НЕПРЕРЫВНАЯ ИНТЕГРАЦИЯ ПРИ РАЗРАБОТКЕ БАЗ ДАННЫХ

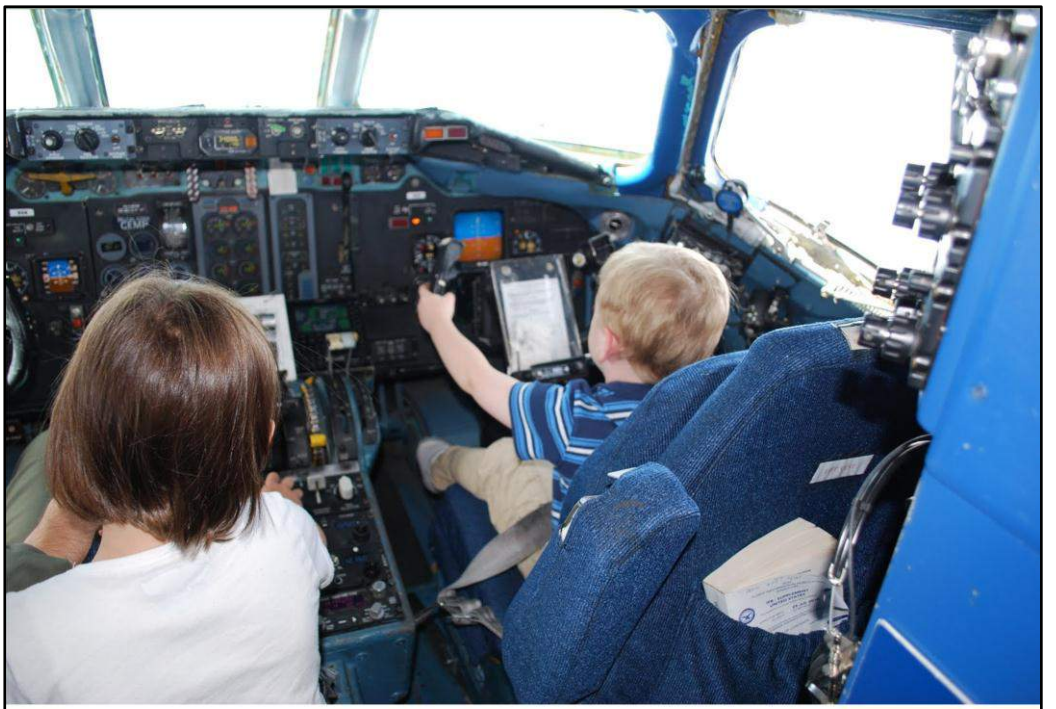
AGILE DataBase Oracle CI Continuous Integration XP

Владимир Бахов, Москва 2011

Владимир Бахов
AT-Consulting
vbakhov@at-consulting.ru
+7 (905) 7165446

Светлана Панфилова
AT-Consulting
spanfilova@at-consulting.ru
+7 (903) 1696490

Группа Google для данной презентации:**vobaks**
Исходные коды проекта доступны на **Google Code**.



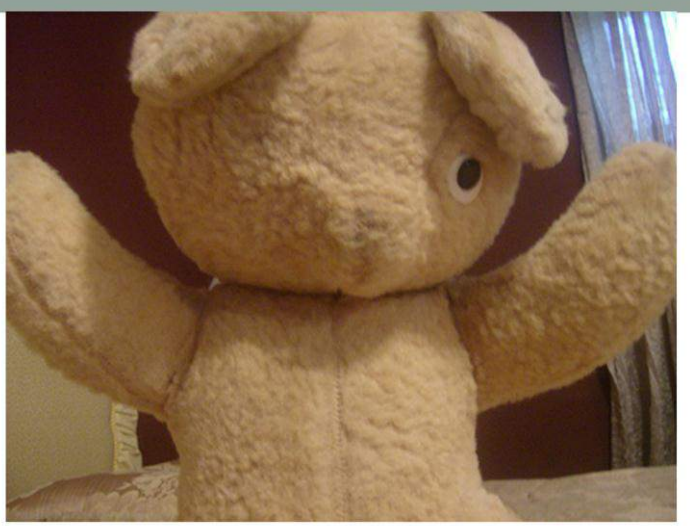
«Непрерывная интеграция»? - Автопилот для команды разработки!

Q: Что такое непрерывная интеграция?
A: Система, которая позволяет делать жизнь команды разработки много комфортнее!

Непрерывная интеграция нацелена на улучшения качества программного продукта и скорости его предоставления. Традиционный подход к контролю качества только после завершения разработки вытесняется непрерывной интеграцией в соответствии с agile парадигмами о том, что тестирование должно быть **ранним**, **автоматическим** и **постоянным**.

Вернемся на землю и посмотрим с какими сложностями приходится сталкиваться без CI.

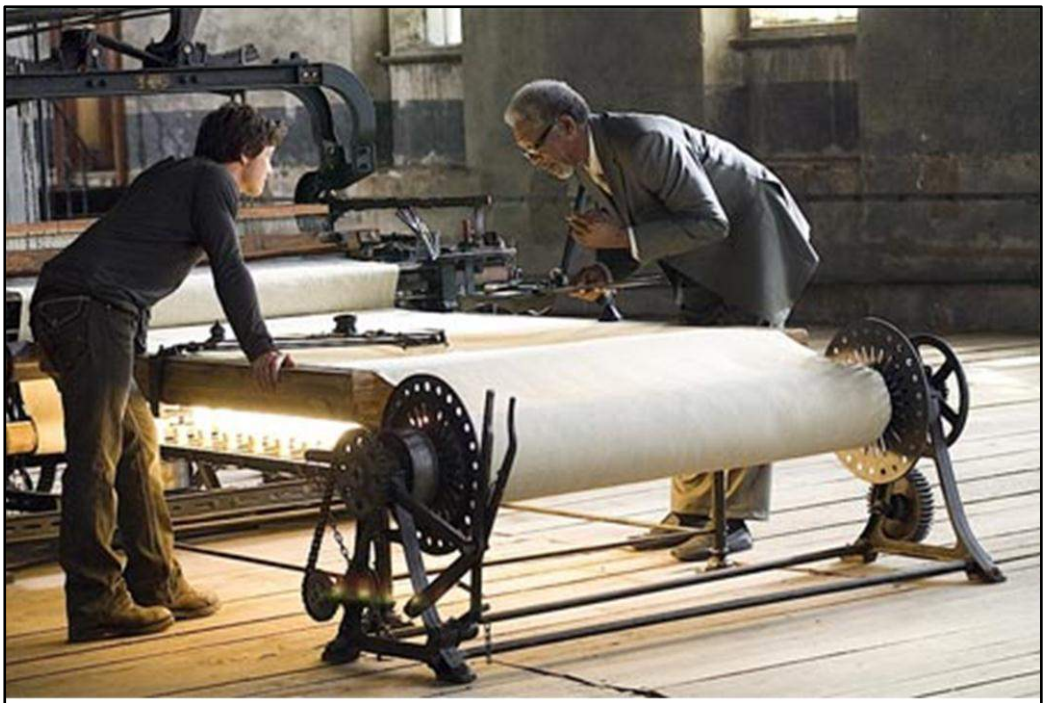
Ручные действия при сборке скриптов наката неизбежно ведут к дефектам и ошибкам



Нужные объекты или скрипты
не накатываются вовремя.
Или про них вообще забывают.

Ручные действия при сборке скриптов наката
неизбежно ведут к дефектам и ошибкам


Несовместимые версии баз в средах разработки, тестирования и продуктива
Нет соответствия версии приложения и базы данных – приложение часто ломается



Сакральные знания о накате концентрируются в руках одного человека. Он может заниматься интеграцией все свое время.

