Grails Transactions

Burt Beckwith

SpringSource

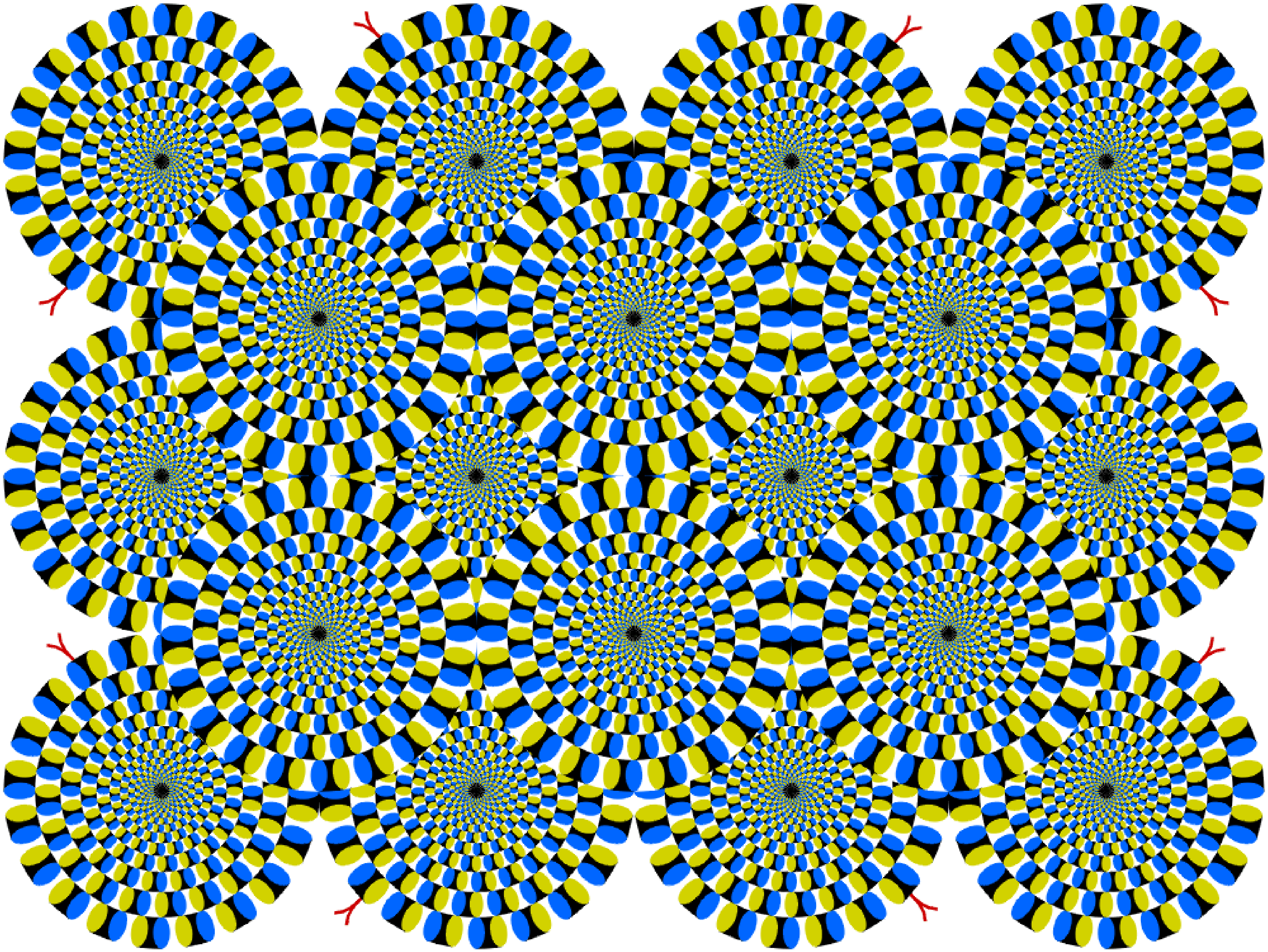
@burtbeckwith

https://burtbeckwith.com/blog/

What is a Transaction?

