

Generierung von Massendaten
und Testausprägungen

Paare bilden



Matthias Scholze

Je komplexer das Programm, desto größer die für seinen gründlichen Test erforderliche Datenmenge. Relativ schnell überschreitet sie einen handhabbaren Umfang. Datengeneratoren und kombinatorische Tricks helfen, die Aufgabe in den Griff zu bekommen.

In vielen Softwareunternehmen ist derzeit eine zunehmende Etablierung entwicklungsbegleitender Qualitätssicherung festzustellen. Hierbei stehen QS-/Testmanagement, Review von Dokumenten und Code sowie die Durchführung von Funktions- und Performanceprüfungen im Vordergrund. Die Einrichtung eines Testdatenmanagements betrachten jedoch die meisten Firmen derzeit nicht oder sie schätzen es auf Grund der Komplexität als schwer umsetzbar ein.

Sinnvolle Daten sind aber ein wesentlicher Baustein für die erfolgreiche Durchführung von Funktions- und Performance-tests. Geeignete Verfahren zu

ihrer Generierung können den dafür nötigen Aufwand und damit die Testvorbereitung erheblich reduzieren. Außerdem lässt sich eine bessere Qualität der Funktionstests durch höhere Abdeckung erzielen, und realistische Leistungsmessungen basierend auf produktionsnahen Daten werden möglich.

Zwei typische Projektsituationen zeigen, wo die Generierung von Testdaten hilft:

– Ein Dienstleistungsbetrieb möchte ein neues IT-System zur Verwaltung und Steuerung seiner Mitarbeiter einführen. In der Evaluierungsphase gilt es zu verifizieren, ob es die Daten von 10 000 Mitarbeitern hinreichend

schnell verarbeitet. Die Aufgabe besteht darin, die Daten für jeden Angestellten zu erstellen und ihm jeweils einen Standort sowie einige Qualifikationen zuzuordnen, Projekte anzulegen und diesen anschließend Mitarbeiter zuzuteilen.

– Ein Versandhandel möchte die Artikelsuche in seinem Portal möglichst vollständig testen. Anhand der Eingabefelder und -werte der Suchmaske ergeben sich 73 600 kombinatorische Suchvarianten. Wie lässt sich das Portal trotz dieser hohen Anzahl effizient testen?

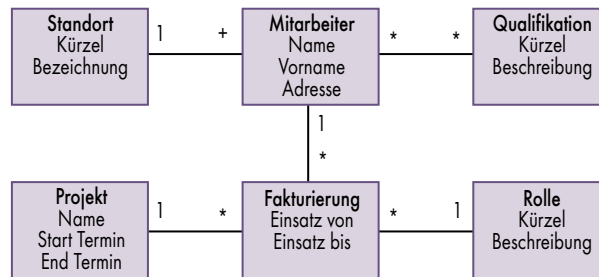
Zur Bewältigung der Herausforderungen in beiden Szenarien gibt es verschiedene später besprochene Strategien. Für die Durchführung von Funktions- und Performance-tests muss das Backend des untersuchten IT-Systems Wirk- und Produktivdaten bereitstellen. Die Verwendung von Produktionsdaten ist nur in den seltensten Fällen möglich. Deshalb müssen Daten mit den gewünschten Eigenschaften erzeugt werden. Dies geschieht auf der Basis eines Datenmodells (s. Abb. 1) und eines Mengengerüsts. Letzteres

gibt an, wie viele Instanzen einer Entität und entsprechende Relationen erforderlich sind.

