Unit-Testing in Informatica PowerCenter

Klassisch gesehen ist das Unit-Testing in der IT ein Thema, welches in der Softwareentwicklung beheimatet ist. Aber zum einfacheren Verständnis auch vergleichbar mit dem Vorgehen aus der Praxis allgemeiner Fertigungsprozesse.

Betrachtet man bspw. die Fertigung eines Autos, so wird hier nicht lediglich eine Teileliste erstellt, alles zusammengebaut und nach Vollendung geprüft, ob es nun wirklich ein funktionsfähiges Auto geworden ist. Bei der Fertigung eines Autos ist es wahrscheinlich für jeden relativ plastisch und klar, dass es viele Bauteile gibt, deren Funktionen einzeln getestet werden können und werden. Die benötigten Eigenschaften und Parameter eines jeden Teils werden definiert und können somit überprüft werden, weit bevor sie ihren Weg in ein Fahrzeug finden.

Die Softwareentwicklung kann hier ganz ähnlich betrachtet werden. Es gibt eine Software, die zu erstellen ist und bestimmte Eigenschaften zu erfüllen hat. Die finalen Eigenschaften werden im Allgemeinen heruntergebrochen, der Gesamtkomplex der Software wird auf viele kleine „Bauteile“ oder Units aufgeteilt. Jede dieser Units kann über bestimmte Eigenschaften beschrieben werden, d.h. jede Unit hat eine gewisse Signatur, bestehend aus Parametern, die von außen hineingehen, und Ergebnissen, die am Ende wieder herauskommen.

Was ist das Ziel beim Unit-Testing?

Ansatz und Ziel beim Unit-Testing ist es, direkt bei der Implementierung der einzelnen Units die Expertise aus Design und Entwicklung zu nutzen und Testfälle daraus abzuleiten. Die Idee dahinter ist, dass die Beteiligten aus dem Design genau wissen, welche Anforderungen die einzelne Unit zu erfüllen hat. Sie betrachten die Unit jedoch als eine Art Blackbox. Der Entwickler hingegen kennt den Code und kann somit den Test ggf. noch ein wenig ausfeilen, da er jede einzelne Verzweigung, die sich in der Implementierung ergibt, genau kennt.
Das Ziel der definierten Testfälle ist, dass möglichst sämtliche „Wege durch den Code“ durch einen Testfall abgedeckt sind. Eine 100%ige Abdeckung ist nicht immer zu erreichen, da dies mitunter auch davon abhängig ist, wie feingranular die einzelnen zu testenden Units gefasst werden.

Nicht selten kann es beim Unit-Testing vorkommen, dass – je nach verwendetem Framework – das Schreiben der Tests schnell die Zeit und den Aufwand erreicht, wie es das Schreiben des eigentlichen Codes benötigt.

Wie werden Unit-Tests umgesetzt?

Unit-Tests werden sinnvollerweise von den Entwicklern parallel mitentwickelt, wobei die (groben) Testfälle zum Teil ggf. bereits im Design definiert werden können. Die Tests können auf unterschiedlichster Granularitätsstufe umgesetzt werden. Je tiefer man in den Code hineingeht, umso höher ist die Abdeckung, die realistisch mit den Testfällen erreicht werden kann. Je größer die Flughöhe gewählt wird, desto schwieriger ist es, sämtliche Varianten abzudecken. Bei späteren Detailänderungen ist es dann ggf. schwerer zu erkennen, ob die neuen Gegebenheiten bereits durch die Tests abgedeckt sind oder nicht.

Für die klassische Softwareentwicklung gibt es entsprechende Frameworks, die es ermöglichen, dass der Code bereits parallel zur Implementierung schnell und einfach die zugehörigen Tests durchlaufen kann. Spätestens aber beim Einchecken der Artefakte in ein Repository oder in nächtlichen automatisierten Läufen, wo der Code anhand der Tests überprüft wird und entsprechende Protokolle generiert werden.

Wie kann Unit-Testing in PowerCenter angewandt werden?

Da die Entwicklung mit Informatica PowerCenter in den eigens dafür vorgesehenen Clients wie dem Designer und dem Workflow Manager abläuft, entzieht sich die Entwicklung hier zunächst ein wenig der Welt der klassischen Softwareentwicklung. Die IDE (Integrated Development Environment) lässt es nicht zu, hier allgemeine oder eigene Frameworks direkt einzubinden. Die Implementierung erfolgt so nicht allzu selten mit dem Vorhandenen; das Testen wird als komplett eigenständige Phase der Entwicklung hintenan gestellt.

