Soirée
Emmanuel Bernard

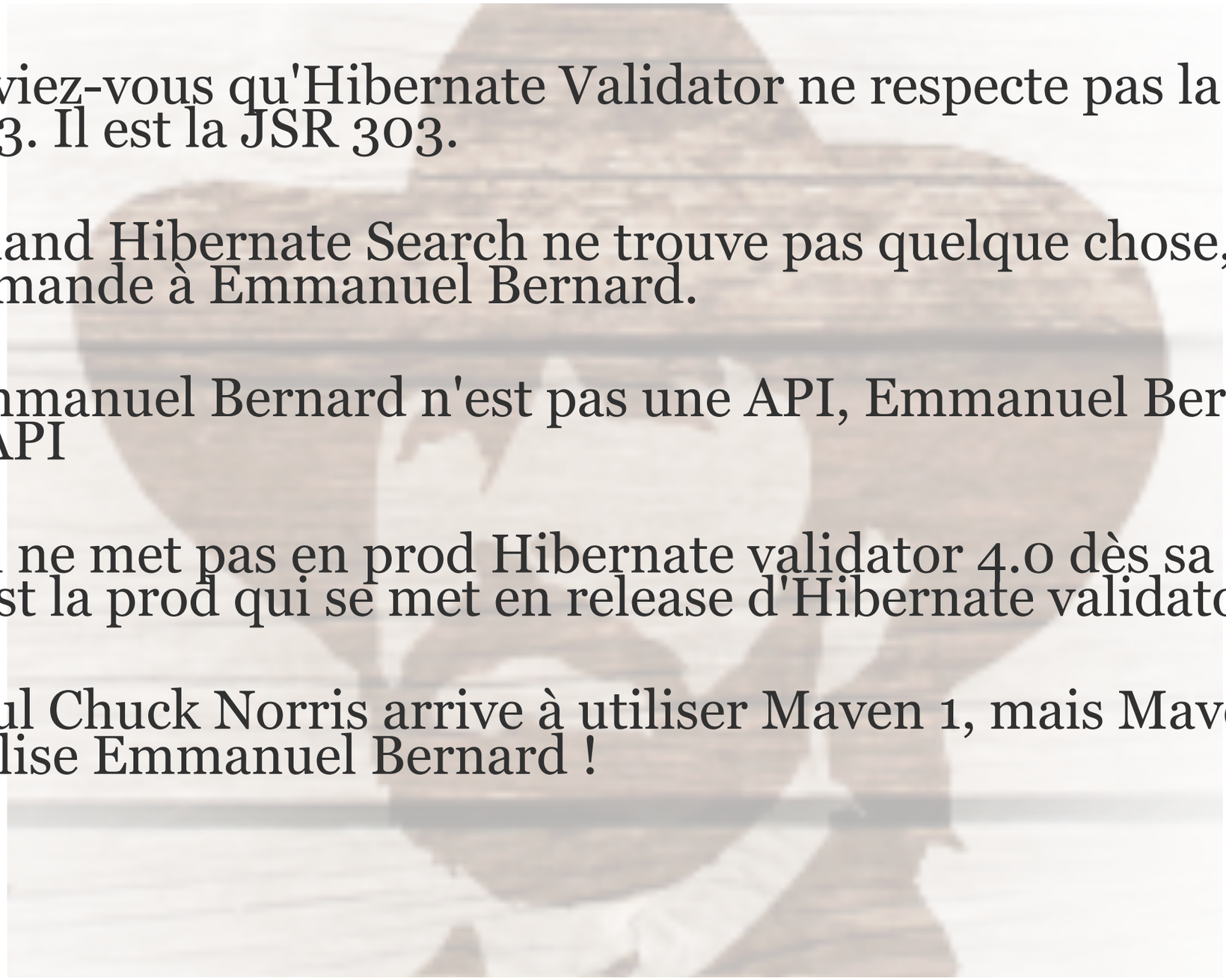
Reactions sur
La soirée Emmanuel Bernard

