

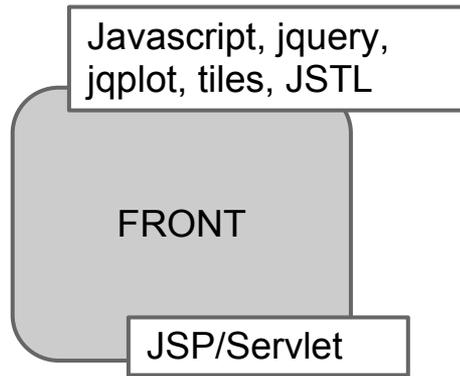
Cerberus design proposal

Principles

Goal : manage Cerberus development

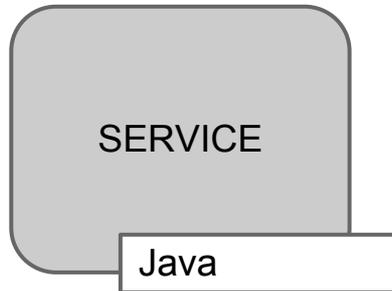
- Use industry standard (no rails or too volatile tools)
- Fix coding guidelines
- Follow coding

Standard transversal layering



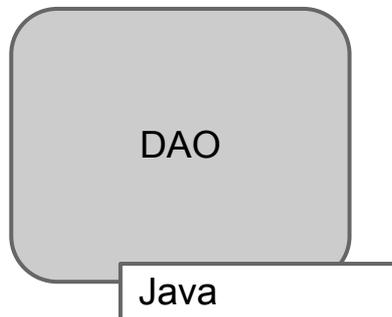
presentation

- jsp and servlet logic
- input control
- controllers that call service



business logic

- core services : testing, launching, CRUD exposition
- called by controller
- call DAO and required utility classes



