Mückenspray, Fliegenklappe oder Holzhammer? Testmethoden im DWH

Dr. Andrea Kennel
InfoPunkt Kennel GmbH
CH-8600 Dübendorf

Schlüsselworte
DWH Projekt, Testmethoden, Code Review, White Box-Test, Blackbox-Test, Regression Test, Qualitätsprüfung, Plausibilisierung.

Einleitung

Auf die Frage, was ich denn beruflich so mache, antworte ich oft: Daten schaufeln. Die Hauptanforderung an ein DWH ist es Daten aus Quellsystemen so in eine separate Datenbank zu laden, dass diese Daten einfach und schnell ausgewertet werden können. Der Zentrale Punkt sind daher Daten. Die programmierente Abläufe sind dazu da, die Daten umzuwandeln und neu zu strukturieren. Daher eigenen sich zum Testen eines DWH neben herkömmlichen Methoden auch weitere Tests, die sich direkt auf die Daten beziehen.

Ziel des Testens oder welche Ungeziefer sind wirklich Bugs?

Ziel des Testens ist es, Fehler oder eben Bugs zu finden. Das ist ähnlich wie Unkraut jäten im Garten: Wenn ich nicht weiss, welche Pflanzen Kräuter sind und welche Unkraut, so wird die Aufgabe schwierig. Auch bei Software und von allem bei einem DWH muss zuerst bekannt sein, was überhaupt als Fehler anzusehen ist. Alle möglichen Fehler (denkbare und besonders undenkbare) Fehler zu definieren ist schlicht unmöglich. So ist es einfacher festzulegen, was keine Fehler sind. Alles, was so läuft wie es spezifiziert wurde, ist korrekt, was anders läuft ist falsch. Somit sehen wir, dass die Definition von Anforderungen wesentlich ist. Dabei ist wichtig, dass die Anforderungen eindeutig, unmissverständlich und nach Möglichkeit messbar sind. Wie beim Jäten hilft es, wenn die Person, die entwickelt und testet, auch Einiges an Fachkenntnissen mitbringt.
In einem DWH werden normalerweise Daten aus der Quelle für Auswertungen im DWH gesammelt. Bei vielen Anforderungen geht es darum, dass die Daten der Quellen vollständig in das DWH geladen werden. Das klingt nach eindeutigen, unmissverständlichen und messbaren Anforderungen. Oft werden aber Datenbereinigungen, Umformungen und Verdichtungen vorgenommen und Daten aus mehreren Quellen zusammengeführt, so dass nicht nur die Funktionalität sondern auch die Tests komplex werden.

Von der Anforderung zum Design

Die Anforderungen werden normalerweise von Fachabteilungen erstellt, die zum Schluss die Daten des DWH nutzen. Basierend auf den Anforderungen erstellt die Informatik das Design. Somit ist das Design eine technische Übersetzung der fachlichen Anforderung. Oder anders gesehen beschreiben die Anforderungen das WAS und das Design das WIE. Wie in jedem Projekt sind Fehler, die sich beim Design einschleichen, die teuersten. Wenn eine Anforderung falsch verstanden wird, dann wird diese eben falsch ungesetzt und muss zum Schluss komplett umgebaut werden. Daher ist die Überprüfung des Designs wichtig. Wie aber kann ein Design getestet werden und durch wen?
Das Problem ist, dass noch kein Code besteht, den man ausführen und testen kann. Daher werden oft statische Testverfahren wie ein Walkthrough oder Review eingesetzt. Am effizientesten können Fehler oder Missverständnisse erkannt werden, wenn die Fachabteilung und die Informatik sich zusammen setzten und gemeinsam prüfen, ob das Design wirklich alle Anforderungen abdecken.

Eine weitere Möglichkeit ist das Prototyping. Basierend auf den Quelldaten werden Beispiele erstellt. Dies eignet sich vor allem, wenn die Anforderungen in Form von Report-Beschreibungen vorliegen. Ein Prototyping ist aber nur möglich, wenn die Daten in den Quellsystemen bereits vorhanden sind.