Paris, Paris, oooooon t'enc*le!
Julien V. - Mars JUG

La soirée must-be-seen.
Clara M. - JDuchess France

Bon sang mais t'es une porn-star!
Je suis fier de toi.
Marc F. - Retraité

Emmanuel Bernard

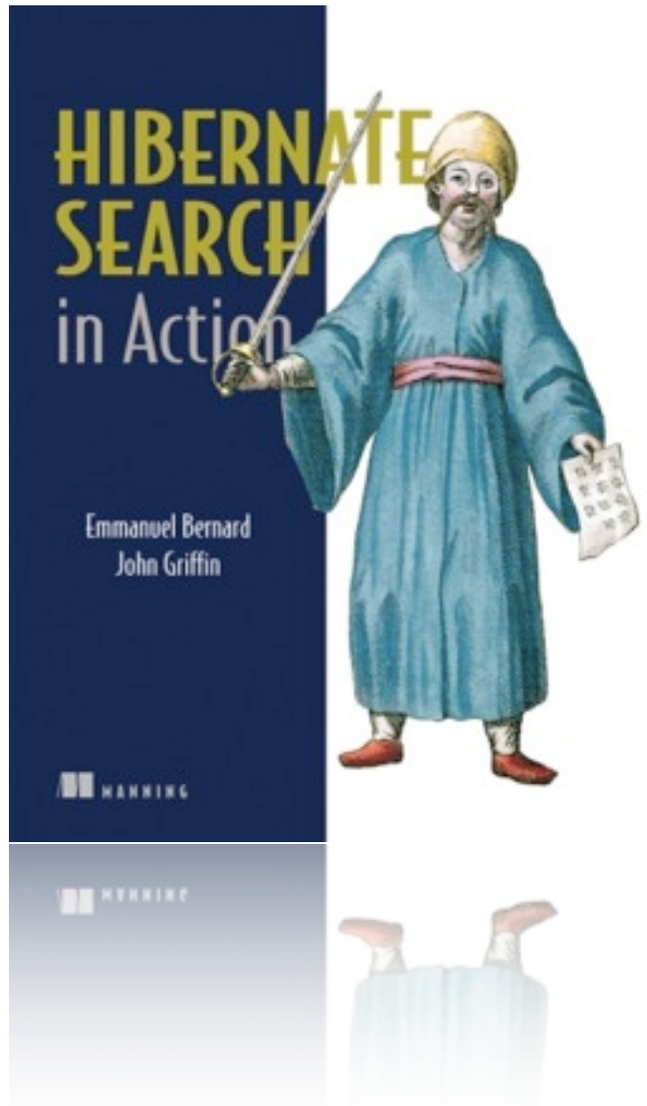


- Saviez-vous qu'Hibernate Validator ne respecte pas la JSR 303. Il est la JSR 303.
- Quand Hibernate Search ne trouve pas quelque chose, il demande à Emmanuel Bernard.
- Emmanuel Bernard n'est pas une API, Emmanuel Bernard est L'API
- On ne met pas en prod Hibernate validator 4.0 dès sa release, c'est la prod qui se met en release d'Hibernate validator 4.0.
- Seul Chuck Norris arrive à utiliser Maven 1, mais Maven 2 utilise Emmanuel Bernard !

Emmanuel Bernard

Le Vrai

- Hibernate Search in Action
- blog.emmanuelbernard.com
- twitter.com/emmanuelbernard
- lescastcodeurs.com