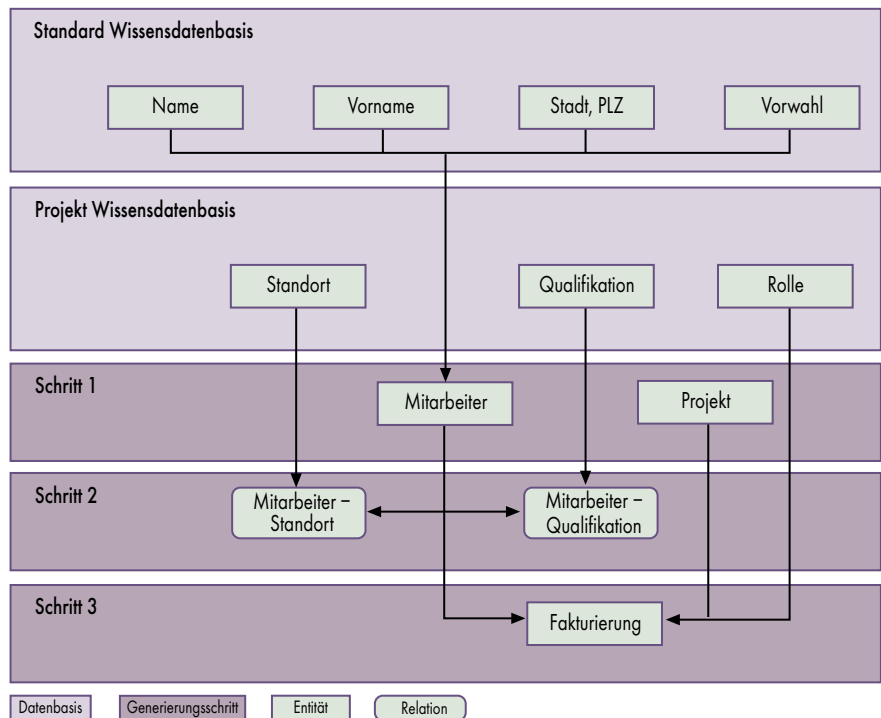
Nicht alles gleichzeitig testen

Für Funktionstests lassen sich die Backend-Daten aus den auszuführenden Testfällen ableiten. So braucht man beispielsweise zum Prüfen einer Überweisung zwei Konten, um einen Betrag belasten und gutschreiben zu können. Ein Performancetest wiederum basiert auf parallel ausgeführten funktionalen Tests. Wegen der Parallelisierung sind mehr Testdaten nötig als für den Funktionstest, denn IT-Systeme führen bestimmte Funktionen nicht parallel aus. So sind etwa simultane Überweisungen von einem Konto nicht möglich, da Online-Banking-Systeme keine gleichzeitigen Anmeldungen pro Konto zulassen. Des Weiteren werden produktionsrelevante Datenmengen benötigt, um das wirkliche Transaktionsverhalten von Komponenten wie Applikationsserver oder Datenbank ermitteln zu können. Bei einer großen Zahl von Testiterationen ist zu beachten, dass sich das Datenvolumen durch Testfälle mit schreibenden Transaktionen verändert. Einen besonderen Typus stellen nur einmal benutzbare Daten dar („konsumierte“ Daten). Bekanntestes Beispiel dafür sind die Transaktionsnummern beim Online-Banking.

Zurück zum ersten Szenario: Ein Dienstleistungsunternehmen möchte eine Anwendung zur Verwaltung und Steuerung seiner Mitarbeiter und Projekte einführen. In der Evaluierungsphase will es verifizieren, dass diese bei



Ein Datenmodell ordnet jedem Mitarbeiter unter anderem Qualifikationen und Projekte zu (Abb. 1).



Ein Generierungsplan legt fest, in welcher Reihenfolge die Testdaten zu erstellen sind (Abb. 2).

10 000 Mitarbeiterdaten schnell genug arbeitet. Der Dienstleister stellt die Informationen für 20 Standorte, 10 Rollen und 300 Qualifikationen sowie weitere Mengenangaben zur Verfügung. Für das Backend des IT-Systems sind daraus Daten zu erstellen, die dem Modell in Abbildung 1 entsprechen.

Aus den Mengenangaben des Kunden und des Modells ergeben sich unter anderem folgende Daten: 1000 Projekte pro Jahr, 350 000 Mitarbeiterqualifikationen (bei durchschnittlich 35 Qualifikationen pro Mitarbeiter), 10 000 Mitarbeiterstandorte. (Jeder Mitarbeiter ist genau einem Standort zugeordnet.) Solche Anforderungen an die bereitzustellenden Testdaten lassen sich nicht mehr als Nebentätigkeit im normalen Projektgeschäft erfüllen. Ein händisches Erstellen oder der Einsatz des „Allzwecktools“ Excel scheiden ebenfalls aus. Spezifizieren und Generieren der Testdaten sollten daher separater Bestandteil des Projektplans sein. Weiterhin muss

der Einsatz dafür geeigneter Werkzeuge eingeplant werden.

Die Erstellung von Testdaten mit komplexen Datenstrukturen und -konstellationen erfolgt in mehreren Schritten. So lassen sich Mitarbeiter nur Projekten zuordnen, wenn sie eine passende Qualifikation besitzen – deshalb muss man im ersten Schritt Instanzen der Mitarbeiter erzeugen, im zweiten die Beziehung zu den Qualifikationen und im dritten die Relation zwischen Mitarbeiter und Projekt auf Basis des Qualifikationsprofils.

