PL/SQL Continuous Integration mittels Hudson

Benjamin Jörger
Essential Bytes GmbH & Co.KG
77749 Hohberg

Schlüsselworte: PL/SQL, Continuous Integration, utPLSQL, PLUTO, Subversion, Hudson

Einleitung

Continuous Integration (CI) beschreibt eine Methodik aus der Softwareentwicklung. Beim Einsatz dieser Methode integriert jeder Mitarbeiter eines Teams mindestens einmal pro Tag seine Fortschritte/Eigenentwicklungen in das Gesamtsystem (Test-/Produktivsystem). Die kontinuierliche Veränderung des Systems von mehreren Personen birgt die Gefahr von Seiteneffekten, die von den Entwicklern nicht überblickt werden. Um diese Fehler bzw. inkonsistente Zustände schnellstmöglich zu entdecken, werden mit einem CI-Server automatische Tests durchgeführt. Die Tests können zum einen aus komplexen Build Prozessen bestehen oder auch aus fragmentalen Funktionsaufrufen. Auch Abhängigkeiten der Prozesse/Funktionen lassen sich durch Testszenarien abbilden.
Der grundsätzliche Aufbau eines CI-Systems

1. Benötigte Infrastruktur
   - CI-Server
   - Versionskontrollsystem (SVN)

2. Entwicklung von Testszenarien
   - Gliederung des Systems in Teilbereiche
   - Definition von Anhängigkeiten
   - Integration der Testfälle

Das CI-System lebt von einer hohen Testabdeckung, d.h. es sollte nach Möglichkeit für jede Prozedur mindestens ein Testfall vorhanden sein. Jedem Teammitglied muss bewusst sein, dass die Entwicklung von Tests mindestens genauso wichtig ist wie die eigentliche Implementierung der Funktionalität. Genau dies stellt im täglichen Betrieb jedoch ein großes Problem dar, da Funktionalitäten möglichst schnell umgesetzt werden sollen. Dieser Herausforderung kann durch die Einführung von Test-Driven-Development (TDD) begegnet werden. TDD schreibt vor, dass vor der Entwicklung der Funktion zunächst der Test entwickelt wird und erst im Anschluss mit der Umsetzung der begonnen wird. Letztendlich muss von der Projektleitung genügend Zeit für die Entwicklung von Tests bereitgestellt werden.
Bis zu diesem Zeitpunkt klingt die Einführung von CI nur nach einem erheblichen Aufwand. Bei der Betrachtung der Vorteile ist schnell zu erkennen, dass die Kosten im Vergleich zu dem Aufwand, der bei einer späten Fehlererkennung entsteht, verhältnismäßig gering sind. Die Kosten zur Beseitigung eines Fehlers verhalten sich äquivalent zu dem Kostenmodell in der Softwareentwicklung. Je früher ein Fehler entdeckt wird, desto geringer sind die Kosten.