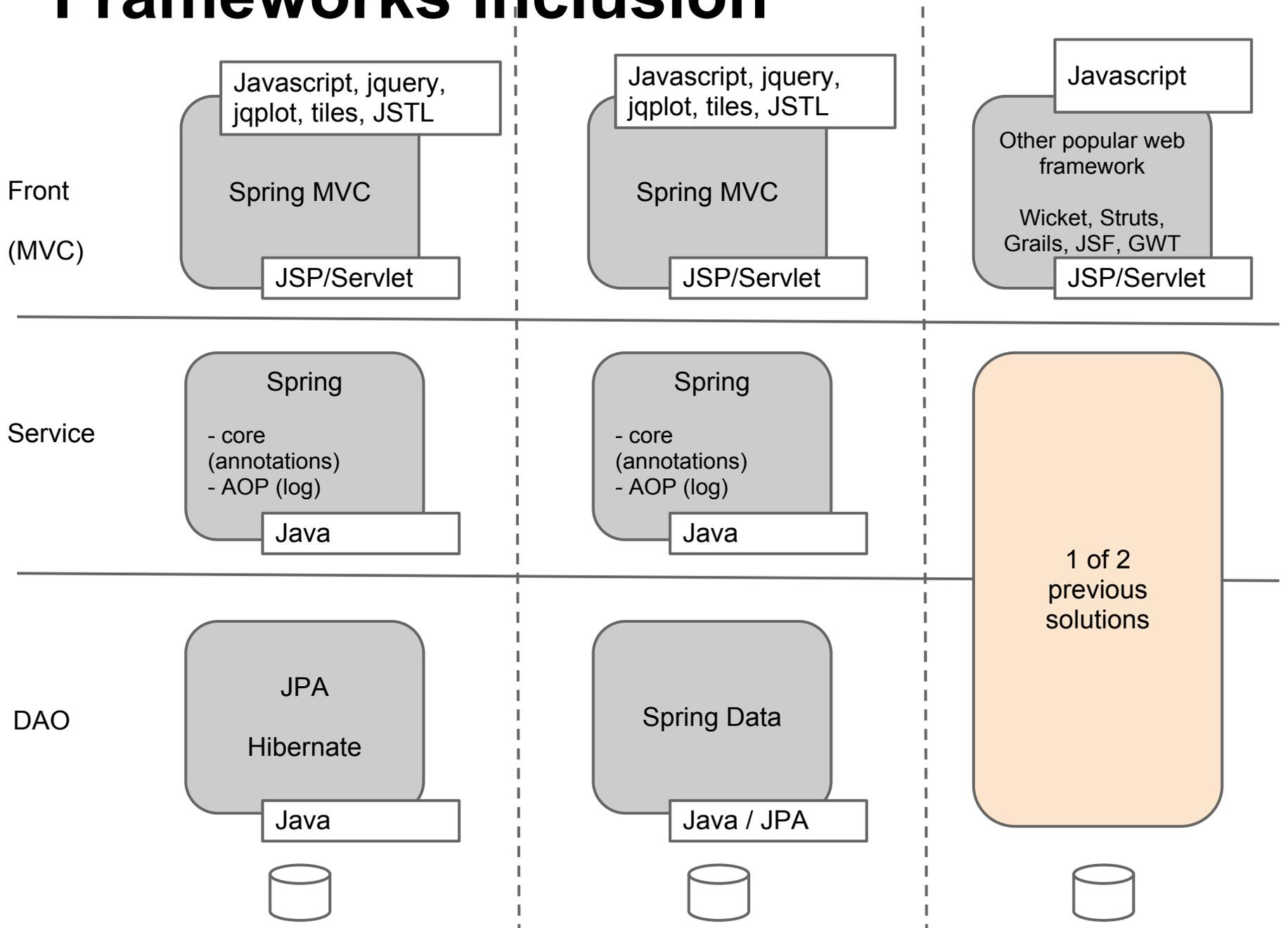
data access

- link between database and Java
- translate tables to Java objects



Basis technology encapsulated

Frameworks inclusion



Frameworks benefits : Front (1/X)

- JQuery : ajax calls, Javascript library
- JQPlot : reporting

- JSTL
 - avoid importing variables in JSP
 - avoid Java `<% %>` in JSP : proper loop, display
 - Java standard

`<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>`

``${testCase.description}`

- Tiles
 - templating : header/footer automated inclusion

Frameworks benefits : Front (2/X)

- Spring MVC
 - proper responsibility concerns between JSP/Java
 - get rid of servlet parameters parsing
 - configuration simplification

@Controller

```
public class HomeController {
```

```
    @RequestMapping("/homepage")
```

```
    public ModelAndView homepage(){
```

```
        // load properties, data to be displayed
```

```
        return new ModelAndView("homepage", model);
```

```
    }
```

```
}
```

Frameworks benefits : Service

- Spring Core

- dependency injection
- transactions management
- configuration with annotations or XML : to define

```
@Repository
public class DAOImpl implements IDAO {
    // code
}
```

```
@Service
public class ServiceImpl
implements IService {
    @Autowired
    private IDAO dao;

    // ... dao.myMethod
    (...);
}
```

```
@Controller //+mapping etc
public class MyController {
    @Autowired
    private IService service;

    // service.myMethod(...);
}
```

- Spring AOP

- transversal concerns centralization
- avoid calling utility classes if service always needed
 - eg : logging of all calls and parameters (Security sanitizer)

Frameworks benefits : DAO (1/X)

- JPA

- Java to physical table mapping & avoid manual transformation between SQL to Java objects
- provide standard CRUD operations
- Reduce dependency with vendor MySQL, Postgresql, ...
- Manage relationships between objects
- Java standard

```
// only jpa imports, no implementation specific
```

```
@Entity
```

```
public class Test implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue
```

```
    private int id;
```

```
    @OneToMany
```

```
    private List<TestCase> testcases;
```

```
}
```

```
// only jpa imports, no implementation specific
```

```
@Entity
```

```
public class TestCase implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue
```

```
    private int id;
```

```
    @ManyToOne
```

```
    private TestCase testcase;
```

```
}
```

Frameworks benefits : DAO (2/X)