What is a Transaction?

- `getConnection()`
- `setAutoCommit(false)`
- **do work**
- `COMMIT`
- **Plus error handling, rollback if necessary**



ACID

- Atomicity
- Consistency
- Isolation
- Durability

How To Use Transactions in Grails?

How To Use Transactions in Grails?

- Transactional services
- Spring's `@Transactional` annotation
- And new in Grails 2.3, the `grails.transaction.Transactional` annotation

Grails Transactional services

- `$ grails create-service the`

```
package tx

class TheService {

}
```

@Transactional

@Transactional

- String value() default "";
- Propagation propagation() default Propagation.REQUIRED;
- Isolation isolation() default Isolation.DEFAULT;
- int timeout() default TransactionDefinition.TIMEOUT_DEFAULT;
- boolean readOnly() default false;
- Class<? extends Throwable>[] rollbackFor() default {};
- String[] rollbackForClassName() default {};
- Class<? extends Throwable>[] noRollbackFor() default {};
- String[] noRollbackForClassName() default {};

@Transactional

- String value() default "";
- Propagation propagation() default Propagation.REQUIRED;
- **Isolation isolation() default Isolation.DEFAULT;**
- int timeout() default TransactionDefinition.TIMEOUT_DEFAULT;
- boolean readOnly() default false;
- Class<? extends Throwable>[] rollbackFor() default {};
- String[] rollbackForClassName() default {};
- Class<? extends Throwable>[] noRollbackFor() default {};
- String[] noRollbackForClassName() default {};

Transaction Isolation

- **org.springframework.transaction.annotation.Isolation**
- **READ_UNCOMMITTED**
  - Order is maintained
  - Allows dirty reads
- **READ_COMMITTED**
  - Only reads committed data, but allows non-repeatable reads
- **REPEATABLE_READ**
  - Allows phantom reads
- **SERIALIZABLE**
  - Blocks read and write access between transactions, guarantees correct results
- **DEFAULT**
  - Use the default database isolation, often READ_COMMITTED (MySQL uses REPEATABLE_READ)

@Transactional

- String value() default "";
- **Propagation propagation() default Propagation.REQUIRED;**
- Isolation isolation() default Isolation.DEFAULT;
- int timeout() default TransactionDefinition.TIMEOUT_DEFAULT;
- boolean readOnly() default false;
- Class<? extends Throwable>[] rollbackFor() default {};
- String[] rollbackForClassName() default {};
- Class<? extends Throwable>[] noRollbackFor() default {};
- String[] noRollbackForClassName() default {};

Transaction Propagation

- org.springframework.transaction.annotation.Propagation
- REQUIRED
- SUPPORTS
- MANDATORY
- REQUIRES_NEW
- NOT_SUPPORTED
- NEVER
- NESTED

@Transactional

- String value() default "";
- Propagation propagation() default Propagation.REQUIRED;
- Isolation isolation() default Isolation.DEFAULT;
- int timeout() default TransactionDefinition.TIMEOUT_DEFAULT;
- boolean readOnly() default false;
- **Class<? extends Throwable>[] rollbackFor() default {};**
- **String[] rollbackForClassName() default {};**
- **Class<? extends Throwable>[] noRollbackFor() default {};**
- **String[] noRollbackForClassName() default {};**

@Transactional

- **String value() default "";**
- Propagation propagation() default Propagation.REQUIRED;
- Isolation isolation() default Isolation.DEFAULT;
- **int timeout() default TransactionDefinition.TIMEOUT_DEFAULT;**
- **boolean readOnly() default false;**
- Class<? extends Throwable>[] rollbackFor() default {};
- String[] rollbackForClassName() default {};
- Class<? extends Throwable>[] noRollbackFor() default {};
- String[] noRollbackForClassName() default {};

Annotated Service Examples

Annotated Service Examples

```
package com.mycompany

import org.springframework.transaction.annotation.Propagation
import org.springframework.transaction.annotation.Transactional

@Transactional
class SomeService {

   def someMethod() {
      ...
   }

   @Transactional(propagation=Propagation.MANDATORY)
   def someOtherMethod() {
      ...
   }
}
```