Von der Schnittstelle ins Core

Im Idealfall sind die Daten, die in das DWH geladen werden sollen, in der Quelle bereits vorhanden. Dann können diese analysiert werden. Dies wird am einfachsten mit einem Data-Profiling-Tool gemacht. So können die Strukturen im Core so definiert werden, dass sicher alle Attribute korrekt und vollständig geladen werden können. Weiter können Konsistenzregeln frühzeitig geprüft werden und wo nötig Datenbereinigungen definiert werden. Denn Daten sorgen immer wieder für Überraschungen, weil es Ausnahmen gibt, an die niemand dachte oder die nicht bekannt sind.
Schwieriger wird es, wenn das DWH parallel zu den Quellen entwickelt wird. Dann sind noch keine Daten vorhanden und oft sind auch die Schnittstellendefinitionen nicht stabil. So müssen häufig Annahmen getroffen werden. Dabei ist es sinnvoll, wenn diese Annahmen protokolliert werden. Damit können diese einfach überprüft werden können, wenn die ersten Daten kommen.
Muss ohne Daten implementiert werden, fragt es sich, wie trotzdem getestet werden kann. Eine Möglichkeit ist das Arbeiten mit Testdaten, die generiert respektive manuell erstellt werden. Dabei liegt die Gefahr darin, dass die Daten "passend" zu den Anforderungen erstellt werden und so unvollständige Anforderungen nicht erkannt werden. Je nach Komplexität wird es auch schwierig und vor allem aufwändig, sinnvolle Testdaten zu erfassen.
Für weitere Testmöglichkeiten lohnt es sich zu überlegen, wie sich Bugs einschleichen. Die hauptsächlichen Gefahren bestehen darin, dass nicht alles implementiert wird, was spezifiziert ist. Auch Flüchtigkeitsfehler sind eine Fehlerquelle. Beispielsweise wird ein Join nicht vollständig formuliert oder es wird ein falsches Attribut selektiert und für das Mapping verwendet. Solche Fehler können mit dem „Vieraugenprinzip“ erkannt werden. Das bedeutet, dass eine zweite Person den fertigen Code reviewed oder zusammen mit dem Entwickler, der Entwicklerin diesen mit einem Walkthrough überprüft. Wird mit einem Tool gearbeitet, können auch automatische Prüfungen vorgenommen werden.
Ein meist kritischer Punkt im DWH ist die Laufzeit. Diese kann aber klar erst getestet werden, wenn eine repräsentative Menge an Daten aus den Quellen verarbeitet werden kann. Doch auch hier können potentielle Engpässe bei einem Review oder Walkthrough erkannt werden.

Vom Core in den Report

Wie beim Laden der Daten in den Core ist das Erstellen von Reports mit vorhandenen Daten einfacher als ohne Daten. Die Struktur der Daten im Core ist bekannt und muss nicht mehr analysiert werden, das vereinfacht das Erstellen von Reports. Dafür müssen aber normalerweise mehrere Tabellen und auch versionierte Tabellen zusammengeführt werden. Hier besteht die Gefahr, dass durch Verknüpfungen von Tabellen und Einschränkungen auf die Gültigkeit Datensätze verloren gehen oder doppelt selektiert werden. Diese Fehler sind mit repräsentativen Testdaten frühzeitig erkennbar. Wenn blind, also ohne Daten implementiert werden muss, gibt es auch hier die Möglichkeit, Fehler durch Reviews oder Walkthroughs zu erkennen. Reviews und Walkthroughs können allerdings nie Tests mit realen Daten ersetzten. Beim Testen mit Daten ist wichtig, dass ein besonderer Augenmerk auf kritische Situationen wie zum Beispiel Zeitübergänge gelegt wird. Wenn also mehrere Versionen einer Dimension Ende Monat ändern, so sind Tests am Ende des Monats und am ersten Tag des Folgemonats wichtig. Hier ist die Gefahr am grössten, dass Daten nicht oder doppelt gelesen werden.

Von der Anforderung zum Report
Bisher wurden vor allem Test während der Entwicklung angesprochen. Dies sind sogenannte Whitebox-Tests und prüfen, ob die Umsetzung dem Design entspricht und die Daten so geladen werden, wie es spezifiziert wurde.
Es fragt sich aber, ob auch das implementiert wurde, was ursprünglich gewünscht war und ob die Daten, die in Reports ausgewiesen werden, für das Fach auch wirklich Sinn machen?
Auf dieser Stufe interessiert der Code nicht mehr direkt, daher wird mit sogenannten „Blackbox-Tests“ gearbeitet. Wie die Daten genau geladen werden, interessiert nicht. Es interessiert „nur“, ob basierend auf der Eingabe, die erwartete Ausgabe produziert wird.
Diese Test sollten von einem Testteam der Fachabteilung erstellt und durchgeführt werden. Entwicklerinnen und Entwickler neigen dazu, das zu testen, was sie umgesetzt haben. Das bedeutet, sie testen nur Spezialfälle, an die sie schon beim Entwickeln gedacht haben. Ein Testteam geht eher mit dem Wissen vom Fach an die Tests und prüft somit mit anderen Daten. Verglichen mit einer Fliegenklappe ist das Maschennetz des Testteams anders und oft enger, so dass mehr Fliegen damit erwischt werden.