- Spring Data
 - simplify data access layer
 - simplify storage solution change
 - framework not yet mature
 - if no spring data required, we can still implement pattern manually I can provide that will provide for any entity classe CRUD operations (findAll, findById, delete, ...)

```
public interface UserRepository extends CrudRepository<User, Long>, UserRepositoryCustom {
```

```
    // specific implementation can still be done
```

```
    User findByTheUserName(String username);
```

```
    @Query("select u from User u where u.firstname = ?1")
```

```
    List<User> findByFirstname(String firstname);
```

```
}
```

Project structuration (1/X)

- standard properties files
- exclude text from jsp in properties files (and enable internationalization)
- separate views / web / ioc config
- database configuration in XML file, not in code

Project structuration (2/X)

pom.xml

=> maven file

Java

src/main/java

=> source code controller, service, dao, etc

src/main/resources

src/main/resources/dbre.xml

=> if we use JPA with xml file

src/main/resources/log4j.properties

=> logging config, rolling file, output, render

src/main/resources/META-INF/persistence.xml

=> hibernate or any JPA compliant configuration

src/main/resources/META-INF/spring/applicationContext.xml

=> global config, wiring of all

src/main/resources/META-INF/spring/database.properties

=> connection pool configuration

src/test/java

=> same package for class under test

src/test/resources

src/test/resources/db.properties

src/test/resources/test-context.xml

src/test/resources/test-db.xml

src/test/resources/log4j.xml

documentation/design

licence.txt

Project structuration (3/X)