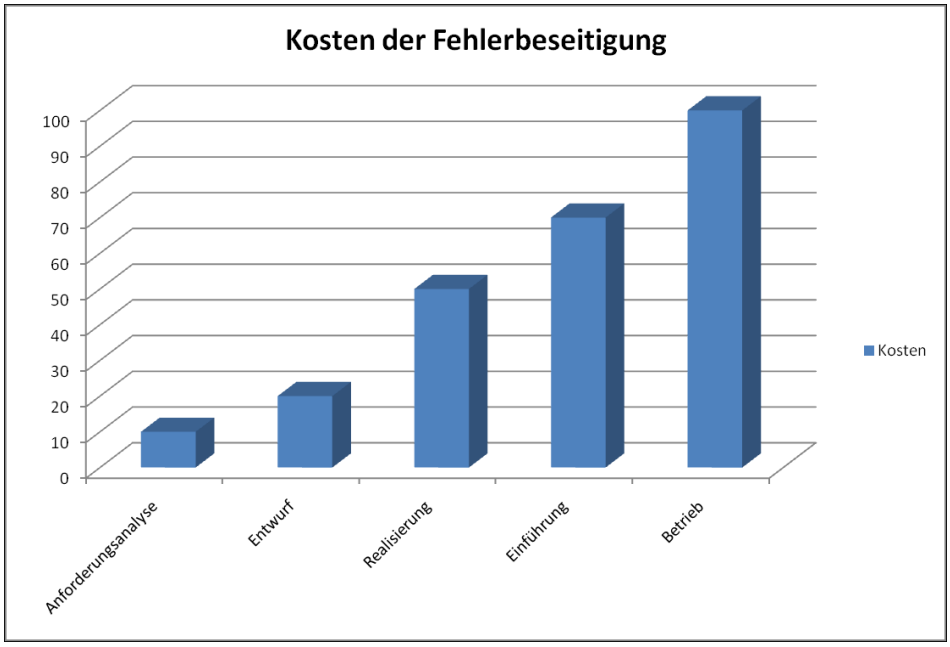


Abb. 1 Kostenübersicht zur Fehlerbeseitigung beim Wasserfallmodell

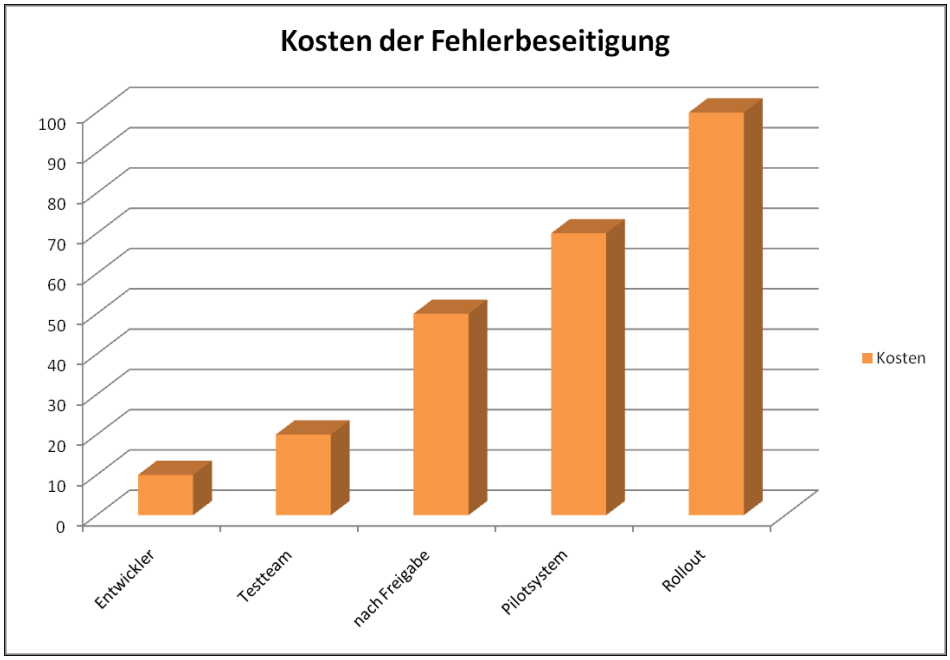
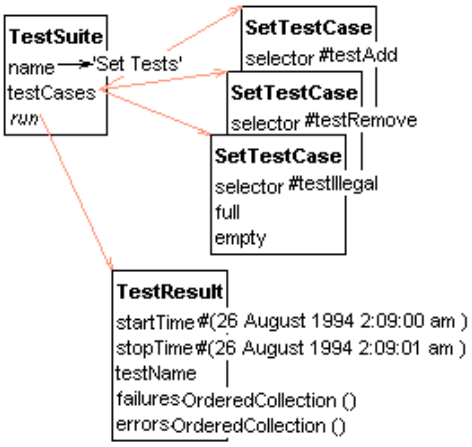


Abb. 2 Kostenübersicht zur Fehlerbeseitigung nach der Entdeckungsphase

Continuous Integration bietet die Möglichkeit einen Fehler **immer** zu einem sehr frühen Zeitpunkt zu entdecken. Die Kosten für das gesamte System können somit durch einen einzigen Fehler, der es bis ins Produktivsystem nicht entdeckt wurde, gedeckt werden!