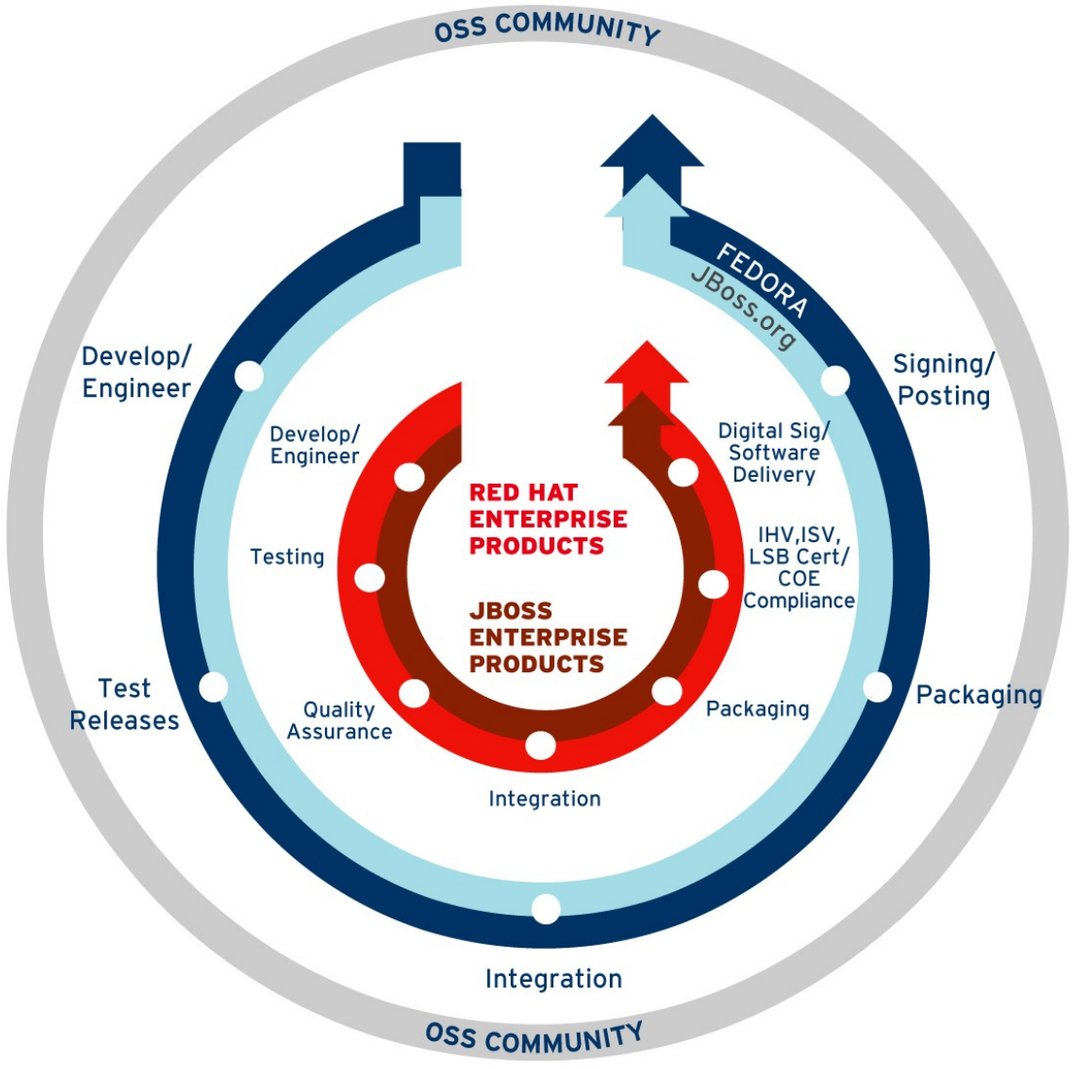
Les CastCodeurs

"En Français dans le Code"