Web

src/main/webapp/WEB-INF

src/main/webapp/WEB-INF/web.xml

src/main/webapp/WEB-INF/spring/*.xml

src/main/webapp/WEB-INF/views/views.xml

src/main/webapp/WEB-INF/classes

src/main/webapp/WEB-INF/layouts

src/main/webapp/WEB-INF/messages

src/main/webapp/WEB-INF/views/<structure>/*.jsp

src/main/webapp/resources/styles

src/main/webapp/resources/ images

src/main/webapp/resources/ plugin

webmvc-context.xml
ioc-context.xml, db-context.xml

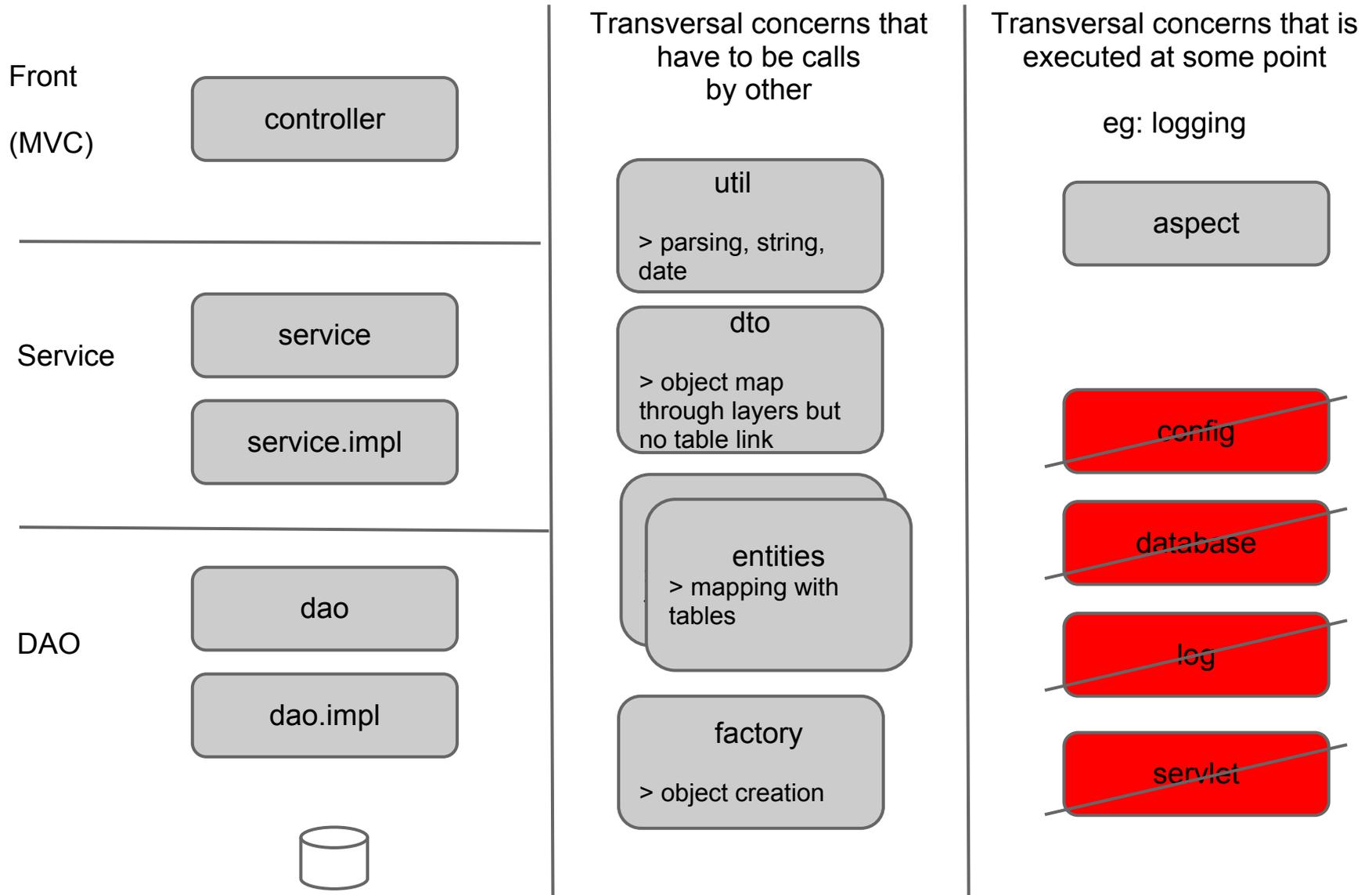
if we want to change css on the fly
tiles for HTML templating, avoid including header etc
messages.properties, messages_fr.properties

css

pictures files

jqplot, jquery folder with sources in

Java packages : com.redcats.tst.



Factory

- encapsulate
 - object creation
 - creation rule
 - only classes with "new" keyword

```
public class FactoryImpl implements IFactory {  
  
    public MyObject create(String name, String description) {  
        MyObject object = new MyObject();  
        object.setName(name);  
        object.setDescription(description);  
        object.setDate(DateUtil.getCurrentDate);  
  
        return object;  
    }  
}
```

Entities

- 1 to 1 mapping with physical table
- Configure relationships
- Only JPA annotations
- Always
 - getters, setters : access encapsulation
 - hashCode and equals : for serialization

```
// only jpa imports
```

```
@Entity
```

```
public class TestCase {
```

```
    @Id
```

```
    private BigInt id;
```

```
    // getters + setters + hashCode + equals
```

```
}
```

DTO

- Data Transfert Objects
- Objects that transit between layers but without physical storage
- Typical usage for reporting objects to build
- Use entity with calculated fields pattern and rule calculated on loading if some fields are not stored in db

```
public class TestCase Reporting {  
  
    private TestCase testcase;  
    private int countOfCreated;  
    private int countOfInProgress;  
  
    // getters + setters + hashCode + equals  
}
```

Util

- Classes calls by other classes for specific treatment
- No instantiation
 - Private constructor & static methods
- Check existing standards for strings, date (Apache)

```
public class DateUtil {  
  
    private DateUtil() {}  
  
    public static Date convertUsingXX() {  
        ...  
    }  
}
```

DateUtil.convertUsingXX(.....)