Mehrere Schritte zu den Daten

Im ersten Schritt sind alle Daten zu erstellen, die sich aus Basisdatentypen oder Informationen der Wissensdatenbanken zusammensetzen. In den folgenden Schritten erzeugt man die Daten, die als Wissensdatenbank die vorher er-

IX-TRACT

- Beim Testen großer oder komplexer Anwendungen können die erforderlichen Datenmengen so groß werden, dass manuelles Testen nicht praktikabel ist.
- Durch das Reduzieren der Testfälle auf solche mit unterschiedlichen Parameterpaaren sind statt mehrerer zehntausend weniger als einhundert Tests nötig.
- Automatisierung bringt gegenüber manueller Ausführung der Tests schon beim zweiten Mal einen Zeitgewinn.

stellen verwenden. Abbildung 2 zeigt einen Generierungsplan für das obige Beispiel. Zur Erstellung der Daten sind in diesem Fall drei Schritte notwendig. So lassen sich die Daten für die Zuordnung von Qualifikationen zu einem Mitarbeiter erst generieren, wenn die Mitarbeiterdaten vorhanden sind.

Im Artikel „The Combinatorial Design Approach to Automatic Test Generation“ [1] stellen die Autoren Methoden zur automatischen Generierung von Testausprägungen auf Basis der Kombinatorik vor. Ausgehend von der Annahme, dass Fehler in den meisten Fällen bei der Kombination zweier Parameter entstehen, betrachten sie nur Konstellationen, die möglichst viele neue Paarkombinationen enthalten.

Die Pairwise-Methode sei im Folgenden an einem Beispiel erläutert: Es geht um den Test einer Eingabemaske mit drei Feldern. Jedes davon kann zwei Werte annehmen. Für den vollständigen Test der Maske ergeben sich daraus die acht Kombinationen (Testausprägungen) in Tabelle 1.

**TABELLE 1:
KOMPLETTE TESTDATEN**

Parameter X	Parameter Y	Parameter Z
1	A	I
2	A	I
1	B	I
2	B	I
1	A	II
2	A	II
1	B	II
2	B	II

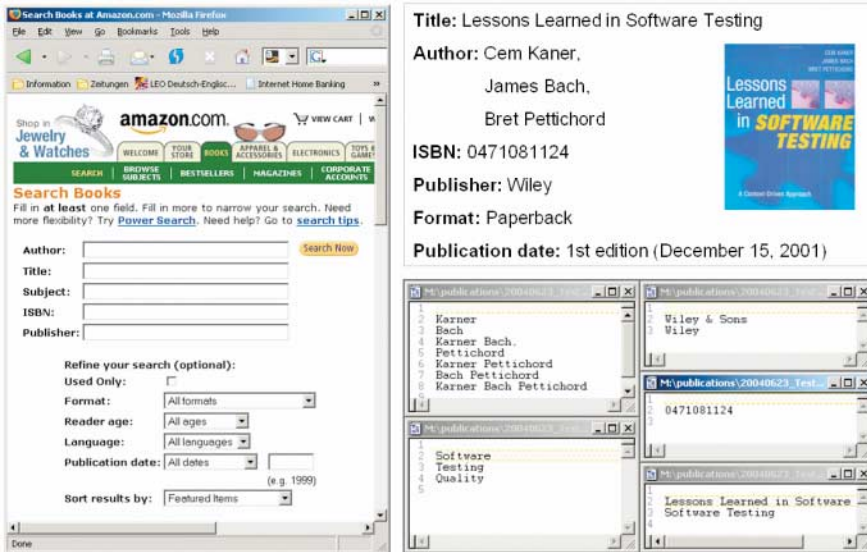
**TABELLE 2:
OHNE DOPPELTE PAARE**

Parameter X	Parameter Y	Parameter Z
1	A	I
2	B	I
2	A	II
1	B	II

**TABELLE 3: ERSPARNIS
DURCH PAARE**

Parameter	alle Kombinationen	pairwise-optimierte Kombinationen
3	27	10
6	729	15
9	19 683	17
12	531 441	21

jeder Parameter kann drei Werte annehmen.



Amazons Suchmaske, rechts unten alle möglichen Werte für die Eingabefelder, rechts oben das gesuchte Buch (Abb. 3).

Es fällt auf, dass alle Paarkombinationen mehrfach vorkommen, sodass sie mehrfach getestet werden. Entfernt man die Kombinationen bereits bekannter Paare, reduziert sich der Testaufwand, und man erhält die vier Kombinationen in Tabelle 2. Die Anzahl der Testfälle hat sich halbiert, ohne die Abdeckung zu verringern. Einen Algorithmus zur Umsetzung der Pairwise-Methode enthält unter anderem das Programm Allpairs von James Bach (www.satisfice.com).

Mit zunehmender Anzahl von Parametern oder Eingabewerten verbessert sich das Verhältnis zwischen allen und den pairwise-optimierten Testausprägungen. Besonders bei vielen Parametern reduziert die Pairwise-Methode die Anzahl der Testausprägungen deutlich (Tabelle 3).