Сложно выпустить быстрый багфикс вне основной ветки кода

Даже если скрипты наката по каждой из задач верные, установка их в неправильной последовательности порождает дефекты

DBA во всем винят разработчиков, разработчики – DBA

Сборка в итоге может получиться не очень дружелюбной

Q: *Что мешает* ***постоянно*** *и* ***автоматически*** *собирать скрипт наката, проверять его ВСЕМИ имеющимися тестами?*

Основные преимущества

- Разработчики обнаруживают и исправляют ошибки внедрения постоянно, избегая хаоса в последние дни перед релизом.
- Быстрая нотификация о дефектном, несовместимом или невнедряемом коде.
- Меньше времени тратится на debug за счет версионности кода и легко доступной разницы кода между текущей и последней работающей сборкой.
- Быстрый результат всех регрессионных тестов на имеющемся коде
- Постоянная доступность текущей работающей сборки релиза для развертывания в продуктивную среду
- Быстрая обратная связь о качестве разрабатываемого кода, его функциональности и влиянии на всю остальную систему
- Метрики от автоматизированного тестирования и CI (покрытие тестами, сложность кода, цитируемость) помогают разработчикам фокусироваться на качественном коде и постоянном прогрессе команды.

Недостатки

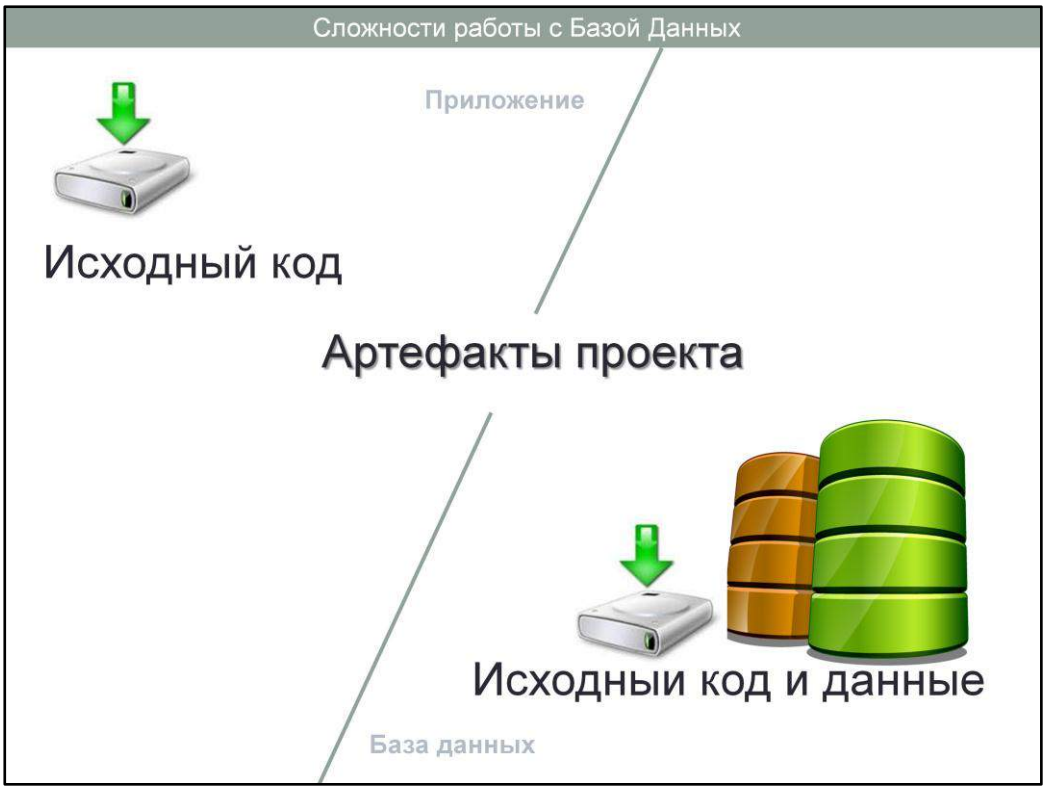
Необходимо время на разворачивание и настройку системы CI

Необходимы хорошие автоматизированные тесты для раскрытия всего потенциала CI

Крупномасштабный рефакторинг может быть затруднен в виду постоянно меняющейся базы исходного кода

Необходимость частой и быстрой сборки может потребовать дорогостоящих hardware ресурсов

Многие команды, использующие CI, считают, что его достоинства с лихвой превосходят недостатки.

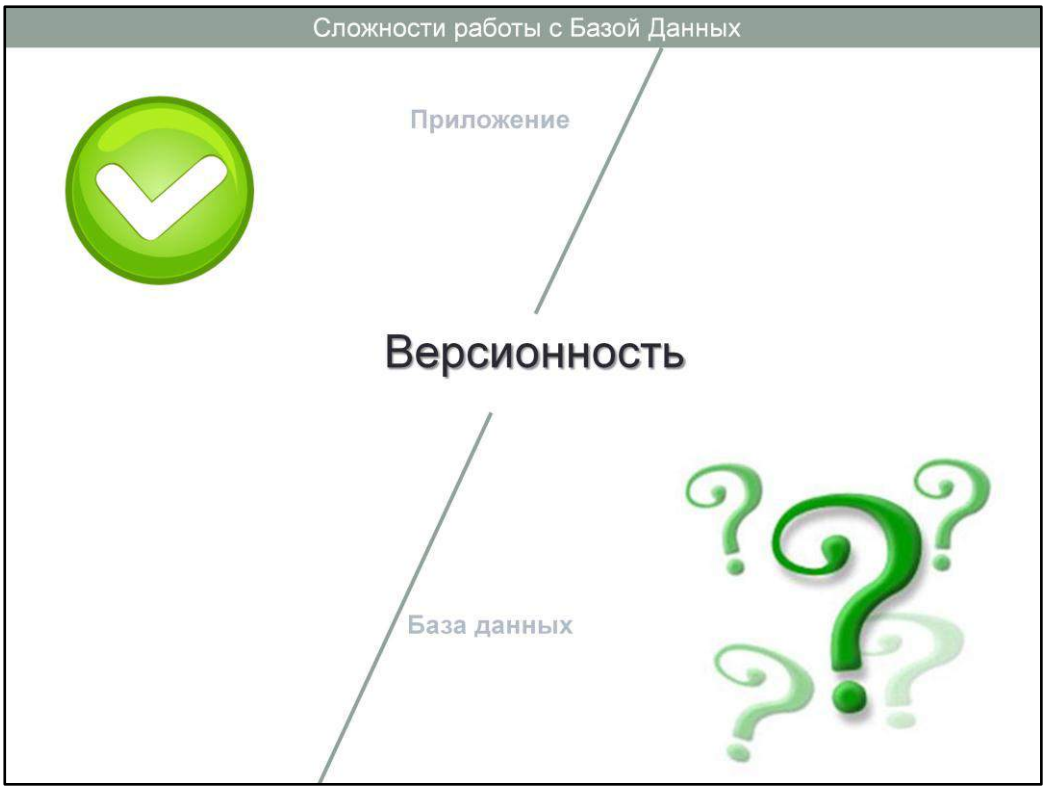


Приложение Ценностью является только код приложения и файлы конфигурации
База данных Ценностью является не только код, но и данные в базе

P.S. Под приложением понимаются артефакты проекта отличные от баз данных– программы написанные на Java, .NET и т.п.