1. PL/SQL Testframeworks

Für die Implementierung von Testfällen (Testcases) in PL/SQL stehen mehrere Frameworks zur Verfügung auf deren Features in diesem Abschnitt kurz eingegangen wird. Es werden jeweils die Vorgehensweisen zur Testimplementierung erörtert und abschließend die Vor- und Nachteile der Frameworks verglichen.
XUnit ist eine Sammlung von Testframeworks die gemeinsame Eigenschaften aufweisen. Alle XUnit basierenden Frameworks erlauben es einzelne Testcases zu einem Szenario zusammenzustellen, womit eine hohe Wiederverwendbarkeit erreicht wird. Bei der Auswahl des Frameworks zur Realisierung von CI mit PL/SQL sollte dieser Faktor berücksichtigt werden!



1.1 PL/UNIT

PL/Unit[1] besteht aus einem einzigen Package mit verschiedenen Funktionen zur Entwicklung von Tests. Die Testfälle selbst werden unter Verwendung des Packages „plunit“ als separates Package erstellt. Es wird empfohlen pro Applikation ein eigenes Testschema anzulegen, um eine klare Trennung zwischen Anwendungslogik und Testlogik zu erhalten. Über Annahmen (Assertions) werden die einzelnen PL/SQL Methoden geprüft. Die Assertions sind in 4 Bereiche gegliedert (Varchar2, Boolean, Date, Number).

```
plunit.assert_equals(
            0.10,
            calc_comm_percent(1000, 100),
            'Commission percent is wrong');
plunit.assert_null(
            find_invalid_values(),
            'Found some invalid values');
```

Im Fehlerfall wird über DBMS Output der entsprechende Fehler ausgegeben. Die einzelnen Testmethoden können in einer Test-Suite organisiert werden in der wiederum andere Test-Suites aufgerufen werden. Vorsicht! PL/UNIT bietet keinen Schutz vor zirkulären Aufrufen.

+ Einfach Installation
+ GUI Integration mit Apollo-Pro[2]
- Keine Statistiken

1.2 PLUTO

Im Gegensatz zu PL/UNIT verfolgt PLUTO[3] den objektorientierten Ansatz. D.h. um Tests zu implementieren werden Objekte von PLUTO abgeleitet. Ein weiterer Vorteil ist die optionale Phasensteuerung.

[1] www.plunit.com
[2] www.apollopro.com
[3] http://code.google.com/p/pluto-test-framework

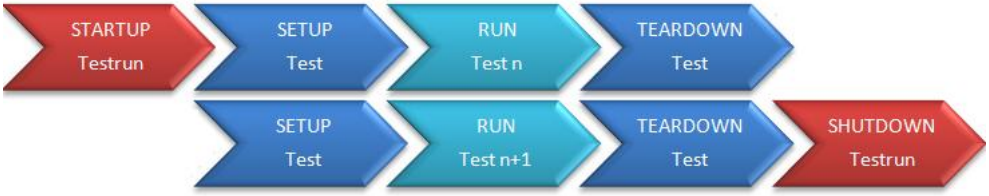


Abb. 3 Phasen eines Testlaufs

Für die einzelnen Tests gibt es jeweils eine Initialisierungs- und eine Shutdownphase, in welcher benötigte Parameter gesetzt bzw. wieder zurückgerollt werden können.
Das Klassenkonzept stellt Klassen zur Steuerung und zur Ausgabe der Testsuite zur Verfügung. Damit die einzelnen Methoden der eigenen Klasse aufgerufen werden, müssen diese bestimmten Konventionen folgen. Jede Methode die mit „test“ beginnt wird als eigenständiger Test aufgeführt. Sind mehrerer Testroutinen in einem Objekt definiert werden diese in der absteigenden Reihenfolge sortiert ausgeführt.
Die Methoden für die Phasen werden jeweils mit den Schlüsselworten „startup“, „setup“, „teardown“ und „shutdown“ identifiziert. Wichtig ist, dass Setup und Teardown vor **jedem** Test aufgerufen wird und nicht explizit für einen einzelnen Testfall definiert werden kann. Wie bei den Testfällen können jedoch mehrere Setup und Teardown Routinen deklariert und ausgeführt werden.
Zur Ausgabe der Meldungen stehen zwei Klassen bereit von denen wiederum eigene Parser abgeleitet werden können. Die Ausgabe erfolgt wieder per DBMS Output.