http://lescastcodeurs.com

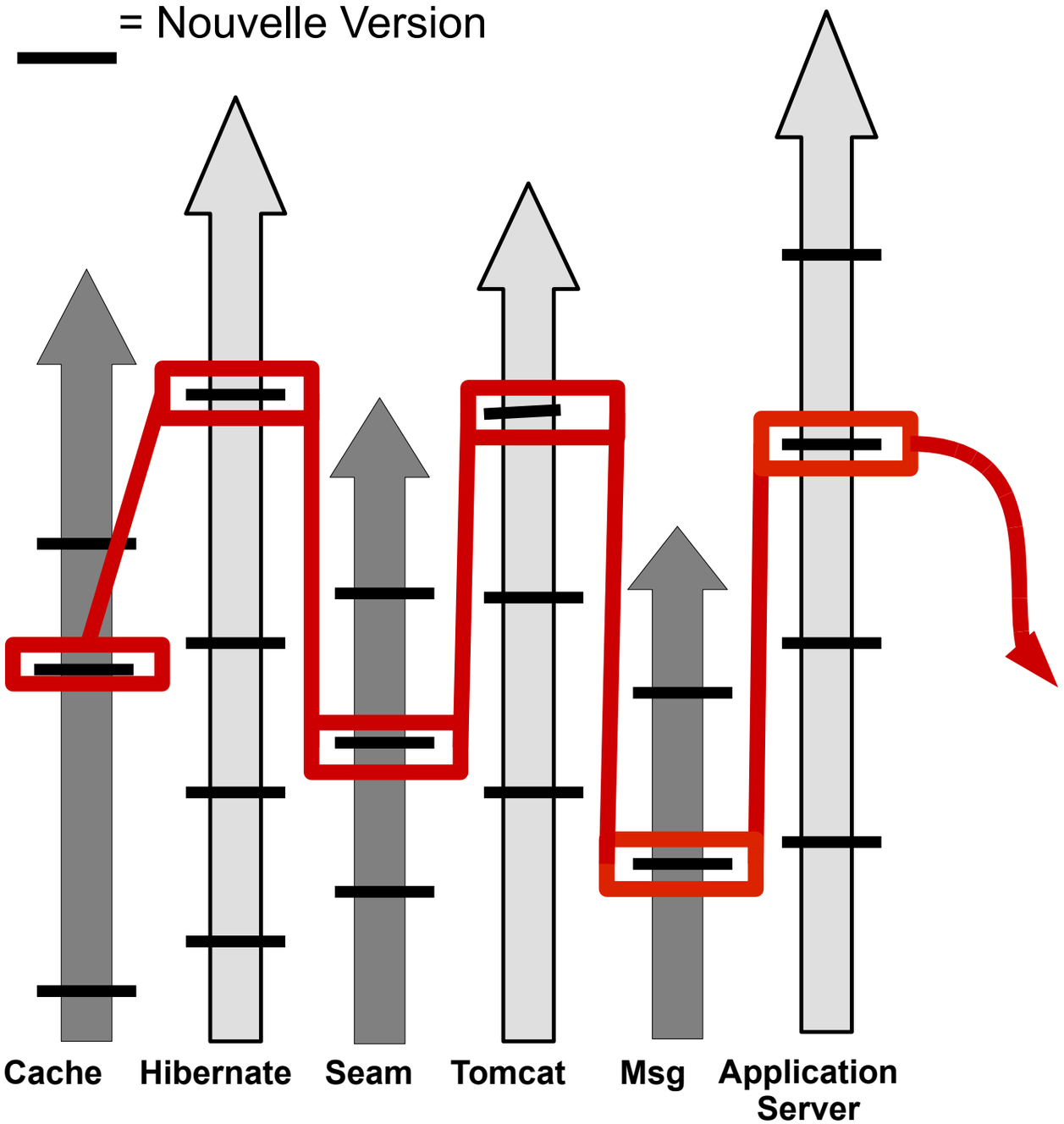
Red Hat et l'intégration Open Source

- **Collaboration avec la communauté pour le développement**
- **Deux distributions complètes, pour deux usages différents**
  - **Fedora / JBoss.org**
    - Vitrine/Laboratoire Technologique et d'Innovations
    - Version majeure tous les 6 mois,
    - Non supporté,
    - Dernières technologies disponibles.
  - **Red Hat Enterprise & JBoss Enterprise**
    - Version Entreprise
    - Version majeure tous les 2 ans,
    - Certifié, Supporté avec SLAs
    - Mature et stable
    - Prêt pour l'utilisation en entreprise
    - Large écosystème en croissance



<nu

JBoss Enterprise : la solution Platform



Projets mutliples, avec des calendriers de sortie différents, des différences de versions, de dépendances, etc.