Nebeneffekte oder Bugs beseitigen, bevor sie geschlüpft sind
Je grösser und komplexer ein DWH wird, desto grösser ist auch die Gefahr, dass beim Entwickeln neuer Teile bestehende Funktionen verändert werden. Daher wäre es gefährlich, nur das zu prüfen, was geändert wurde. Das bedeutet aber, dass bei jedem Release möglichst die gesamte bisherige Funktionalität getestet wird. Dass dies manuell rein vom Aufwand her nicht machbar ist, leuchtet ein. Daher braucht es vordefinierte Regressiontests. Dies ist bei einem DWH zwar auch mit einem Aufwand verbunden, aber immerhin nicht zu komplex.
Vereinfacht gesehen ist es wichtig, dass die bestehenden Reports auch nach Einführung eines neuen Releases weiterhin dieselben Daten liefern. Da aber die Daten vor dem Release mit Daten nach dem Release verglichen werden müssen, kann es trotzdem zu Abweichungen kommen, die fachlich begründet sind. Daher werden bei Regressiontests oft Grenzwerte definiert, ab welchem Abweichungen als Fehler interpretiert werden.
Idealerweise werden beim Entwickeln neuer Teile gleich die Regressiontest mit entwickelt.

Qualitätsprüfung als Moskitonetz
Es ist nie möglich, bei den Tests alle Datenkonstellationen zu prüfen und testen. Es besteht bei jedem Ladevorgang die Gefahr, dass Daten falsch oder nicht geladen werden. Daher lohnt es sich, in die täglichen, wöchentlichen oder monatlichen Ladevorgänge auch Prüfungen einzubauen.Eine erste Prüfung ist das Zählen der Datensätze von der Quelle. Hier kann auch mit Grenzwerten festgelegt werden, welche Abweichungen zum letzten Load tolerierbar sind oder wie viele Datensätze minimal oder maximal erwartet werden. Bei der Weiterverarbeitung der Daten kann bei jedem Mapping eine solche Prüfung eingebaut werden. Dabei kann die Datenmenge entweder mit den vorhergehenden Ladeprozess verglichen werden oder mit der Anzahl Datensätze der Quelle. So können die geladenen Daten plausibilisiert werden.
Dabei ist aber zu beachten, dass die Selects zum Zählen von Datensätzen nicht zu komplex sein sollten, sonst benötigen sie zu viel Zeit und der Ladevorgang wird unnötig verlangsamt. Es gilt, die Qualitätsprüfung so zu gestalten, dass kritische Fehler rasch erkannt werden, dafür aber nicht zu viel Laufzeit verbraucht wird.