Abb. 4 Klassenkonzept von PLUTO

PLUTO_OBJ ist das Core Paket es beinhaltet die Methoden zur Ausführung der Testfälle.
PLUTO_UTIL_OBJ enthält die einzelnen Assertions welche durch IS_OK(<Params>) aufgerufen werden
PLUTO_OUTPUT_OBJ dient dem Logging und der Ausgabe der Testergebnisse

+ Einfache Installation
- Keine Statistiken

1.3 utPLSQL

utPLSQL[4] ist das umfassendste Framework, es besteht aus mehreren Packages und Tabellen. Es ist das einzige Framework das über Tabellen konfigurierbar ist, Dateiausgaben erlaubt und über einen eigenen Codegenerator verfügt. Prinzipiell werden Testpackages erstellt, die wiederum in Testsuiten verwendet bzw. zusammengestellt werden können. Die Phasen Setup und Teardown sind ebenfalls in die Testpackages integriert. Die Ergebnisse der einzelnen Testläufe werden in Tabellen gespeichert, so ist es möglich Langzeitanalysen zu fahren. Gespeichert wird der Status und die einzelne Teilergebnisse.

[4] http://utplsql.sourceforge.net

Package	Beschreibung
utPLSQL	Kontrollmechanismen für die Tracinginformationen, Starten von Tests bzw. Testsuiten
utConfig	Konfiguration der Testumgebung (User, Fileoutput, Dateformat, Delimiter, Recompile, u.v.m.)
utResult	Diverse Möglichkeiten die Ergebnisse des letzten Testlaufs auszuwerten (Iteration der Ergebnisse, Zugriff auf alle Ergebnisse, Fehler true/false, …)
utAssert	Umfangreiche Assertions nicht nur auf skalare Werte sondern auch für Tabellen, Objekte, Queries
utGen	Codegenerator für die einfache Erstellung von neuen Testsuiten
utOutput	Speicher für mehrere Testläufe(Ergebnisse) und Zugriffsmethoden
utSuite	Handling von Testsuites (CRUD)
utPackage	Handling von Testpackages einer Suite (CRUD)

Tab. 1 Packageübersicht von utPLSQL

2 Hudson

Hudson ist ein webbasierter Continuous Integration Server, der sich insbesondere durch hohe Stabilität, Skalierbarkeit, Benutzerfreundlichkeit und Flexibilität auszeichnet. Das Open Source Tool hat sich im Java Umfeld zum de facto Standard entwickelt. Durch diverse Plugins kann Hudson auch für andere Entwicklungssprachen eingesetzt werden. Je nach Anforderung gibt es Plugins für unterschiedliche Build-Verfahren (z.B. Maven, Ant), Test-Tools, Code-Quality-Checks und Benachrichtigungsmechanismen.
In der Standard-Konfiguration bringt Hudson einen eigenen Webserver (Winstone) mit, so dass sich Hudson ohne Vorarbeiten installieren lässt: Herunterladen – Starten - Fertig. Im Normalfall ist die Installation mit wenigen Klicks und in weniger als fünf Minuten ausgeführt. Die weitere Konfiguration von Hudson findet über das Webinterface statt.
Im Continuous-Integration Konzept (mit Hudson und PL/SQL) werden folgende Punkte berücksichtigt:

- Update der aktuellen PL/SQL Packages/Funktionen aus Subversion
- Ausführen der Testfälle und Protokollierung der Ergebnisse
- Archivieren des Builds und aller Testergebnisse

3 PL/SQL Continuous Integration mittels Hudson

3.1 Voraussetzungen

Da Hudson ein sehr offenes Produkt ist, müssen zunächst die Voraussetzungen geschaffen werden. Tabelle zeigt die entsprechenden Produkte und deren Quellen.