Im Folgenden geht es um das erwähnte Versandhausszenario. Abbildung 3 zeigt beispielhaft eine Suchmaske des Amazon-Portals, ein zu suchendes Buch und die möglichen Suchparameter. Die Maske gliedert sich in die zwei Bereiche „Suche“ und „Verfeinerung der Suche“. In Ersterem können Interessenten Informationen über das gewünschte Buch eingeben. Die Suche sollte es mit jedem einzelnen Wert oder mit allen möglichen Kombinationen aus den Eingabefeldern und deren Werten liefern. Insgesamt ergeben sich 575 zu verifizierende Kombinationen: Acht Werte für den Autor, drei für den Verlag, zwei ISBN-Nummern, drei Titelvarianten und vier Schlagwörter. Das Produkt

aus diesen minus 1 (alle Felder leer) ist 575.

Weniger Arbeit durch Pärchen

Dasselbe Verfahren kann man auf die Verfeinerung der Suche anwenden. Für die Prüfung einer positiven Suche können 128 Kombinationen gebildet werden. Will man die Suche und ihre Verfeinerung vollständig testen, ergeben sich $575 \times 128 = 73\,600$ Kombinationen – spätestens jetzt kommt jeder Tester ins Schwitzen.

Im ersten Schritt lassen sich die Testkombinationen mittels der Pairwise-Methode verringern. Dadurch bleiben nur 69 von ursprünglich 73 600 übrig. Dies verringert die Zeit für die manuelle Testdurchführung von $90\text{ s} \times 73\,600 = 76,6$ Tage auf $90\text{ s} \times 69 = 1,7$ Stunden (wenn einmal Testen 90 s dauert). Soll der Test mehrmals laufen, (beispielsweise zur Kontrolle von Patches oder Hotfixes) ist seine Automatisierung sinnvoll. Dafür wäre ein Skript zu erstellen, das die Eingabemaske mit den nötigen Daten füllt und die Ergebnismenge auf das gesuchte Buch kontrolliert. Der Aufwand für die Testautomation mit einem Werkzeug wie Mercurys Winrunner beträgt circa zwei Arbeitsstunden. Die unbeliebte Dokumentation der Testdurchführung schrumpft auf ein Minimum, da das Werkzeug die erforderlichen Informationen zum größten Teil in Form von Berichten zur Verfügung stellt. Wenn die Ausführung eines automatisierten

Testfalls drei Sekunden benötigt, ergeben sich deutlich geringere Durchführungszeiten von $3\text{ s} \times 73\,600 = 2,6$ Tage für den Test aller Ausprägungen und $3\text{ s} \times 69 = 3,5$ Minuten für die pairwise-reduzierten Ausprägungen. Da das manuelle Testen nur wenig schneller erfolgt, rentiert sich die Erstellung des Skripts in diesem Fall bereits ab dem zweiten Durchlauf.

Nach dem Beschriebenen ergeben sich folgende Anforderungen an ein Werkzeug für die Generierung von Testdaten:

- Erzeugung realistischer synthetischer Testdaten mittels Wissensdatenbanken;
 - einfache Integration neuer Wissensdatenbanken;
 - Darstellung gängiger Datentypen (Datum, Integer, Long, Boolean et cetera);
 - Erweiterbarkeit der Datentypen mittels einer Plug-in-Schnittstelle;
 - plattformunabhängige Ausführbarkeit des Generators;
 - offenes und einfaches Format für die Definition der zu generierenden Testdaten;
 - performante Generierung von Massendaten sowie
 - Unterstützung bei der Ermittlung von Testausprägungen mittels Kombinatorik von Eingabeparametern und Pairwise-Optimierung.
- Testdatengenerierung beschleunigt die Vorbereitung und Durchführung von Funktions- sowie Performancetests. Der Einsatz von Kombinatorik und Pairwise-Optimierung beim funktionalen Testen erhöht die Abdeckung bei gleichzeitiger Senkung des Aufwands für die Durchführung. Automatisierung kann die Zeit für die Durchführung nochmals drastisch reduzieren. Für Leistungskontrollen und Produktbewertungen lassen sich Massendaten mit komplexer Struktur generieren, die den Aufbau realer Datenvolumina und Datenkonstellationen ermöglichen. (ck)

MATTHIAS SCHOLZE

arbeitet als Testmanager in der Abteilung Solution Quality der SAP SI AG in Berlin.

Literatur

- [1] David M. Cohen, Siddhartha R. Dalal, Jesse Parelius und Gardner C. Patton; The Combinatorial Design Approach to Automatic Test Generation; IEEE Software September 1996; pp. 83–87