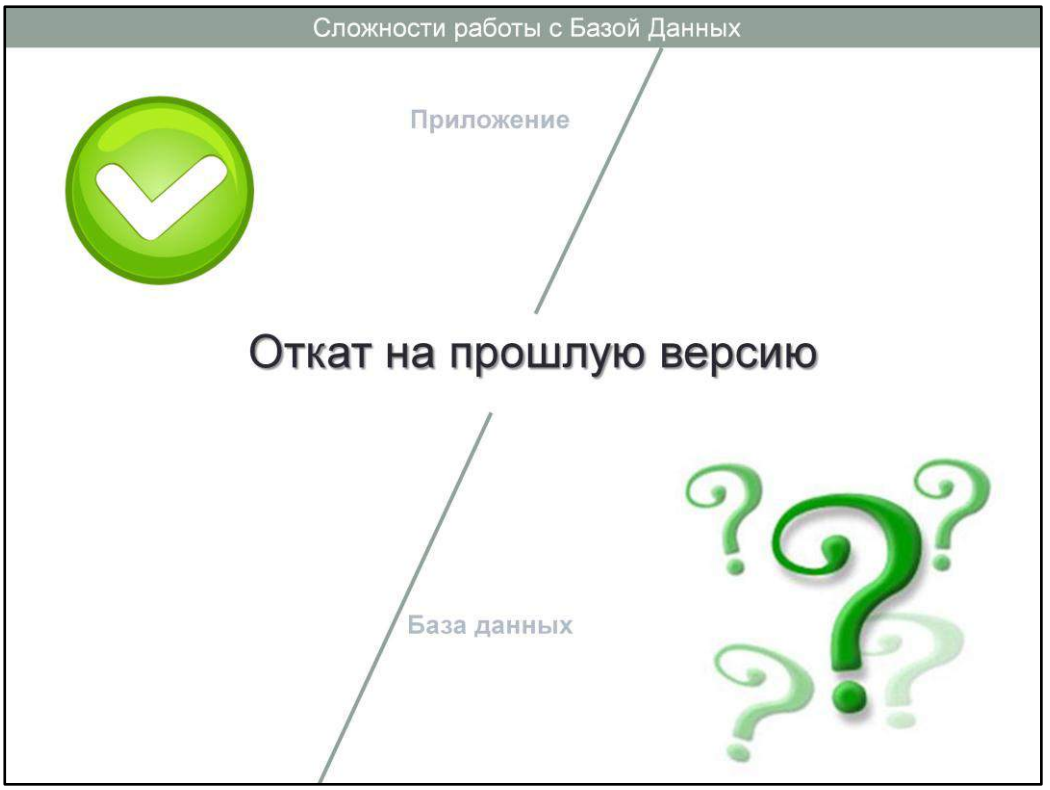
Приложение
Накат изменений происходит заменой старых файлов на новые. Последовательность не важна.

База данных
Для установки очередного релиза кроме замены кода, требуется запуск в продуктивной среде скриптов изменения структуры данных и, возможно, миграции данных. Для соблюдения целостности объекты и данные должны разворачиваться в строгой последовательности.



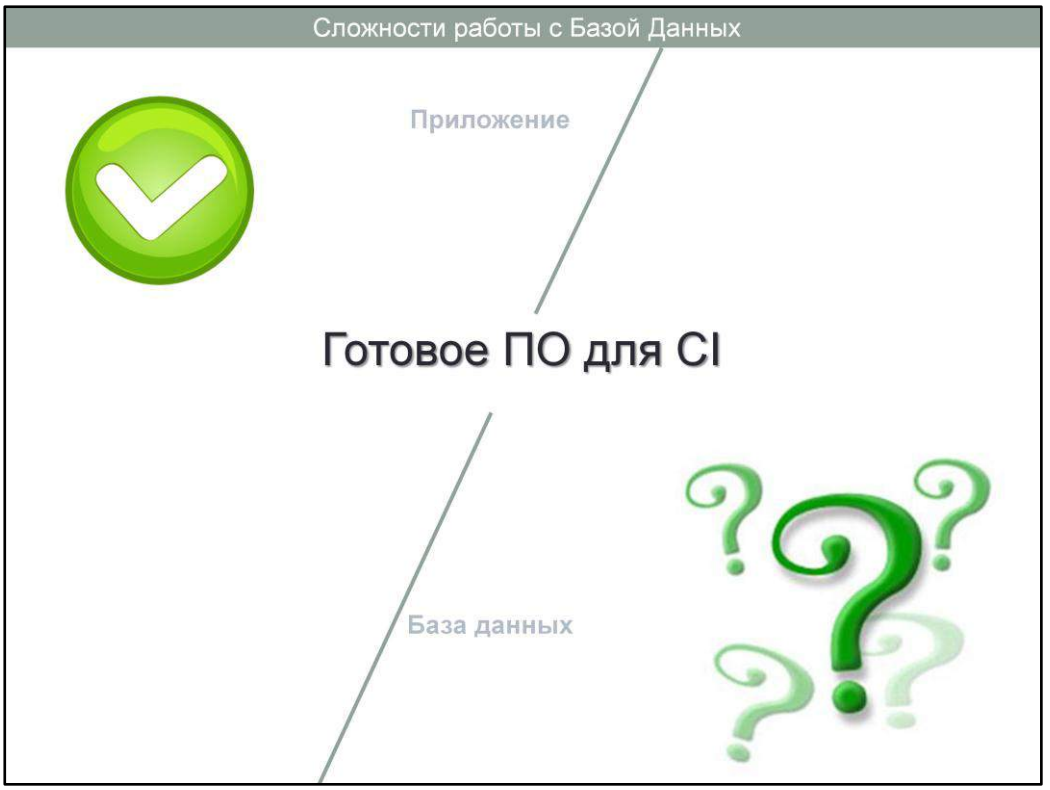
Приложение
Единый репозиторий контроля версий хранит всю историю изменений. Четкие правила работы с репозиторием.

База данных
Версионность кода может не использоваться. Отсутствует общепринятая методика работы с системой контроля версий для кода, скриптов изменения структуры и данных. Нет правил синхронизации изменений кода нескольких разработчиков.



Приложение Откат на n версий назад прост

База данных Откат на n версий назад требует написания трудоемкого ручного скрипта, который зависит от текущей версии базы, или бывает невозможен.