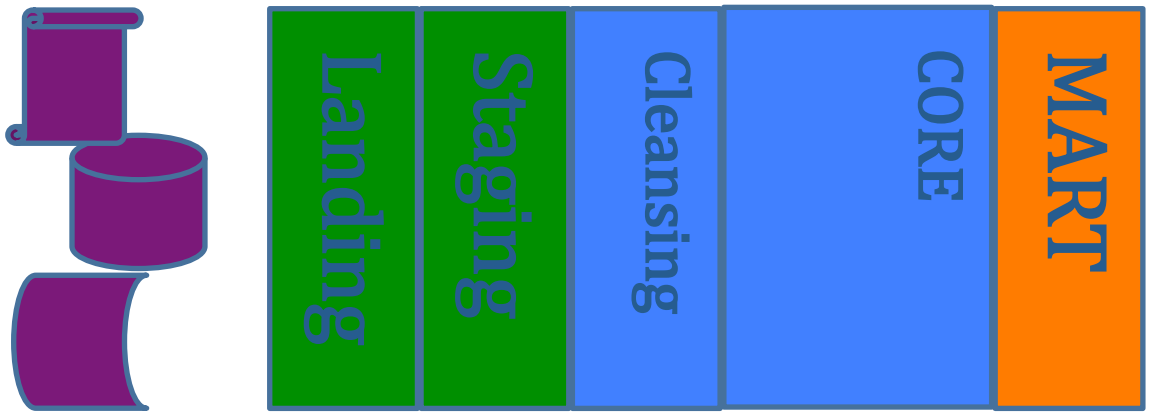

Abb. 1: DWH-Architektur

Wenn wir die Architektur eines DWH betrachten, so wird zuerst die Staging und dann der Core geladen. Die Marts und Reports bauen dann auf den Daten des Core auf. Wenn Daten falsch in die Staging kommen, dann werden sie auch bis in die Marts falsch geladen und eine nachträgliche Korrektur wird aufwändig. Treten Fehler erst zwischen Core und Mart auf, so sind diese meist einfacher zu korrigieren, denn die Marts können normalerweise basierend auf dem Core neu aufgebaut werden.
Somit sind Qualitätsprüfungen vor allem beim Lesen der Quelldaten und beim Schreiben ins Core wichtig. Das Moskitonetz beim offenen Fenstern und bei der Zimmertüre ist somit effektiver als jenes über dem Bett.

Quell- und Ladeinfo – damit der Holzhammer trifft
Und was, wenn doch etwas schief geht?
Dazu eine kleine Geschichte aus der Praxis. Die Daten der Quellen werden täglich in das DWH geladen. Der Betrieb überwacht den Prozess und greift falls nötig ein. Nun meldet das System, dass von einer Quelle Daten unvollständig gekommen sind. Es ist Samstag und das DWH ist 3 Tage verspätet mit dem Laden von Daten. Die zuständige Person geht davon aus, dass die fehlenden Daten auch später nachgeladen werden können. Nach Rücksprache mit dem Chef, der möglichst rasch den zeitlichen Rückstand aufgeholt haben möchte, gibt es grünes Licht.
Der Rest des täglichen Ladeprozesses läuft tatsächlich ohne Probleme durch und der nächste Ladeprozess wird gestartet. Dieser meldet nun bei den Marts Probleme, weil eben doch Daten in mehreren versionierten Tabellen fehlen. Der Betrieb steht an und muss bis Montag warten. Nach genauen Analysen durch Entwickler müssen die geladenen Daten nachträglich doch wieder gelöscht werden, um die Dimensionen in der richtigen Reihenfolge zu laden.
Dadurch, dass in jeder Tabelle für jeden Datensatz festgehalten wird, mit welchem Load dieser geladen wurde und falls möglich aus welcher Quelle, können einzelne Loads auch wieder rückgängig gemacht werden. Wenn also alle anderen Methoden zur Qualitätssicherung nicht gegriffen haben, besteht dank Loadinformationen die Möglichkeit, falsch geladene Daten zu identifizieren und zu löschen oder zu korrigieren.
Diese Methode ersetzt aber kein Backup. Denn wenn wirklich alle Stricke reissen, muss auf ein vollständiges Backup zurück gegriffen werden können. Das bedingt aber, dass auch der Widerherstellungsprozess (recovery) regelmässig getestet wird und funktioniert.

Fazit
Beim Entwickeln eines DWH müssen, wie in jedem Projekt, Test geplant werden. Dabei stehen die Daten im Zentrum. Falls aber die Daten während des Entwickelns nicht zur Verfügung stehen, müssen andere Methoden wie Code Review genutzt werden, um möglichst Fehler zu vermeiden. Bevor aber ein DWH zum Einsatz kommt, müssen die Prozesse auch mit repräsentativen Datenmengen geprüft werden.
Bei jedem weitern Release in einem DWH Projekt ist es wichtig, dass auch die bisherige Funktionalität weiterhin korrekte Daten ermittelt. Dazu eignen sich automatisierte Regression-Tests basierend auf bestehenden und neuen Reports.
Für den Betrieb und das Sicherstellen, dass auch neue Datenkonstellationen korrekt geladen werden können Qualitätsprüfungen eingebaut werden, die die Menge der geladenen Daten prüfen. Dabei muss aber darauf geachtet werden, dass diese Prüfungen einfach und somit schnell sind.

Kontaktadresse:
Dr. Andrea Kennel
InfoPunkt Kennel GmbH
Am Wasser 4
CH-8304 Wallisellen

Telefon: +41 (0) 44-820 71 40
E-Mail andrea@ infokennel.ch
Internet: www.infokennel.ch