Annotated Service Examples

```
package com.mycompany

import org.springframework.transaction.annotation.Propagation
import org.springframework.transaction.annotation.Transactional

class SomeOtherService {

   def someMethod() {
      ...
   }

   @Transactional
   def someOtherMethod() {
      ...
   }

   @Transactional(propagation=Propagation.MANDATORY)
   def yetAnotherMethod() {
      ...
   }
}
```

Unintentionally Bypassing the Bean Proxy

Unintentionally Bypassing the Bean Proxy

```
@Transactional
void someMethod(...) {
   // do some work ...
   storeAuditData(...)
}

@Transactional(propagation=Propagation.REQUIRES_NEW)
void storeAuditData(...) {
   //
}
```

Unintentionally Bypassing the Bean Proxy

```
def grailsApplication

@Transactional
void someMethod(...) {
   // do some work ...
   def myProxy = grailsApplication.mainContext.fooService
   myProxy.storeAuditData(...)
}

@Transactional(propagation=Propagation.REQUIRES_NEW)
void storeAuditData(...) {
   //
}
```

Spring Classes and Utility Methods

PlatformTransactionManager

- org.springframework.transaction.PlatformTransactionManager

  (org.springframework.orm.hibernate3.HibernateTransactionManager)

  - TransactionStatus getTransaction(TransactionDefinition definition) throws TransactionException;
  - void commit(TransactionStatus status) throws TransactionException;
  - void rollback(TransactionStatus status) throws TransactionException;

TransactionSynchronizationManager

- org.springframework.transaction.support.TransactionSynchronizationManager
  - public static void bindResource(Object key, Object value)
  - public static Object unbindResource(Object key)
  - public static boolean hasResource(Object key)
  - public static Object getResource(Object key)

TransactionSynchronizationManager

- For example:

  -
    ```
    TransactionSynchronizationManager.bindResource(
          getSessionFactory(), new SessionHolder(session))
    ```

  -
    ```
    TransactionSynchronizationManager.bindResource(
          getDataSource(), new ConnectionHolder(connection));
    ```

TransactionAspectSupport

- org.springframework.transaction.interceptor.TransactionAspectSupport
  - public static TransactionStatus currentTransactionStatus() throws NoTransactionException

TransactionAspectSupport

- Useful for MetaClass utility methods:
  - `isTransactionActive()`
    - → `TransactionSynchronizationManager.isSynchronizationActive()`
  - `getCurrentTransactionStatus()`
    - → `TransactionAspectSupport.currentTransactionStatus()`
  - `setRollbackOnly()`
    - → `TransactionAspectSupport.currentTransactionStatus().setRollbackOnly()`
  - `isRollbackOnly()`
    - → `getCurrentTransactionStatus().isRollbackOnly()`

LazyConnectionDataSourceProxy

- org.springframework.jdbc.datasource.LazyConnectionDataSourceProxy
  - **Used by default in Grails 2.3**
  - **Waits to create a connection until it's actually needed**
  - **Caches calls to `setAutoCommit`, `setReadOnly`, `setTransactionIsolation`, etc.**