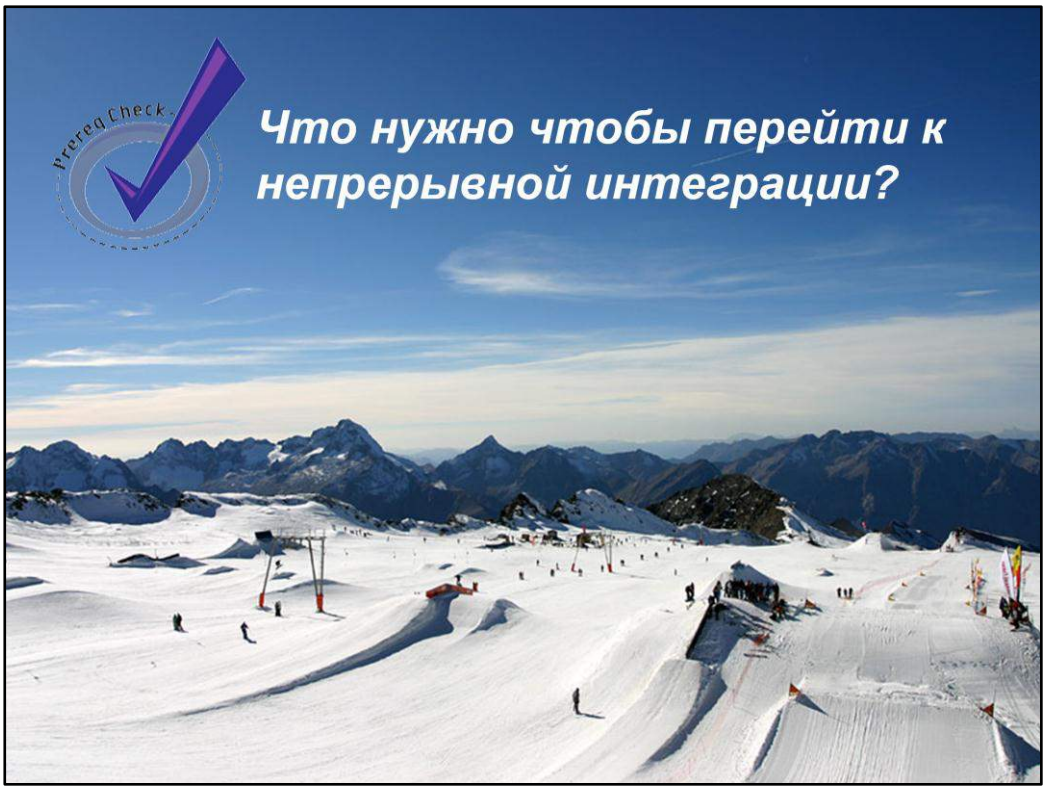


Приложение Существующие программы CI автоматизируют сборку приложений «щелчками мыши»

База данных Нет универсальных программных продуктов для поддержки CI БД. Скрипты наката для БД обычно собираются руками или в полуавтоматическом режиме.

Существует ли заветная кнопка для разработчиков базы данных?

Самый главный слайд!
CI для базы данных является достаточно сложным инструментом разработки. Но в жизни всему можно научиться! Какой бы сложной не была база – это возможно!
Сначала наша база просто поднималась из дампа 20 часов. После 2х недель рефакторинга и оптимизаций мы добились времени в 20 минут. Сейчас установка релиза базы (а мы делаем это минимум раз в день) занимает порядка 2х минут.


Prereq Check
Что нужно чтобы перейти к непрерывной интеграции?

Система контроля версий

**Subversion - наша любимая система версионности*

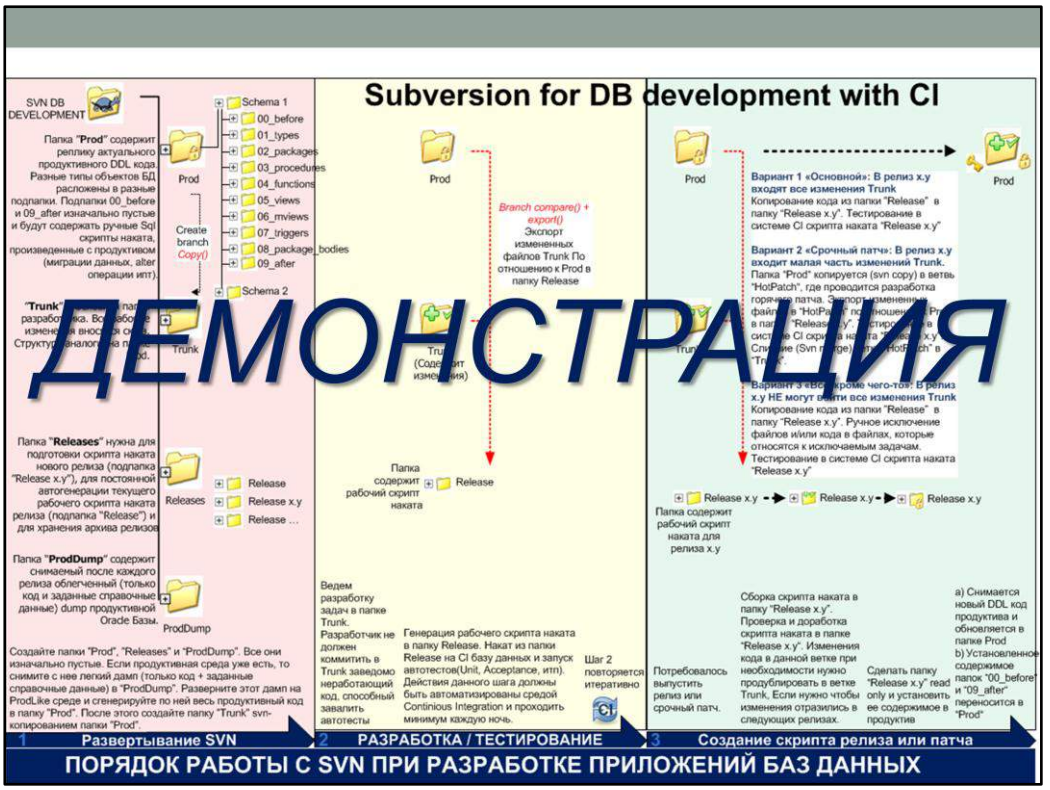
У вас должны быть автоматизированные тесты базы данных, иначе CI не имеет смысла. UTPLSql – наш любимый фрэймворк создания автотестов для oracle базы данных. Для автотестов oracle базы данных также можно использовать Toad Code Tester, SQL Unit test для SQL Developer, самописное решение. Автотестирование БД и ETL процессов – тема отдельной презентации.

ПО непрерывной интеграции для автоматизированной сборки скриптов наката и их развертывания

Остановимся на системе версионности более подробно.

- Все изменения должны происходить скриптами (никаких изменений структуры таблицы через GUI, например SQL Навигатора).
- Скрипты исходного кода вашей базы должны быть в виде файлов sql: пакеты, процедуры, представления, триггеры и т.д. Скрипты по изменению структуры и миграции данных должны быть в виде файлов sql: alter операции, создание/изменение таблиц insert операции итп. Скрипты - главный артифакт проекта.
- Все скрипты должны подпадать под систему контроля версий
- Скрипты имеют владельца, версию – их можно поддерживать
- В отличии от ручных изменений, скрипты можно автоматически запускать и многократно тестировать.
- Скрипты можно переносить на разные платформы и среды.
- Скрипты можно автоматически связать с задачами системы Bug Tracking (Jira и тп.). Всегда можно посмотреть, кто, что и когда менял в рамках данной задачи.

Организация работы с ветками и способ сборки релиза для кода базы данных отличается от классического подхода, используемого при разработке, скажем, WEB-приложений. Предполагается что у вас уже есть система контроля версий и вы умеете проводить с ней базовые операции.


