutz ist nur ein Aspekt davon. Es ist Aufgabe des Testdatenmanagements, dafür zu sorgen, dass sowohl die Testdatenbestände als auch der Umgang mit ihnen die Anforderungen erfüllen, oder bei Nichterfüllung die Risiken im Auge zu behalten. (Beispiele aus [Muru13] sind PCIDS, Gramm-Leach-Bliley Financial Act (GLBA), BASEL II, Dodd Frank Act, Solvency II.)

Testdatenabdeckung: Mangelnde Nachverfolgbarkeit zwischen Testfall, Testdaten und fachlichen Anforderungen

Eine große Herausforderung im Test liegt darin, die notwendige Nachverfolgbarkeit zwischen den Anforderungen aus »dem Business« und den Testfällen herzustellen und – vor allem auch – aufrechtzuerhalten.

Erschwerend kommt hinzu, dass man eine solche Nachverfolgbarkeit ebenso für die Testdaten benötigt: Welche Testdaten werden durch welchen Testfall oder welche Testfälle benutzt? Welche Testfälle sind demzufolge betroffen, wenn Änderungen an bestimmten Testdaten vorgenommen werden?

»No traceability between test data to test cases to business requirements [are] leading to issues on the test data coverage for a particular test case« [Muru13]. Demnach stört ein Mangel an Nachverfolgbarkeit die Testdatenabdeckung für einen bestimmten Testfall.

Praxisbericht

Vor Jahren arbeitete ich in einem Testteam, das einen Teil der benötigten Testdaten quasi als Service durch ein anderes Team bereitgestellt bekam. Es handelte sich dabei um Umgebungsdaten, Stammdaten u. Ä. Da der Grundsatz galt, möglichst wenige Daten neu anzulegen, mussten bestehende angepasst werden.

Wann immer Änderungen an diesen zentral verwalteten Daten nun vorzunehmen waren, beriet ein Gremium die geplanten Modifikationen. Wir schätzten unseren Aufwand zur Anpassung unserer Testfälle, den diese Änderungen bewirken würden. Wir prüften, welche unserer Testfälle diese Daten verwendeten und wie viel Zeit es uns kosten würde, diese auf die geänderten Daten »umzustellen«. Unsere Testfälle verwalteten wir in einem Testwerkzeug.

Manchmal war eine große Anzahl an Testfällen betroffen, aber die Änderung einfach und daher schnell eingepflegt. Zum Beispiel in den Fällen, in denen wir eine bestimmte Funktionalität des Testwerkzeugs nutzten.

In anderen Fällen fiel der Aufwand so hoch aus, dass das Gremium abweichend vom o.g. Grundsatz das Erstellen eines neuen Datensatzes (statt der Anpassung eines bestehenden) genehmigte.

Nachvollziehbarkeit: Anforderung – Testdaten – Testfall und Version der Testdaten – Version der Software – Version der Anforderung

Eine der Herausforderungen im Test besteht darin, eine gute Nachvollziehbarkeit zwischen den Anforderungen und den sie testenden Testfällen zu erreichen und vor allem auch zu erhalten.

Wir wollen ausgehend von der Anforderung wissen, durch welche Testfälle sie abgedeckt wird und wie die Testresultate ausfielen. Diese Angaben zeigen uns, wie

gut die Anforderung durch den Test abgedeckt und ob die Anforderung erfüllt ist oder nicht.

Darüber hinaus wollen wir ausgehend von den Testfällen wissen, welche Anforderungen sie abdecken und welche Testdaten die Testfälle dafür benutzen und welche sie erzeugen.

Praxisbericht

In einem Vorstellungsgespräch wurde ich einmal gefragt, wie ich in einem neu angeschafften Testmanagementwerkzeug die Anforderungen (Requirements) und Testfälle strukturieren würde, um eine gute Nachvollziehbarkeit zu erreichen. Im Gespräch hatte ich den Eindruck, dass mindestens einer der Fragesteller im Hinterkopf hatte, beide Ordnerstrukturen identisch aufzubauen.