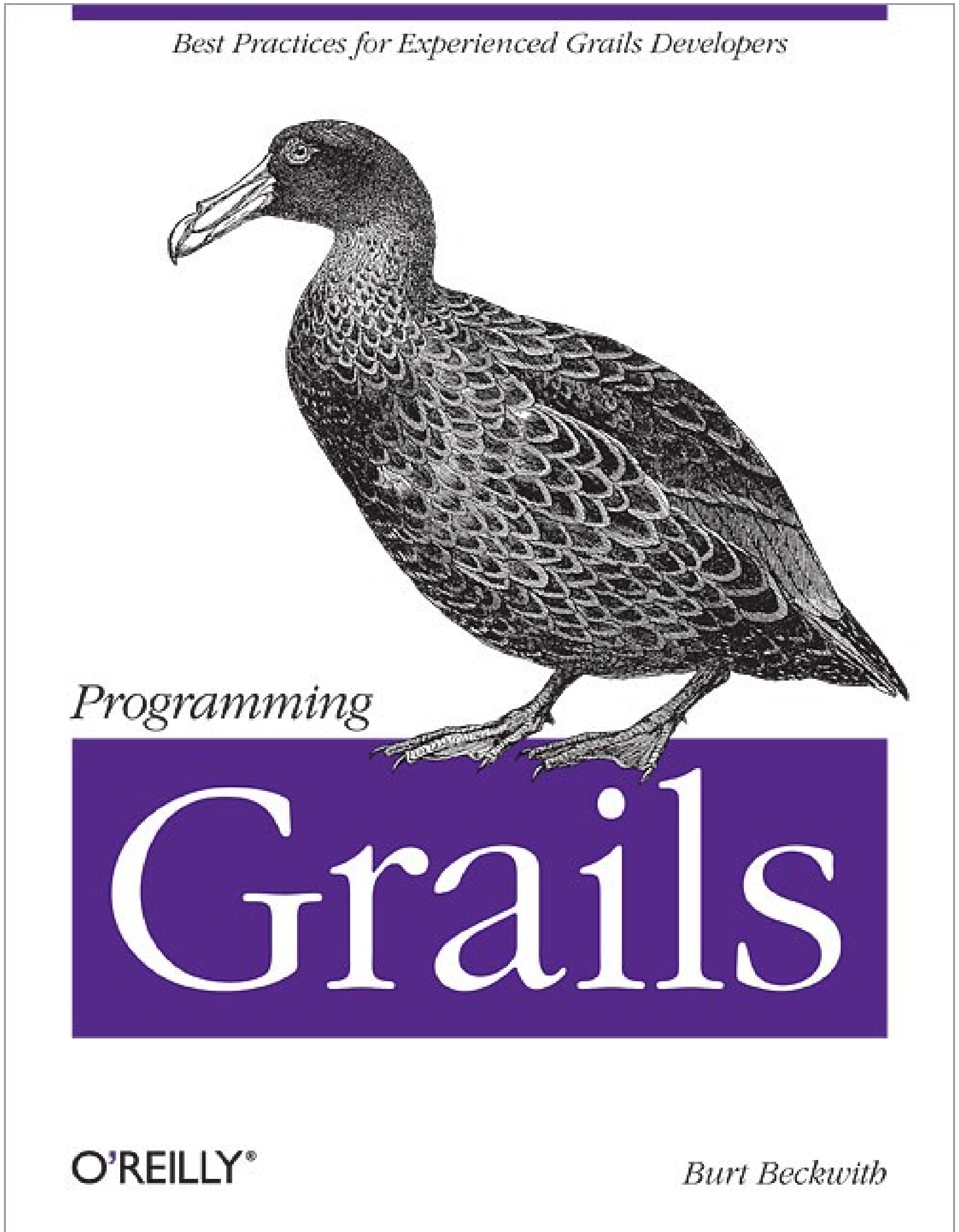
Two-phase Commit (2PC)

Two-phase Commit (2PC)

- The Atomicos plugin is the easiest way
  - **http://grails.org/plugin/atomikos**
  - **http://grails-plugins.github.io/grails-atomikos/docs/manual/index.html**
- **Supports multiple databases and JMS (and any other XA-compliant technology)**

Some Thoughts About Scaffolding

Want To Learn More?