Subversion for DB development with CI
ДЕМОНСТРАЦИЯ
SVN DB DEVELOPMENT
Папка "Prod" содержит реплику актуального продуктивного DDL кода. Разные типы объектов БД расложены в разные подпапки. Подпапки 00_before и 09_after изначально пустые и будут содержать ручные Sql скрипты наката, произведенные с продуктивом (миграции данных, alter операции ипт)
Prod
Create branch Copy()
Trunk
Schema 1
00_before
01_types
02_packages
03_procedures
04_functions
05_views
06_mviews
07_triggers
08_package_bodies
09_after
Schema 2
Папка "Releases" нужна для подготовки скрипта наката нового релиза (подпапка "Release x.y"), для постоянной автогенерации текущего рабочего скрипта наката релиза (подпапка "Release") и для хранения архива релизов
Releases
Release
Release x.y
Release ...
Папка "ProdDump" содержит снимаемый после каждого релиза облегченный (только код и заданные справочные данные) dump продуктивной Oracle Базы.
ProdDump
Создайте папки "Prod", "Releases" и "ProdDump". Все они изначально пустые. Если продуктивная среда уже есть, то снимите с нее легкий дамп (только код + заданные справочные данные) в "ProdDump". Разверните этот дамп на ProdLike среде и сгенерируйте по ней весь продуктивный код в папку "Prod". После этого создайте папку "Trunk" svn-копированием папки "Prod".
1 Развертывание SVN
Prod
Branch compare() + export()
Экспорт измененных файлов Trunk По отношению к Prod в папку Release
Папка содержит рабочий скрипт наката
Release
Ведем разработку задач в папке Trunk. Разработчик не должен коммитить в Trunk заведомо неработающий код, способный завалить автотесты
Генерация рабочего скрипта наката в папку Release. Накат из папки Release на CI базу данных и запуск автотестов(Unit, Acceptance, итп). Действия данного шага должны быть автоматизированы средой Continious Integration и проходить минимум каждую ночь.
Шаг 2 повторяется итеративно
2 РАЗРАБОТКА / ТЕСТИРОВАНИЕ
Prod
Вариант 1 «Основной»: В релиз x.y входят все изменения Trunk
Копирование кода из папки "Release" в папку "Release x.y". Тестирование в системе CI скрипта наката "Release x.y"
Вариант 2 «Срочный патч: В релиз x.y входит малая часть изменений Trunk.
Папка "Prod" копируется (svn copy) в ветвь "HotPatch", где проводится разработка горячего патча. Экспорт изменен
Вариант 3 «Все кроме чего-то»: В релиз x.y НЕ могут войти все изменения Trunk
Копирование кода из папки "Release" в папку "Release x.y". Ручное исключение файлов и/или кода в файлах, которые относятся к исключаемым задачам. Тестирование в системе CI скрипта наката "Release x.y"
Release x.y
Release x.y
Release x.y
Папка содержит рабочий скрипт наката для релиза x.y
Потребовалось выпустить релиз или срочный патч.
Сборка скрипта наката в папку "Release x.y". Проверка и доработка скрипта наката в папке "Release x.y". Изменения кода в данной ветке при необходимости нужно продублировать в ветке Trunk, Если нужно чтобы изменения отразились в следующих релизах.
Сделать папку "Release x.y" read only и установить ее содержимое в продуктив
a) Снимается новый DDL код продуктива и обновляется в папке Prod
b) Установленное содержимое папок "00_before" и "09_after" переносится в "Prod"
3 Создание скрипта релиза или патча
ПОРЯДОК РАБОТЫ С SVN ПРИ РАЗРАБОТКЕ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

Live Demo

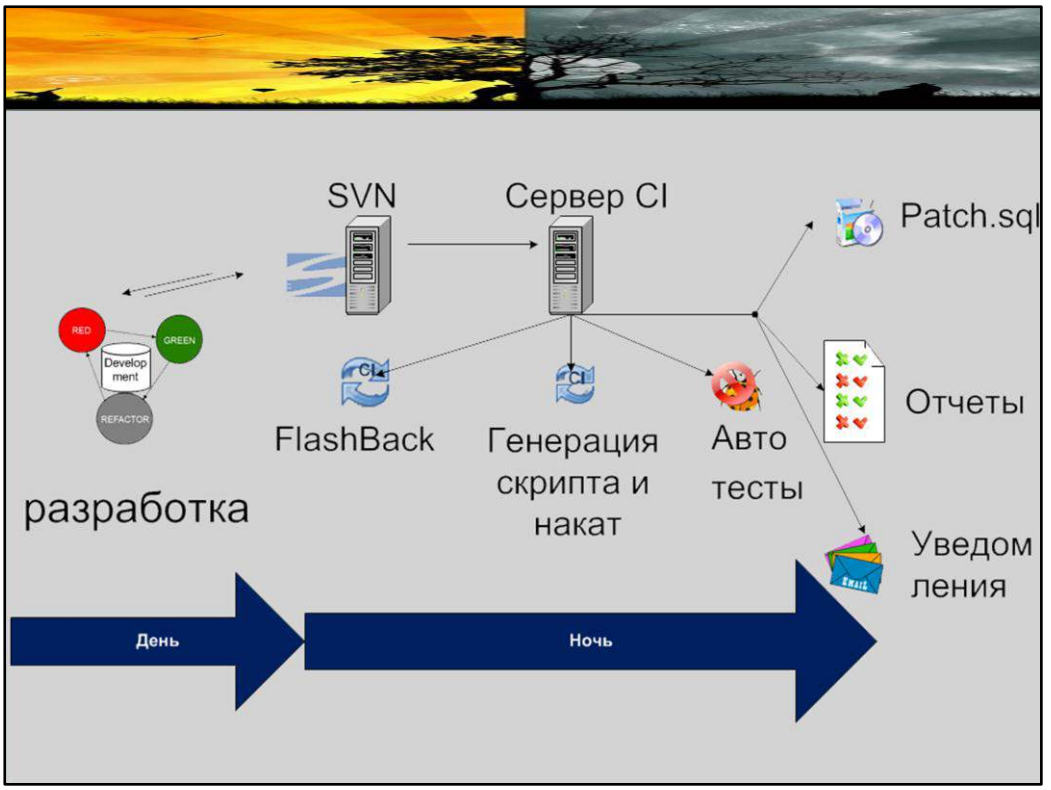
TRUNK – единая мастер копия базы со всей историей изменения структур, данных и кода.

- Все версии изменений кода, структур и данных за всю историю проекта доступны в папке TRUNK.
- TRUNK – единая мастер копия базы.

Удобный доступ и возможность ветвления разработки

- Удобная для разработчиков организация – объекты кода и скрипты изменения структур и данных не раскиданы по папкам разных релизов. Чтобы начать работать над желаемым кодом достаточно открыть файл из папки TRUNK в любимой IDE и сохранить перед выходом.
- При ветвлении разработки система исключает из скрипта наката уже попавшие на продуктив не только изменения кода, но и изменений структур, миграции и прочие ручные sql изменения по задачам баг-трекера.