Ich antwortete sinngemäß:

Da eine Anforderung durch mehrere Testfälle abgedeckt werden kann und ein Testfall auch mehrere Anforderungen testen kann, besteht zwischen den Anforderungen und den Testfällen eine m:n-Beziehung. Daher ist es kaum möglich und wenig erstrebenswert, auf beide, also sowohl auf die Anforderungen als auch auf die Testfälle, exakte dasselbe Ordnungskriterium anwenden zu wollen. Demzufolge würde ich beim Anlegen der Ordnerstruktur zur Ablage der Testitems von dem ausgehen, wonach bzw. worin ich eher suchen würde. Oberste Priorität beim Einrichten der Struktur hat für mich das Wiederfinden. Als oberstes Ordnungskriterium würde ich die zu testende Funktionalität wählen.

Offensichtlich wurde meine Antwort nicht als ganz abwegig empfunden, denn ich bekam den Job. Und durfte mit dem neuen Werkzeug arbeiten :)

Für das Werkzeug bedeutete das, die (hier nur funktionalen) Anforderungen wurden nach Funktionalität gruppiert, die Testfälle mit einigen Abweichungen im Wesentlichen auch. Denn es gab Testfälle, die funktionsunabhängige Dinge testen, oder Testfälle, die wir zum Anlegen von Testdaten verwendeten. Diese fassten wir jeweils in einem eigenen Ordnerbereich zusammen und verlinkten diese selbstverständlich auf die abgedeckten Anforderungen.

Für das Problem der Nachvollziehbarkeit gibt es meines Erachtens keine allgemeingültige Lösung. Wichtig ist, dass wir die Anforderungen und Testfälle so ablegen, dass wir sie wiederfinden. Das heißt, dass wir zu jeder Anforderung einen Verweis auf die sie abdeckenden Testfälle haben, idealerweise auch umgekehrt.

Wenn Sie zur Verwaltung ihrer Testfälle ein Testwerkzeug einsetzen, in dem Sie auch Anforderungen (zumindest als Kopie) zu Testzwecken halten, dann achten Sie darauf, welche Verlinkungen zwischen Testfällen und diesen Anforderungen möglich sind. Es empfiehlt sich, die Links zwischen Testfällen und Anforderungen ständig aktuell zu halten und aktiv zu pflegen.

Testdaten in mehreren Testumgebungen verwalten

Mehrere Umgebungen gleichzeitig

In jeder Testumgebung wird ein Testdatenbestand benötigt. Existieren mehrere Testumgebungen, sind diese alle zu versorgen. Dazu muss der Testdatenbestand eventuell angepasst werden (z.B. Umgebungsdaten).

Die Testumgebungen unterscheiden sich zum einen anhand der Teststufe in ihrer Realitätsnähe: Je näher an der Produktion, desto realistischer. Weitere Unterschiede gehen auf die geplante Testart zurück. Für Last- und Performance-Tests sollte eine eigene Umgebung zur Verfügung stehen, um gegenseitige Beeinflussungen z.B. mit dem funktionalen Test auszuschließen. In der Regel haben Lasttester andere Anforderungen an die Mengen der Testdaten als funktionale Tester.

Mehrere Versionen gleichzeitig

Darüber hinaus soll die Version des Testdatenbestands zur Softwareversion passen, die in dieser Umgebung getestet werden soll. Das heißt, im Rahmen des Testdatenmanagements sind mehrere Versionen des Testdatenbestands zu verwalten.

Irrtum, mit der Anschaffung eines TDM-Werkzeugs sei das TDM-Problem gelöst

Bei Recherchen zum Thema Testdatenmanagement fiel mir auf, dass es im Wesentlichen zwei verschiedene Ansichten über bzw. Herangehensweisen an das Testdatenmanagement gibt:

Die eine Gruppe von Publikationen widmet sich dem Thema von der methodischen Seite her. Sie erläutert Best Practices und Rahmenwerke, diskutiert »Challenges« oder Probleme mit Testdaten und deren Management. In Texten dieser Gruppe finden sich neben Erfahrungsberichten und Fallstudien oft auch Hinweise auf Werkzeuge, die die Arbeit mit den Testdaten unterstützen.