- **Défi :**
  - Intégrer et maintenir l'intégration de plusieurs projets en une plate-forme satisfaisant les besoins de l'Entreprise
  - Coûteux, en maintenance et en temps
- **Solution : JBoss Enterprise Platforms**
  - Distributions intégrées et certifiées
  - Processus intensif d'assurance qualité
  - Support de qualité reconnu comme tel
  - Documentation
  - Configurations par défaut sécurisées, qualité production
  - Maintenance de 7 ans

<nu

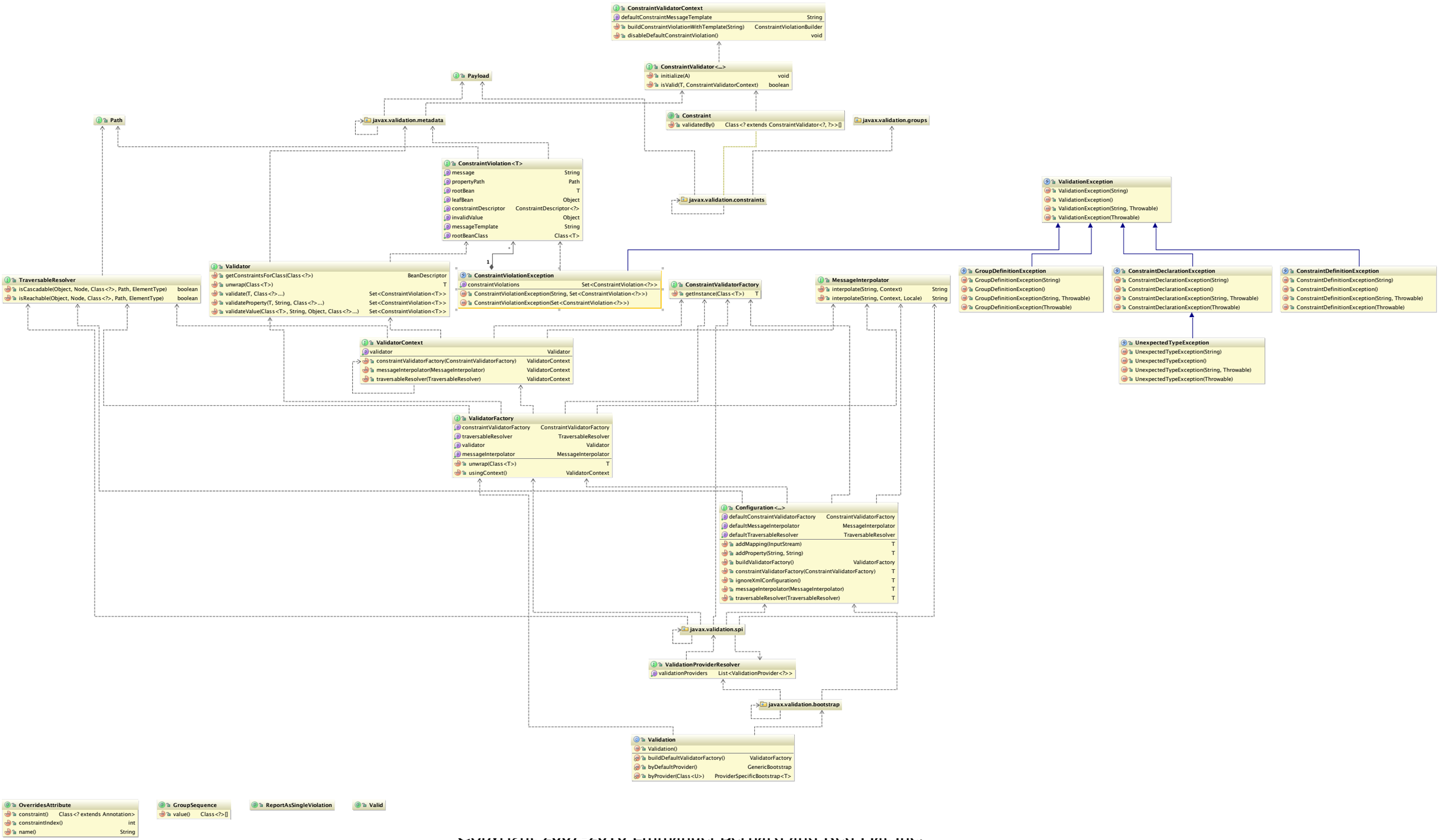
API design

ups and downs

API - why spend time?