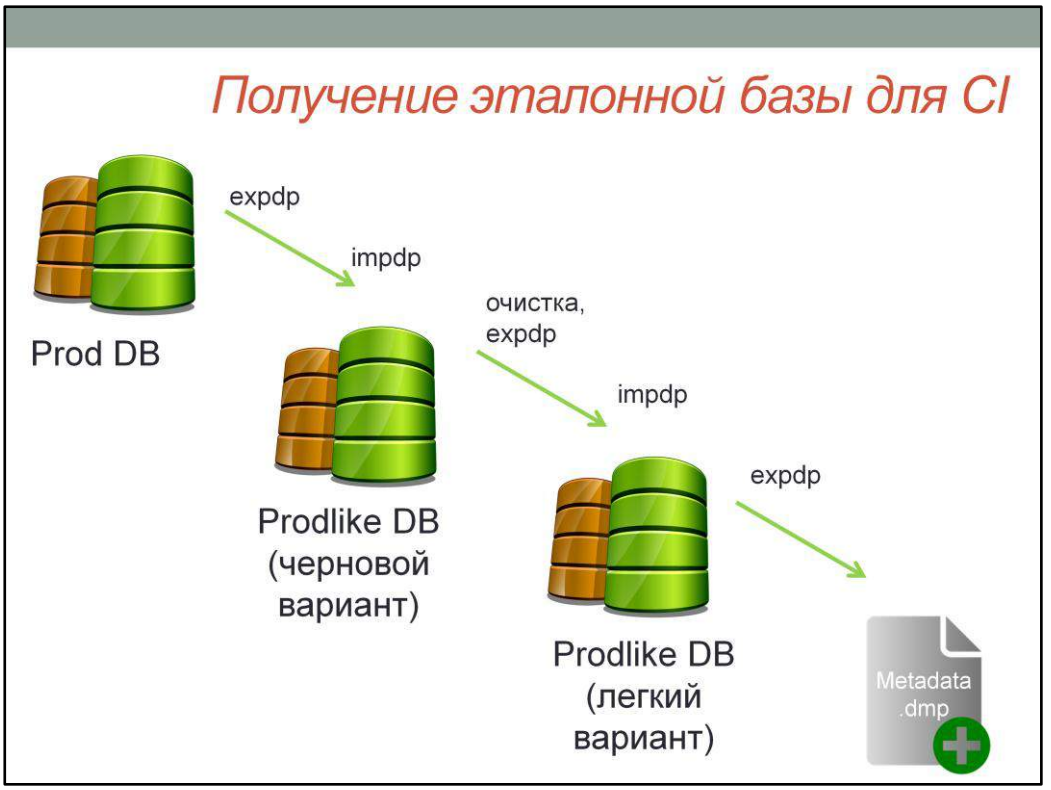

Суточный цикл непрерывной интеграции



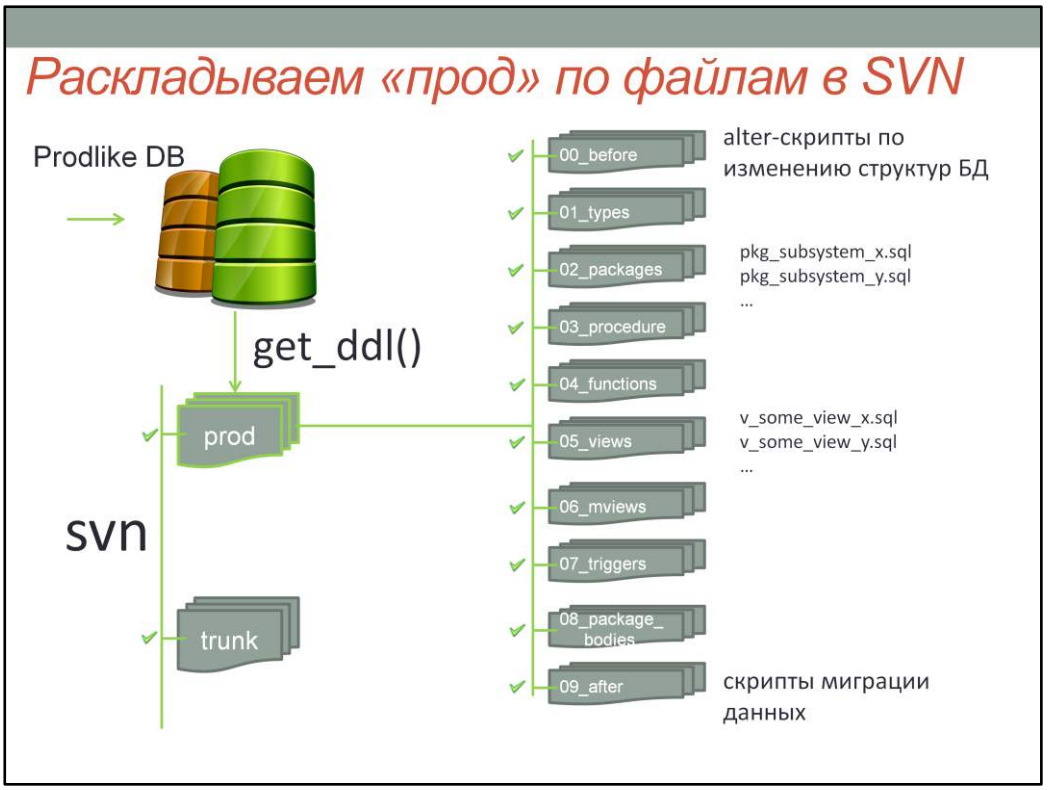
День.
Итеративная разработка/bugfix и commits в систему контроля версий.

Ночь.

1) Prodlike FlashBack() или востановление из Cold backup базы данных. Откат всех изменений. Возврат Prodlike среды в идентичное текущему продуктивному состояние
2) Автоматически полученный как разница между папкой «Prod» и «Trunk», скрипт наката устанавливается на ProdLike. SQL файлы скрипта наката разворачиваются в фиксированном по типам объектов и типу изменений порядке (изменения структуры, код, потом миграции данных, рекомпиляция и т.п.)
3) Запускается автоматизированное тестирование
4) Файл текущей версии скрипта наката поступает в репозиторий релизов
5) На Dashboard сервера CI поступает отчет о ночной сборке и пройденных тестах
6) Заинтересованным лицам и ответственным за дефекты разработчикам рассылаются уведомления



- Первоначальное получение Prodlike DB
  - Импорт дампа утилитой impdp
  - Восстановление из бэкапа
- Очистка (удаление лишних партиций и т. п.)
- Сохранение нового дампа в SVN папку «dump» для быстрого развертывания новых сред CI

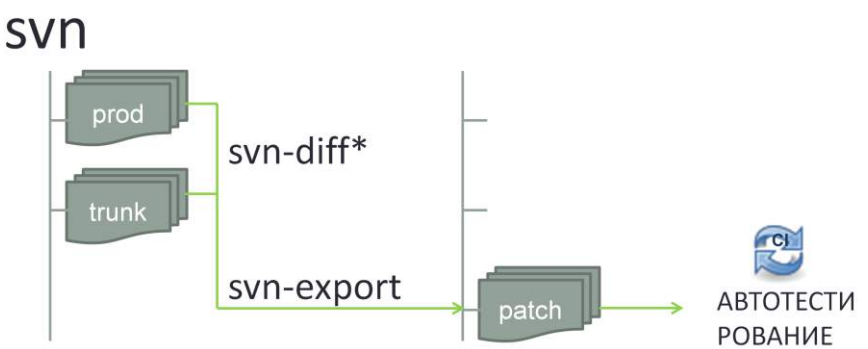


В папке «prod» хранится текущий исходный код (как на продуктиве). Папка используется как эталон для сравнения при генерации скрипта наката. Структура «trunk» аналогична «prod», но в данной папке ведется разработка. В нулевой точке папки «trunk» и «prod» равны.

Запись DDL кода в файловую систему с помощью UTL_FILE. Каждый объект располагается в отдельном файле.

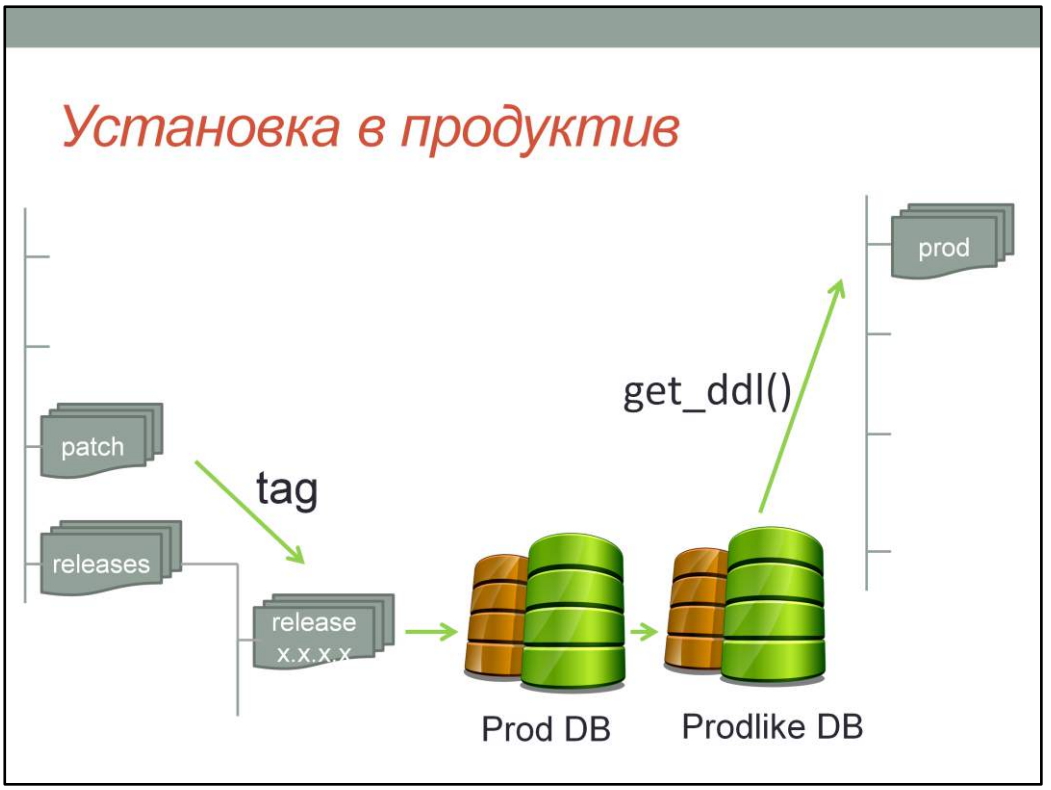
- **Types, Type Bodies**. Создание и удаление типов в правильном порядке. Сначала каскадно (по dba_dependencies) удаляются, затем создаются в порядке зависимостей друг от друга. Гранты на них. Все типы системы хранятся в одном файле, и перенакатываются целиком
- **Triggers**. Исключение триггеров на views и materialzied views.
- **Functions, Procedures, Packages**. Гранты на них.
- **Views**, а также триггеры и гранты на них.
- **Materialized Views**. Триггеры, индексы и гранты на них.
- **Package Bodies**.
- **…**