Die andere Gruppe sieht Testdatenmanagement als eher technisches Problem. Diese Ansicht wird gemäß [Ende12] durch die Hersteller von Werkzeugen zur

Maskierung und Reduzierung von Testdaten unterstützt. Die Werkzeuge fokussieren auf das Erstellen von Testdaten oder Bearbeiten von Produktionsabzügen. Das Problem dabei: Die meisten derzeit verfügbaren Werkzeuge werden kaum in der Lage sein, alle für ein sinnvolles Testdatenmanagement benötigten Funktionalitäten abzudecken.

Als Leser der Produktbeschreibungen o.g. Tools drängt sich mir oft der Eindruck auf, wenn ich dieses oder jenes Testdatenmanagement-Werkzeug einsetze, sind alle meine Testdatenprobleme gelöst. Ich bezweifle stark, dass es ein Testdatenmanagement-Werkzeug gibt, das den kompletten Testdatenmanagementprozess unterstützt (sozusagen von Requirement bis Retirement) und nicht nur die Arbeit mit den Testdaten.

Wenn Sie bereits Erfahrungen mit verschiedenen Softwarelösungen fürs Testdatenmanagement sammeln konnten und wenn in Ihrem Unternehmen ein strukturiertes Testdatenmanagement betrieben wird, dann wissen Sie, was diese Werkzeuge wirklich leisten (können) und was nicht.[1]

Zu frühe Festlegung auf bestimmte Werkzeuge

In der zu frühen Entscheidung für bestimmte Werkzeuge sieht [Ende12] einen wesentlichen Stolperstein auf dem Weg zu einem guten Testdatenmanagement.

Vor der Auswahl eines Werkzeugs sollen demnach die Anforderungen der Beteiligten geklärt werden (ggf. auch der Vertriebs- und Compliance-Teams). Des Weiteren sind »Ziele, Prozesse und Zuständigkeiten« [Ende12] herauszuarbeiten. All diese Aspekte fließen in die Auswahl eines Werkzeugs ein.

Erfolgte die Anschaffung des Testdatenmanagement-Werkzeugs ohne die oben genannten Informationen, dann besteht die Gefahr, dass die gewählte Lösung wesentliche der benötigten Funktionalitäten nicht umfasst und/oder die gewünschten Prozesse nicht unterstützt. (Zu Anforderungen an Werkzeuge, Kategorien von Werkzeugen und zur Auswahl derselben siehe Kap. 11.)

Zuständigkeiten: Wer erzeugt die Testdaten oder stellt sie bereit?

Unklare Zuständigkeiten oder »überbuchte« Personalressourcen sind eher eine Frage mangelnden oder unzureichenden Testdatenmanagements.

Ein Hindernis beim Erstellen von Testdaten besteht oft darin, dass derjenige, der die Daten benötigt, sich fachlich vielleicht nicht gut genug auskennt, die Daten selbst zu erzeugen. Oder er besitzt keine Zugriffsberechtigungen auf die dafür notwendigen Anwendungen. Derjenige, der über das nötige Wissen und die

Zugriffsmöglichkeiten verfügt, hat oft keine Zeit und/oder sieht sich nicht in der Pflicht. Oder der Mitarbeiter mit dem Know-how und den Zugriffsberechtigungen hat gerade keine Zeit für das Anonymisieren der Produktionsdatenkopie.

Sind Tester oder andere Personen dann gezwungen, selbst Hand anzulegen, führt das im besten Falle dazu, dass die Tester etwas über das System und die Daten lernen. Im schlechten Fall aber können dadurch Testdaten inkorrekt erstellt werden und im Test zu falsch negativen oder falsch positiven Testergebnissen führen (mehr dazu in Abschnitt 4.1.3).