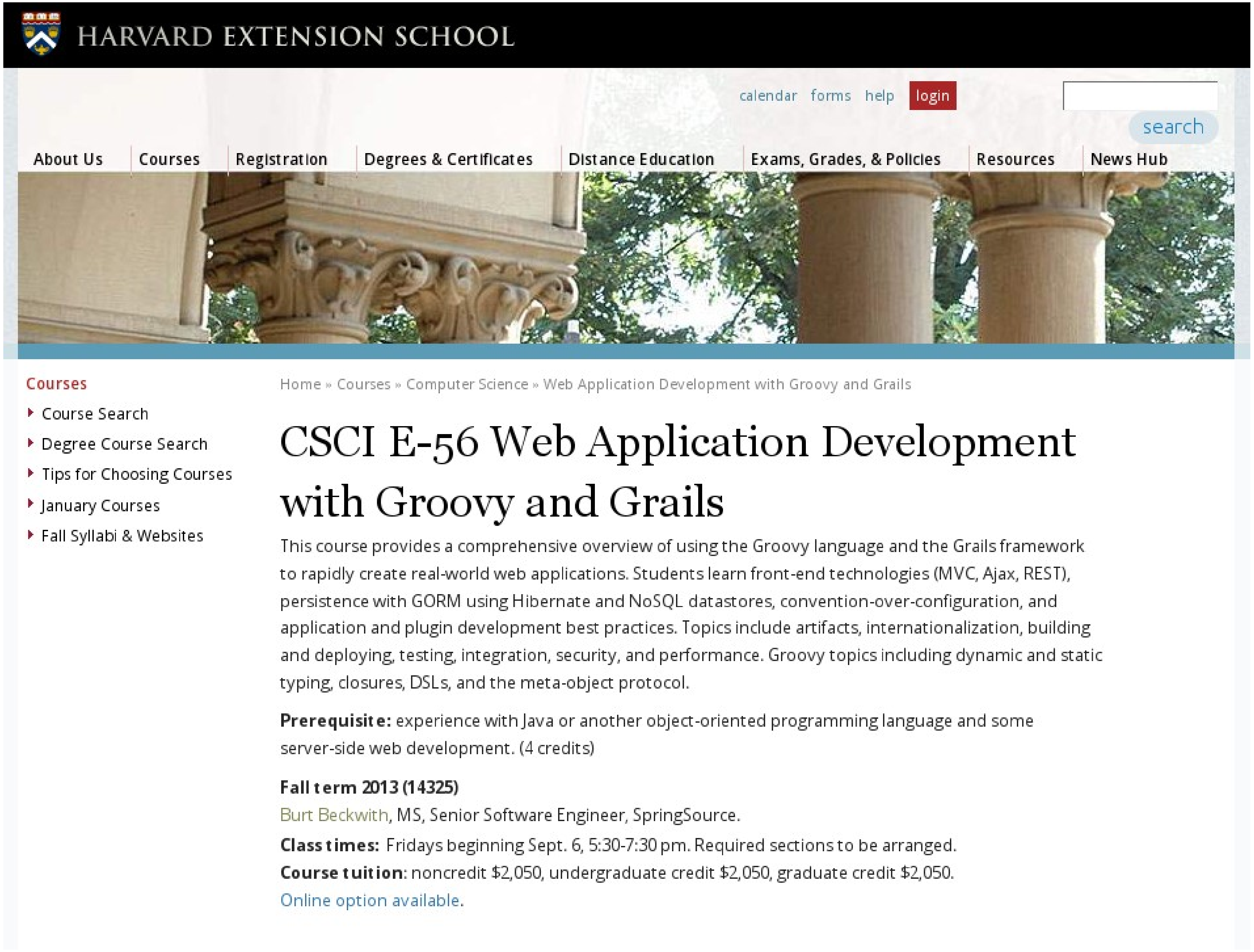

Best Practices for Experienced Grails Developers

Programming

Grails

O'REILLY®

Burt Beckwith



HARVARD EXTENSION SCHOOL

CSCI E-56 Web Application Development with Groovy and Grails

This course provides a comprehensive overview of using the Groovy language and the Grails framework to rapidly create real-world web applications. Students learn front-end technologies (MVC, Ajax, REST), persistence with GORM using Hibernate and NoSQL datastores, convention-over-configuration, and application and plugin development best practices. Topics include artifacts, internationalization, building and deploying, testing, integration, security, and performance. Groovy topics including dynamic and static typing, closures, DSLs, and the meta-object protocol.

Prerequisite: experience with Java or another object-oriented programming language and some server-side web development. (4 credits)

Fall term 2013 (14325)

Burt Beckwith, MS, Senior Software Engineer, SpringSource.

Class times: Fridays beginning Sept. 6, 5:30-7:30 pm. Required sections to be arranged.

Course tuition: noncredit $2,050, undergraduate credit $2,050, graduate credit $2,050.

Online option available.

Thanks!