Автогенерация скрипта наката



*Используется java-утилита **svn_diff_export**

- Разница между prod и trunk получается посредством svn-команды diff
- Патч получается посредством команды export над результатом diff
- Папка patch повторяет структуру папок prod и trunk, но содержит только те поддиректории и файлы, которые образуют разницу между prod и trunk



ANT: SQLPLUS вызывает все SQL-файлы из имеющихся директорий в определенном порядке.

Варианты наката патча.

- Последовательный накат всех sql-файлов, отсортированных по имени, из всех директорий, попавших в папку patch. В данном случае для установки патча на продуктив администратор должен также использовать svn и вызывать установку из ant, задавая параметры (логин/пароль/sid) через файл свойств (build.properties)
- Конкатенация всех sql-файлов в отсортированном порядке и накат единого файла. В данном случае администратору высылается единый sql-файл, который он может запустить из sqlplus

Средства ant, используемые для наката скриптов

- Для наката sql-файлов через sqlplus используется плагин *incanto* для ant, обеспечивающий портируемость на разные среды
- Из наката можно исключать файлы посредством указания маски в атрибуте

excludes тасков *concat* или *fileset*. Таким образом, из наката можно исключать пакеты автоматизированного тестирования, которые должны попадать только на среды тестирования и CI

Собирайте и тестируйте автоматически и постоянно!

Согласно описанной методологии с помощью ANT скриптов подключите автоматическую сборку скриптов наката релиза из системы контроля версий к системе CI, также к CI подключите ваши автотесты (мы используем maven-utplsql-plugin).

Средства мониторинга

- Быстрое отображение успешности компиляции и развертывания
- Детальное отображение успешности автотестирования
- Состояние текущей сборки наглядно для всех членов команды
- Устройства визуализации делают процесс интереснее

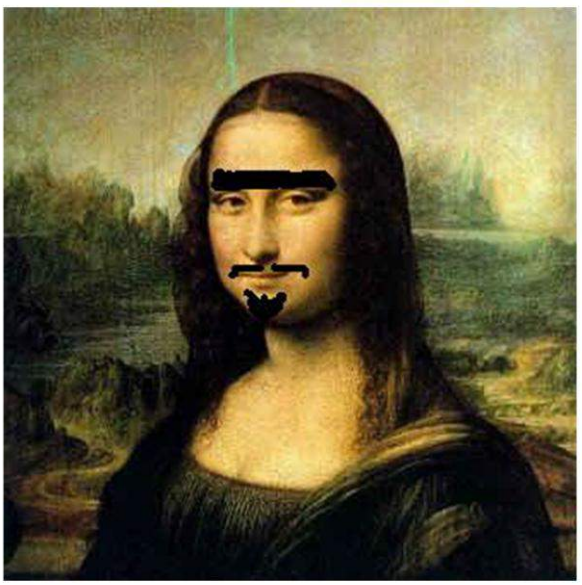
Эффект от нахождения и исправления на ранних этапах разработки дефектов кода и дефектов внедрения приносит существенное снижение издержек при производстве ПО.

CI Checkpoints

* ВСЕМИ – не всегда обязательно – ваша система автотестирования может получать из системы контроля версий только изменившиеся за день объекты и найти все зависимые по коду от них объекты (в oracle это делает через dba_dependencies). В ETL программных продуктах вы также можете анализировать зависимости.

Используйте легковесные синтетические данные для автотестов.

- оптимизируйте автотесты
  - используйте синтетические данные
  - автоматизируйте получение синтетических данных
- инвестируйте в инфраструктуру
- используйте базовые средства системы CI по распараллеливанию вычислений на подключенных машинах (build agent)



DBA поддержки не вносят несогласованные изменения в продуктивный код. Все изменения только через CI.

Разработчик может поднять нужную легковесную версию базы из скриптов наката и дампа в системе контроля версий на требуемой среде для целей разветвленной разработки и тестирования.


Спасибо!

Опциональная часть

Рекомендации oracle-разработчику

- Весь код должен быть пригоден для наката через sqlplus. Views/Materialized Views не должны содержать пустых строк в теле. Скрипты PL/SQL должны оканчиваться символом **/**
- Для обеспечения наката объектов без ошибок необходимо пользоваться порядком наката объектов. Для этого нужно отказаться от использования процедур/функций PL/SQL. Вместо них используются пакеты: сначала накатываются заголовки, затем views/materialized views, в последнюю очередь тела пакетов
- В папку 00_before скрипты по изменению структур БД также необходимо выкладывать в порядке, обеспечивающем корректность при накате: сначала удаление объектов, потом добавление
- При подключении новых источников через db links, необходимо вносить скрипты их наката в svn
- При импорте/экспорте необходимо выставлять одинаковую кодировку клиента (NLS_LANG)

CI Базовые принципы

- Держите код в системе версионности.
- Автоматизируйте поднятие легкой ProdLike среды, содержащей только код и заданные справочные данные.
- Автоматизируйте создание программы сборки и наката на ProdLike среду тестирования текущей версии релиза.
- Сделайте сборку релиза самотестирующейся.
- Разработчики фиксируют изменения (commit) каждый день.
- Следите, чтобы сборка и тесты были быстрыми.
- Держите ProdLike среду тестирования идентичной продуктивной среде: код, версии базы данных, ОС и прочей инфраструктуры.
- Запретите DBA поддержи изменять любые объекты кода в продуктиве без уведомления. Разграничьте какие объекты/данные является кодом, а какие – настроечными параметрами.
- Делайте пользовательское тестирование функциональности как можно раньше.
- Результат сборки релиза должен быть доступен всем.
- Автоматизируйте развертывание релиза в продуктивную среду.