Organisatorischer und zeitlicher Aufwand für Erstellen und Warten von Testdaten

Die Aktivitäten »Testdaten erstellen« und »Testdaten warten« fallen in jedem Softwaretestprojekt an. Der Aufwand dafür hängt u.a. davon ab, wie gut diejenigen, die damit betraut sind, sich fachlich und technisch auskennen. Aber auch die Lebensdauer der Testdaten oder bestimmter Testdatensätze beeinflussen den Pflegeaufwand. »Ewig lebende« Testdaten müssen nur einmal erstellt werden. Testdaten aber, die bereits nach der ersten Verwendung verbraucht sind, müssen viele Male neu erstellt werden.

In jedem Falle aber sind Aufwände im Testprojekt oder im Entwicklungsprojekt in angemessenem Umfang zu berücksichtigen. Nähere Informationen dazu sollte das Testdatenmanagement liefern.

Uneinheitliche Vorgehensweise bei Anforderung und Bereitstellung von Testdaten

Testdaten werden durch verschiedene Teams für unterschiedliche Testzwecke angefordert. Ein uneinheitliches Vorgehen führt zu Verzögerungen im Testprozess. Abhilfe schaffen hier Prozesse inklusive Dokumentvorlagen und Zuständigkeiten. Diese regeln das Anfordern und Bereitstellen von Testdaten (siehe auch [Muru13]).

Fehlendes TDM führt zu hohem Aufwand für die Testdatengewinnung

Dass man Testdaten nicht umsonst bekommt, ist klar. Allerdings wirkt ein fehlendes oder unzureichendes Testdatenmanagement zusätzlich noch kostensteigernd.

Bei [Muru13] heißt es dazu: »Test data identification, extraction and conditioning consume large effort in testing life cycle as much as 12-14% and at times much higher.« Demnach belaufen sich die Kosten für das Identifizieren,

Extrahieren und die Aufbereitung der Testdaten im Lebenszyklus des Tests auf 12 bis 14 % und manchmal entstehen noch weit höhere Kosten. Hier ist zwar die Rede von Testdatengewinnung aus einem Echtdatenbestand, es ist aber davon auszugehen, dass die Kosten für das Erstellen synthetischer Testdaten nicht wesentlich geringer ausfallen.

Wenn Teams für ihre Testzwecke anstelle eines gemeinsamen Testdatenbestands jeweils einen eigenen in Form einer vollständigen Kopie von Produktionsdaten unterhalten, dann verursachen sie damit hohe Kosten für Speicherplatz, Lizenzen und Wartung.

Ohne ausreichendes Testdatenmanagement laufen die verschiedenen Softwareentwicklungsteams Gefahr, nebeneinander statt miteinander an den Testdaten zu arbeiten. Was wiederum zu ineffizientem Vorgehen führt: »With no standard processes followed, (…) results in inefficiency as there are no plans for reuse of the test data artifacts and optimization of data, /environment« [Muru13]. Ein Testdatenmanagement sollte demzufolge nicht nur das Gewinnen der Testdaten, sondern auch deren Wiederverwendung und Optimierung umfassen und auch sonstige Testdatenartefakte und die Testumgebung berücksichtigen.

Unreifes oder fehlendes Testmanagement

»Das Testdaten-Management leidet oft darunter, dass schon das Test-Management im Unternehmen alles andere als ausgereift ist. Häufig stellt sich erst nach der Einführung einer Testdaten-Management-Lösung heraus, dass grundsätzliche Prozesse zur Planung und Konzeption der Testfälle fehlen oder noch nicht etabliert sind« [Ende12].

Wir sind uns sicher einig, dass ein Testteam, das mit einem unreifen Testprozess zu kämpfen hat, sich kaum intensiv und strukturiert um ein Testdatenmanagement kümmern kann.

4.2 Risiken bei Testdaten

Dieses Kapitel betrachtet im Wesentlichen die folgenden zwei Gruppen von Risiken: *Testdaten als Produktrisiko* und *Testdaten als Projektrisiko*. Diese Risiken sollen die Auswirkungen von Problemen mit Testdaten beispielhaft illustrieren. Beide Risiken lassen sich mithilfe organisatorischer und technischer Maßnahmen reduzieren. Wertvolle Anregungen für das Risikomanagement liefert auch Abschnitt 3.1.