Oftmals gibt es Datenbanken mit Inhalten, die aus den produktiven Systemen abgeleitet sind und von allen Entwicklern, Testern etc. gleichzeitig genutzt werden. Es werden bestimmte Konstellationen in den Daten ermittelt, die Prozesse durchlaufen und anschließend wird geprüft, ob sich das Ergebnis mit der Erwartung/ Anforderung deckt. Ob durch die Tests sämtliche Varianten und Möglichkeiten abgedeckt sind, lässt sich dabei oft schwer überblicken und wird teilweise grob heuristisch angenommen, wenn man eine entsprechend große Datenmenge verarbeiten lässt.
Ob der vermeintlich gleiche Test einen Tag oder eine Woche später das gleiche Resultat bringt, ist ebenfalls oft schwer abzuschätzen, da sich die Daten in der Zwischenzeit durch andere Prozesse und Tests bereits wieder verändert haben können.

Um nun mit PowerCenter zu aussagekräftigen Unit-Tests zu gelangen, bedarf es einiger Ansatzpunkte. Dazu seien hier einige Ideen genannt, die bei Unit-Tests aber auch allgemein weiterhelfen können.

1. Die Prozesse sind idealerweise in feingranulare logische Einheiten zu unterteilen.
2. Die Verwendung von reusable Objekten ist hilfreich. Wo es Sinn macht, sollten reusable Transformationen eingesetzt werden oder aber sinnvolle Komponenten Mapplets zusammengefasst werden.
3. Es kann sehr hilfreich sein, wenn jedem Entwickler/Tester eine eigene „Datenbank-Sandbox“ zur Verfügung steht, in der die Tests ausgeführt werden können.
4. Um den Aufwand für Tests so gering wie möglich zu halten, sollte hierfür ein entsprechendes Framework in Form von Scripten o.ä. zur Verfügung stehen.

Ein beispielhaftes Konzept und Erfahrungen

Bei einem unserer Kunden wurde ein Unit-Testing wie folgt aufgebaut und eingeführt:

1. Nahezu alle Komponenten werden als reusable Objekte in shared Foldern angelegt, z.B. TOPIC_001_shared.
2. Zu jedem dieser shared Folder gibt es zwei weitere Folder. Einen für die „echten“ Prozesse und einen für Tests. (z.B. TOPIC_001_exec und TOPIC_001_test)
    a. In TOPIC_001_exec werden die realen Prozesse erstellt, welche das geforderte Szenario abbilden.
    b. In TOPIC_001_test werden die Unit-Tests erstellt, welche je nach Bedarf aus 1 bis m standardisierten Sources bestehen und am Ende 1 bis n Mal ein standardisiertes Mapplet zum Schreiben des/ der Targets beinhalten.
3. Es gibt eine Datenbank für die Unit-Tests, in der automatisiert für jeden testenden User Schemas, Tabellen etc. angelegt werden können.
4. Für die Ausführung der Unit-Tests wurde ein kleines Framework (mit Java) implementiert, welches die notwendigen Schritte kapselt und somit das Testen erleichtert.
5. Der Setup für jeden Test besteht aus fünf Dateien:
    a. Data Definition Language (DDL)-Statements für die benötigte Tabellen
    b. DATEN zur Vorbefüllung der Tabellen mit Basisdaten
    c. expRESULT mit den erwarteten Ergebnissen
    d. ETL-Parameter für die Ausführung des PowerCenter Workflows

e. Properties mit den notwendigen Parametern für das Framework

Die Erfahrungen mit dem eingeführten Unit-Testing sind insgesamt sehr gut und auch die Qualität der Entwicklungen konnte merklich gesteigert werden. Entsprechend wurde auch das Vertrauen vom Fachbereich in die Prozesse gefördert.

Für die Entwickler waren anfangs ein paar Iterationen und Erkenntnisse notwendig, um einen allgemein als praktikabel anerkannten Weg für den Prozess zu finden. Bei Themengebieten, wo wirkliche Neuentwicklungen anstanden, war die Umsetzung im Vergleich deutlich einfacher, als dort wo die Mappings etc. schon deutlich älter waren und zumeist oft von Entwicklern implementiert wurden, die nicht mehr in dem Projekt arbeiteten.

Aber auch unter den Entwicklern ist die Erkenntnis gewachsen, dass es mit den Unit-Tests später deutlich einfacher und schneller geht, nach Änderungen in den Implementierungen diese zu testen.