Разъяснения по данным пунктам можно найти здесь:
http://en.wikipedia.org/wiki/Continuous_integration

Установка и настройка БД для CI

- Установка БД из шаблона (.dbt) с необходимыми настройками (размер блока, кодировка) и tablespaces с помощью DBCA
- Создание директорий Oracle для импорта дампов и экспорта DDL
- Выдача необходимых грантов пользователю SYSTEM (или другим системным пользователям)
- Эмуляция внешних источников (db links)
- Первоначальное наполнение необходимыми данными
- Установка средств автоматизированного тестирования UTP
- Сохранение Prodlike DB: бэкапирование, настройка flashback.
- Экспорт сохраняемого в БД кода (Packages, Views и т. д.) в svn

Код простой системы версионности для oracle в дополнение к SVN

Если у вас совсем нет версионности в Oracle, то вам практически ничего не будет стоит включить ее в виде тригера на все DDL операции с базой (если они проходят не супер часто, что может отрицательно сказаться на производительности системы).

```
CREATE TABLE aud_data_log
   (time_key DATE,
   sid NUMBER,
   serial NUMBER,
   username VARCHAR2(30 BYTE),
   osuser VARCHAR2(4000 BYTE),
   host VARCHAR2(4000 BYTE),
   logon_time DATE,
   owner VARCHAR2(30 BYTE),
   object_name VARCHAR2(60 BYTE),
   txt clob)
/

create or replace procedure write_aud_data_log
( asid number, aserial number, ausername varchar2, aosuser varchar2, ahost varchar2,
  alogon_time date, aowner varchar2, aobj varchar2, txt clob)
is
 pragma autonomous_transaction;
begin
   insert into aud_data_log
   (time_key, sid, serial, username, osuser, host, logon_time, owner, object_name, txt)
   values (sysdate, asid, aserial , ausername , aosuser , ahost , alogon_time , aowner , aobj , txt);
 commit;
end;
/
grant execute on write_aud_data_log to ВВЕДИТЕ имена всех схем, которые хотите мониторить
/
-- ЗАПУСТИТЬ на ВСЕХ схемах, которые вы хотите мониторить.
create or replace trigger log_ddl after DDL on schema
declare
  --atxt varchar2(32767);
  li ora_name_list_t;
  l2 ora_name_list_t;
  n number;
  k number;
  asid number;
  aserial number;
  ausername varchar2(30);
  aosuser varchar2(30);
  ahost varchar2(30);
  alogon_time date;
  aobj varchar2(4000);
  v_string clob;
begin
  select SID, SERIAL#, username, upper(osuser), upper(machine), logon_time
    into asid, aserial, ausername, aosuser, ahost, alogon_time
    from v$session s
   where --AUDSID=userenv('sessionid') and rownum=1;
         SID=SYS_CONTEXT('USERENV','SID');

  n:=ora_sql_txt(li);
  for i in 1..nvl(n, 0) loop
   v_string:=v_string||li(i);
  end loop;

  n:=ora_dict_obj_name_list(li);
  k:=ora_dict_obj_owner_list(l2);
  aobj:=ora_dict_obj_owner||'.'||ora_dict_obj_name;

  for i in 1..nvl(greatest(nvl(n,0),nvl(k,0)),0) loop
    aobj:=aobj||' '||case when i<=k then l2(i) ||'.'
                  else null end||
                  case when i<=n then li(i) else null end||', ';
  end loop;

  CXEMA_где_создали_"aud_data_log".write_aud_data_log(asid, aserial, ausername,
aosuser, ahost, alogon_time, ora_dict_obj_owner,ora_dict_obj_name, v_string);
  end;
```

ANT

- java-утилита для автоматизации процесса сборки программного продукта.
- Ant — платформонезависимый аналог UNIX-утилиты make (в качестве *«Makefile»* применяется *«build.xml»*).
- Управление процессом сборки происходит посредством XML-сценария
- Сценарий содержит определение проекта, состоящего из отдельных целей (***Targets***).
- Цели сравнимы с процедурами в языках программирования и содержат вызовы команд-заданий (***Tasks***).

Часто применяемые задания (Tasks)

Код	Действие
Javac	компиляция Java-кода
Copy	копирование файлов
Delete	удаление файлов и директорий
Move	перемещение файлов и директорий
Replace	замещение фрагментов текста в файлах
JUnit	автоматический запуск юнит-тестов
Exec	выполнение внешней команды
Zip	создание архива в формате Zip
SVN	**выполнение SVN-команды**
Mail	отправка электронной почты
Xslt	наложение Xslt-преобразования

http://ru.wikipedia.org/wiki/ANT

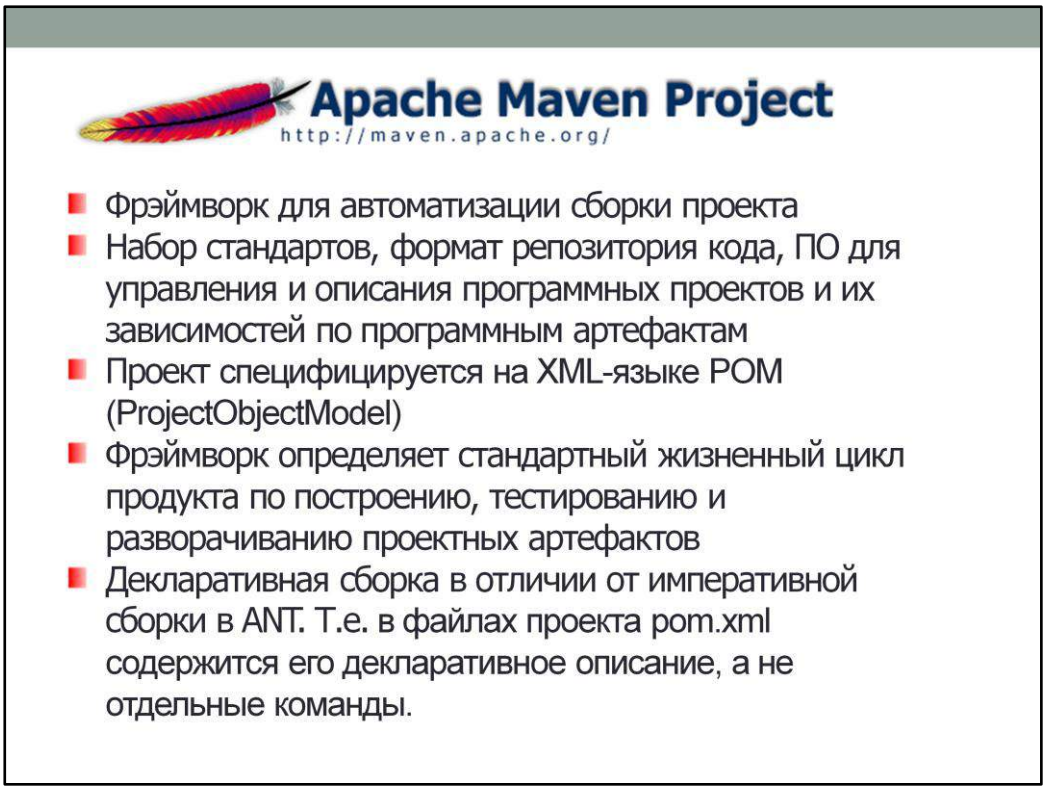
Ant для автоматизации CI

Используемые плагины

- ***Incanto***. Для работы с sqlplus
- ***Svnkit***. Для работы с svn
- ***Ant-contrib***. Для сортировки sql-файлов при накате
- ***svn-diff-export***. Для получения разницы между prod и trunk и ее экспорта в папку patch.

Основные цели (targets) ant

- Import. Получение Prodlike среды: импорт, очистка, настройка flashback и создание точки восстановления
- CI. Откат БД к точке восстановления, удаление папки patch, генерация нового патча, накат патча
- UT. Вызов unit-тестов



Maven нужен, для запуска Unit-тестов UTPLSQL с помощью плагина maven-utplsql-plugin.

Цитата-бонус

“Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove.”

-- Martin Fowler