Bezogen auf die Testdaten stellt das Einführen eines Testdatenmanagements oder das Betreiben eines vorhandenen eine solche organisatorische Maßnahme dar (die wiederum zu weiteren organisatorischen oder technischen Maßnahmen führen wird). Analog zum Vorgehen im Testprojekt sollte im Bereich des Testdatenmanagements ebenfalls ein Risikomanagement betrieben oder im Risikomanagement des Testprojekts berücksichtigt werden.

Nicht zu vernachlässigen als Unternehmensrisiko sind auch Verletzungen des Datenschutzes oder Unternehmensgeheimnisse, die ungeschützt in Testumgebungen liegen.

4.2.1 Fehlende und fehlerhafte Testdaten als Produktrisiko – unentdeckte Fehler

Fehlende und fehlerhafte Testdaten stellen bezüglich der zu testenden Anwendung ein *Produktrisiko* dar. Wir sprechen vom Risiko unentdeckter Fehler im zu testenden System oder gar im produktiven System.

Der Test soll unter anderem zeigen, dass das System unter Test funktioniert wie angefordert, dass also alle geforderten Funktionalitäten implementiert sind und korrekt arbeiten.

Wenn im Test mangels geeigneter Testdaten Teile der Anwendung nicht ausgeführt werden, bleiben dort enthaltene Fehler unentdeckt und treten womöglich im produktiven Einsatz zutage.

Die so erhaltenen Testergebnisse sind unbrauchbar.

Beispiel: Fehler im Echtbetrieb – Gepäckförderanlage verweigert Arbeiten am Schalttag

Sie erinnern sich sicher an die Gepäckförderanlage eines Flughafens, die an einem Schalttag die Arbeit verweigerte. Koffer mussten per Hand sortiert werden, einige blieben liegen (siehe [Berk16], [Anon16a]). In diesem Falle fehlten mit dem Testdatum <Schalttag> nicht nur Testdaten, sondern mindestens ein Testfall.

Praxisbericht: Testdurchführung nicht möglich

Als Tester in einem Team, das Software sowohl erstellte als auch für deren Betrieb verantwortlich war, hatte ich für jede unternehmensweite Softwareversion zu testen, ob die Anbindung bestimmter fachlicher Anwendungen an die von meinen Teamkollegen betreute Formularverwaltung funktioniert. Es waren ein oder zwei Dutzend Anwendungen zu testen.

Dazu meldete ich mich mit einer von der Fachabteilung vorgegebenen Test-Benutzerkennung bei der Fachanwendung an und führte bestimmte Schritte in der Anwendung aus, bis ich die Schnittstelle zur Formularverwaltung erreicht hatte. Dort rief ich ein bestimmtes Formular auf, füllte es aus und druckte mindestens eines auch aus.

Es kam jedoch vor, dass die Kennung gesperrt war, sodass ein Anmelden damit nicht möglich war. Gelang es innerhalb des oft nur dreitägigen Testzeitraumes nicht, eine nutzbare Kennung zu erhalten, dann blieb die Verbindung zwischen dieser Fachanwendung und der Formularverwaltung ungeprüft. Im schlechtesten Falle hätte das bedeuten können, dass die Benutzer der Fachanwendung keine Schreiben hätten erstellen und versenden können – und zwar so lange, bis der Fehler behoben oder ein Workaround gefunden war.