Produkt	Quelle
Java Development Kit (JDK)	http://www.oracle.com/technetwork/java/javase/downloads/index.html#need
Subversion	http://tortoisesvn.tigris.org/ http://subversion.apache.org/
Maven	http://maven.apache.org/
utPLSQL	http://utplsql.sourceforge.net/
Maven Plugin für utPLSQL	http://code.google.com/p/maven-utplsql-plugin/
Oracle JBDC Treiber für Maven	{$ORACLE_HOME}\jdbc
Hudson	http://hudson-ci.org/

Tab. 2 Produkt/Quellen

Subversion wird benötigt um den Code (PL/SQL-Packages) und deren Testroutinen zu verwalten. Bei dieser Methode übermittelt jeder Entwickler seine Änderungen am Code an das sogenannte Repository. Auf dieses Repository greift auch Hudson zu. Der Server lädt sich immer den aktuellsten Quellcode und prüft diesen auf Fehler mit Hilfe der Testfälle.

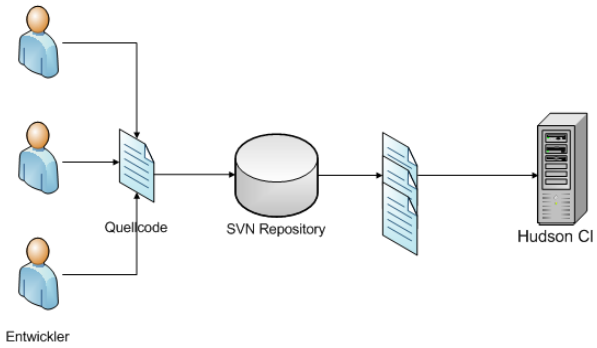


Abb. 5 Repository Zugriff

Maven ist ein Build-Tool mit dem (Java) Programme standardisiert erstellt und verwaltet werden können. Maven ist verantwortlich für den Aufruf der Testfälle und die Erstellung eines von Hudson interpretierbaren Ergebnisses.

3.2 Installation

3.2.1 Maven

Für die Installation von Maven müssen lediglich die Sourcen entpackt werden und eine Umgebungsvariable (`M2_HOME`) gesetzt werden. Voraussetzung ist eine korrekte Installation des JDK. Über den Kommandozeilenbefehl `mvn -v` kann die korrekte Funktionsweise getestet werden. Maven verfügt über ein lokales Repository in dem alle benötigten Bibliotheken lokal vorgehalten werden. Der Speicherort des Repositories kann in der „settings.xml“ angepasst werden.

3.2.2 Subversion

Die Installation von Subversion erfolgt per Installer / Paketmanager. Über beliebige Repositories werden die Quellen der Entwickler verwaltet. Der Administrator muss nur die entsprechenden Repositories anlegen.

```
svnadmin create <path_to_repository>
```

Ist das Repository angelegt, muss dieses noch als Daemon bereitgestellt werden

```
sc create svnserve svnadmin create <path_to_repository>
```

3.2.3 utPLSQL

Das Testframework wird direkt in das Schema integriert, in dem die entsprechenden Tests ausgeführt werden sollen.
Anlegen des Schemas und Vergabe der Rechte

```
create user utp identified by utp default tablespace users temporary
tablespace temp;

grant create session, create table, create procedure,
  create sequence, create view, create public synonym,
  drop public synonym to utp;

alter user utp quota unlimited on users;
```

Die eigentliche Installation erfolgt per SQL-Skript `@ut_i_do` wobei die Parameter install/uninstall die Steuerung des Skriptes vornehmen.

Das Maven Plugin für utPLSQL welches die Ausführung de Testfälle vornimmt, muss ebenfalls noch installiert werden.

```
mvn install:install-file -Dfile=/tmp/maven-utplsql-plugin-10-snapshot1.jar
-DpomFile=/tmp/pom1.xml
```

3.2.4 Hudson

Die Hudson Installation ist denkbar einfach, Download der Archivs und ausführen mit `java -jar hudson.war --httpPort=9081`. Und schon ist der CI-Server erreichbar.
Die restliche Konfiguration ist über das Webinterface vorzunehmen. Die minimale Konfiguration besteht aus der Definition der Pfade für das JDK und für Maven.
Wie einzelne Testfälle in Hudson integriert werden zeigt der Abschnitt 4.3

3.2.5 Zusammenfassung

Durch die verschiedenen Komponenten, die zur Realisierung benötigt werden, ist eine strikte Einhaltung der Vorgehensweisen zu empfehlen. Es gibt pro Modul noch viele weitere Konfigurationsmöglichkeiten wie Berechtigungskonzepte und Outputformate usw.. Je komplexer die Konfiguration desto schwieriger gestaltet sich auch die Fehlersuche. Eine detaillierte Dokumentation des Systemaufbaus ist hier besonders zu empfehlen.