Aspect

@Component

@Aspect

```
public class LoggerAspect {
```

```
    private final Log log = LoggerFactory.getLog(this.getClass());
```

```
    @Around("execution(* com.foo.bar..*.*(..))")
```

```
    public Object logTimeMethod(ProceedingJoinPoint joinPoint) throws Throwable {
```

```
        // logging implementation, start, end
```

```
        // then log4j config is used to write to table, service, call url, have a rolling
```

```
file ...
```

```
    }
```

```
}
```

Controller

can easily standardize urls

*.configuration

*.execution

*.reporting

@Controller

```
public class HomeController {
```

```
    @RequestMapping("/homepage")
```

```
    public ModelAndView homepage(){
```

```
        // load properties, data to be displayed
```

```
        return new ModelAndView("homepage", model);
```

```
    }
```

```
}
```

Code convention

- naming
 - interfaces starts with "I"
 - no visibility on interface methods
 - implementation
 - in /impl directory
 - if 2 implementation, explicit name
- dependencies with interfaces, not implementation
- javadoc on all interfaces

Testing

Test typology

Front
(MVC)

Selenium
integrated

no cerberus
directly

=> Integration testing
=> Functional testing

Service

JUnit

Mockito

=> Unit testing

DAO

JUnit

Spring context

=> DB Integration testing
=> Unit testing

Test convention

- Same package as tested class
- naming
 - same as tested class
 - prefix by "Test"

src/main/java/com/redcats/tst/dao/ITestCaseDAO

src/main/java/com/redcats/tst/dao/impl/**TestCaseDAOImpl**

src/**test**/java/com/redcats/tst/dao/impl/**Test****TestCaseDAOImpl**

Test example : DAO

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"/test-context.xml"})
public class ITestCaseDAOTest extends AbstractTransactionalJUnit4SpringContextTests {

    @Autowired
    private ITestCaseDAO dao;

    @Test
    public void testGetThisResultInThisCondition() {
        // GIVEN
        int id = ..

        // WHEN
        dao.find/remove/..

        // THEN
        assert(..)
    }
}
```

Test example : Service

```
@RunWith(MockitoJUnitRunner.class)
```

```
public class TestCaseServiceImplTest {
```

```
    private TestCaseServiceImpl testCasetServiceImpl;
```

```
    @Mock private Test test;
```

```
    @Test
```

```
    public void testXXX() {
```

```
        // given
```

```
        // when
```

```
        // then
```

```
    }
```

```
}
```

Build

goal

- repeatable build in any environment
- easier test configuration

today issues

- no tests, part due to lack of proper build in place
- no repeatable build

from maven test

1. setup in-memory database or still use local
2. recreate database structure / truncate tables
3. execute sql script
4. execute unit test
5. execute sql script
6. execute integration test
7. package
8. to decide how to manage changes in UAT/prod

Detailed frameworks versions

- Front
 - Spring MVC 3.2.3 Release
- Service
 - Spring 3.2.3 Release
- DAO
 - Hibernate 3.2

can use spring roo to generate project structure or at least get all dependencies

Migration plan proposal

1. Get to one project : services + GUI
2. Standardize
3. Progressive framework integration
 - Spring IOC
 - Spring MVC
 - Spring Data / Hibernate

Migration plan : gap analysis

Get to one project : services + GUI

- integrate GUI files progressively, commit regularly
- remove old SVN files
- migrate all developers connectivity
- remove unused jenkins jobs
- remove unused packages on glassfish

Migration plan : gap analysis

Standardize

- code review priority 1 fix - main :
 - externalize database configuration
- properties files externalization
- enable repeatable build
 - sql script or unit test data in each class
 - selenium test for integration testing
- fix unit test DAO integration
- fix unit test service in isolation

Migration plan : gap analysis

Progressive framework integration

- Spring IOC - already there but need to
 - organize property files like specified
 - standardize naming conventions
- Spring Data / Hibernate
 - configuration file
 - implement DAO pattern for all CRUD
 - remove unnecessary code replaced by patterns
- Spring MVC
 - setup configuration files
 - setup Tiles template header/footer
 - progressively migrate JSP with integration test on GUI (with JSTL & Tiles)

Decisions to take

- Frameworks stack
- Migration plan
- Next actions