Indikatoren für Risiken dieser Art können sein:

- Datumswerte sind nicht als Datentyp Datum implementiert, sondern als Zahlen (integer oder Ähnliches).
- Es gibt nur eine sehr geringe Anzahl an Testfällen pro Testobjekt (Methode, Anwendungsfall oder Funktionalität), z.B. nur Geradeaus-Testfälle, keine Negativ-Testfälle.
- Testen von Querschnittsfunktionen oder Systemgrenzen überschreitend

Die Gegenmaßnahmen fallen ähnlich vielfältig aus, hier ein paar Beispiele:

- Angemessene Testaufwände planen und angemessene Ressourcen bereitstellen, damit die Testenden nicht aufgrund akuten Zeitmangels auf Testfälle verzichten müssen. Auch Aufwand für notwendige Kommunikation ist einzuplanen.
- Alle Testenden sollten Testspezifikationsmethoden kennen und anwenden können.
- Test-Checklisten einsetzen, z.B. Checklisten für mögliche Eingabewerte.
- Aufmerksamkeit auch auf Testdaten lenken und alle Beteiligten dafür sensibilisieren, welche Auswirkungen nicht kommunizierte Änderungen an Testdaten haben können.

4.2.2 Fehlende und fehlerhafte Projektrisiko als Projektrisiko – Verzögerungen und spät entdeckte Fehler

Wer Software testet, hat mit an Sicherheit grenzender Wahrscheinlichkeit bereits mindestens einmal auf die Bereitstellung von Testdaten warten (oder aber die Testdaten selbst erstellen) müssen, bevor er mit der Testdurchführung loslegen konnte.

Fehlende Testdaten führen zu Verzögerungen in der Testdurchführung und zu verspäteter Bereitstellung der Testergebnisse. Das kann zur Folge haben, dass spät entdeckte Fehler nicht mehr für diese Version behoben werden können und mit ausgeliefert werden müssen.

Folgender Erfahrungsbericht kann auch als Beispiel für fehlerhafte Testdaten dienen:

Praxisbericht: Testen einer Webanwendung mit CMS-Beteiligung

»Wieso fällt das jetzt erst auf?«

Nun, weil jemand im Content-Management-System Änderungen vornahm, die nicht kommuniziert waren. Nach Bekanntwerden der Änderung musste der Testlauf wiederholt werden. Während der erste Lauf (auf den alten Daten) erfolgreich endete, schlug die zweite Testdurchführung (auf den neuen Daten) fehl.

Wie kann man dem Vorbeugen? Zum Beispiel mit diesen Maßnahmen:

- Alle Voraussetzungen zur Testdurchführung frühzeitig und sorgfältig planen.
- Bereitstellung der Testdaten frühzeitig regeln.
- Fachleute aus der Fachdomäne einbeziehen.
- Kommunikationsbedarf (Anlass, Beteiligte, Inhalt) feststellen und darauf achten, dass Änderungen mitgeteilt werden.

4.3 Zusammenfassung

In diesem Kapitel wurden im Zusammenhang mit Testdaten möglicherweise auftretende Probleme vorgestellt. Diese wurden unterschieden in

- *Probleme, die sich auf den Faktor Mensch* zurückführen lassen,
- *Probleme, die in den Testdaten selbst* liegen,
- *Probleme aufgrund fehlerhafter, ungeeigneter oder gänzlich vergessener Testdaten*,

- *Probleme in der Gewinnung, Herstellung und Wartung von Testdaten* sowie
- *organisatorische Problemstellungen*.

Alle vorgenannten Punkte stellen Herausforderungen dar, die wir – bis auf das fehlende Testmanagement – im Rahmen des Testdatenmanagements lösen oder zumindest mildern können.

Die Risiken in Bezug auf Testdaten lassen sich im Wesentlichen entweder den Produktrisiken (*unentdeckte Fehler*) oder den Projektrisiken (*Verzögerungen und spät entdeckte Fehler)* zuordnen. Auch sie sind Gegenstand des Testdatenmanagements.

Zum Weiterlesen

Eine sehr umfassende Sammlung berühmter Softwarefehler und ihrer Folgen finden Sie z.B. hier:

[Huck99] Huckle, Thomas: Kleine BUGs, große GAUs. Vortrag am 2.12.1999, *http://www5.in.tum.de/~huckle/bugs.html*, 02.01.2016.

[Huck15] Huckle, Thomas: Collection of Software Bugs. Last modified April/14/2015, *http://wwwzenger.informatik.tu-muenchen.de/persons/huckle/bugse.html*, 02.01.2016.

Index

S

T

U

V

Z