3.3 Integration einer Testsuite in Hudson

3.3.1 Entwickeln von Testfällen mit utPLSQL

Die Tests sind in einzelnen Packages organisiert, die immer einen identischen Aufbau besitzen. Über Konventionen wird gesteuert welche Methoden während eines Testlaufs aufgerufen werden.

```
CREATE OR REPLACE PACKAGE ut_<testpackage>
IS
   PROCEDURE ut_setup;
   PROCEDURE ut_teardown;

   PROCEDURE ut_<testname 1>;
   PROCEDURE ut_<testname n>;
END ut_<testpackage>;
/
```

In den einzelnen Testmethoden wird über diverse Assertions geprüft, ob die aufgerufenen Methoden das gewünschte Ergebnis liefern. Ist dies nicht der Fall wird ein Fehler ausgegeben. Die einzelnen

ut_<testpackages> können in einer Testsuite organisiert werden, so dass eine Wiederverwendbarkeit gegeben ist. Ein Testfall ist letztendlich ein Wrapper um das zu testende Package.

3.3.2 Aufruf der Test mittels Maven

```
<plugin>
  <groupId>com.theserverlabs.maven.utplsql</groupId>
  <artifactId>maven-utplsql-plugin</artifactId>
  <version>1.1-SNAPSHOT</version>

  <dependencies>
      <dependency>
          <groupId>com.oracle</groupId>
          <artifactId>ojdbc14</artifactId>
          <version>9.0.2.0.0</version>
      </dependency>
  </dependencies>

  <configuration>
    <driver>oracle.jdbc.driver.OracleDriver</driver>
    <url>jdbc:oracle:thin:@localhost:1521:bjdb</url>
    <username>hudsontest</username>
    <password>hudson</password>
    <!--<packageName>betwnstr</packageName>-->
    <testSuiteName>All</testSuiteName>
  </configuration>
  <executions>
    <execution>
      <id>run-plsql-test-packages</id>
      <phase>process-test-resources</phase>
      <goals>
        <goal>execute</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Maven, mit dem Plugin für utPLSQL, bietet die Möglichkeit sich gegen eine Datenbank zu verbinden und eine Testsuite oder einzelne Testpackages auszuführen. Die Datei `pom.xml` enthält die komplette Konfiguration, welche einen Datenbankverbindung und die auszuführende Testsuiten/ Packages enthält.

Die Konfiguration kann per ‚`mvn test` überprüft werden. Durch dieses Kommando wird das Pom-File an Maven übergeben und ausgeführt. Die Konsole bildet das Protokoll für den Testlauf. Genau dieser Aufruf wird später von Hudson durchgeführt.

3.3.3 Integration von Maven in Hudson

Die eigentliche Arbeit ist zu diesem Zeitpunkt bereits durchgeführt! Die Infrastruktur ist vorbereitet, die Tests sind entwickelt und können mit Maven ausgeführt werden. Schlussendlich fehlt nur noch das Einrichten eines sogenannten „Jobs“ in Hudson.
Die minimalen Angaben beim einrichten eines „Jobs“ beschränken sich auf den Projektnamen, ein Repository inkl. der Zugangsdaten und einer zeitlichen Strategie. Neben diesen Parametern bestehen noch weitere Optionen wie Benachrichtigungsfunktionen, Archivierungsfunktionen u.v.m.

4. Sonstiges

In der Präsentation werden Beispiele für die einzelnen Testframeworks sowie die komplette Integration von utPLSQL in Hudson gezeigt. Ein komplexeres Beispiel wird in einer Livedemo präsentiert.

Kontaktadresse:

Name
Essential Bytes GmbH & Co.KG
Steinebühlstraße 30
D-77749 Hohberg
Telefon: +49 (0) 611 – 220 70 78
Fax: +49 (0) 7808 – 913 363
E-Mail bjoerger@essential-bytes.de
Internet: www.essential-bytes.de