- Very time consuming
- Nobody ends up 100% happy
- Once 1.0 it's for EVER
- Save time down the road

Is this a good API?




ConstraintValidatorFactory
getInstance(Class<T>) T
TraversableResolver
isCascadable(Object, Node, Class<?>, Path, ElementType) boolean
isReachable(Object, Node, Class<?>, Path, ElementType) boolean
MessageInterpolator
interpolate(String, Context) String
interpolate(String, Context, Locale) String
ValidatorFactory
constraintValidatorFactory ConstraintValidatorFactory
traversableResolver TraversableResolver
validator Validator
messageInterpolator MessageInterpolator
unwrap(Class<T>) T
usingContext() ValidatorContext
ConfigurationState
ignoreXmlConfiguration boolean
constraintValidatorFactory ConstraintValidatorFactory
traversableResolver TraversableResolver
mappingStreams Set<InputStream>
messageInterpolator MessageInterpolator
properties Map<String, String>
Configuration<...>
defaultConstraintValidatorFactory ConstraintValidatorFactory
defaultMessageInterpolator MessageInterpolator
defaultTraversableResolver TraversableResolver
addMapping(InputStream) T
addProperty(String, String) T
buildValidatorFactory() ValidatorFactory
constraintValidatorFactory(ConstraintValidatorFactory) T
ignoreXmlConfiguration() T
messageInterpolator(MessageInterpolator) T
traversableResolver(TraversableResolver) T
BootstrapState
validationProviderResolver ValidationProviderResolver
defaultValidationProviderResolver ValidationProviderResolver
ValidationProvider<...>
buildValidatorFactory(ConfigurationState) ValidatorFactory
createGenericConfiguration(BootstrapState) Configuration<?>
createSpecializedConfiguration(BootstrapState) T
ValidationProviderResolver
validationProviders List<ValidationProvider<?>>
GenericBootstrap
configure() Configuration<?>
providerResolver(ValidationProviderResolver) GenericBootstrap
ProviderSpecificBootstrap<...>
configure() T
providerResolver(ValidationProviderResolver) ProviderSpecificBootstrap<T>
Validation
buildDefaultValidatorFactory() ValidatorFactory
byDefaultProvider() GenericBootstrap
byProvider(Class<U>) ProviderSpecificBootstrap<T>

API - why spend time?

- It's for humans, stupid
- Usable by humans
  - IDE / type-safety
  - # of methods
  - from beginners to experts
- Readable by humans
- Docs are no substitute

Type-safety love affair

- The language prevents mistakes
- Unit testing?
- Help from any auto-complete aware IDE
- Built-in documentation and error descriptor

API and me

- Heavy consumer (like all of us)
- Producer
  - internal
  - R&D projects
  - projects
  - specification

API Design - rules

- #0 If you don't fall, you don't make progress

  - Snowboard saying

- #1 When in doubt, leave it out

Hibernate Annotations

- Annotations
  - new paradigm / playground

- Binder
  - most horrible design publicly displayed
  - very procedural
  - the OO design failed too :)

Hibernate Validator 3

- Mantra
  - keep things simple

- Grew organically
  - became hairy
  - semantic inconsistencies

Hibernate Search

- Much better
- Had to extend existing APIs

- Keep things simple
- Prepare for the future
  - Extension points for power user

Bean Validation

- Specification
- Compromise
- Less is more
- APIs are For Ever

My best failures: nested annotations

```
class @interface OneToMany {
  JoinColumn[] joinColumns() default {};
  JoinTable joinTable() default {};
}
```

```
@OneToMany(
  joinTable=@JoinTable(name=”HOME”,
     joinColumns = {@JoinColumn(name=”user_id”)}
  )
)
Address home;
```

My best failures: Technical inheritance

- Use inheritance for technical extension
  - `AnnotationConfiguration extends Configuration`
  - What to share, what not to share?
  - Myriad of micro method for code reuse and lifecycle hooks
  - hard to predict the future

My best failures: Delegation

- Better approach overall
- `FullTextSession implements Session`
- Duplicates some parts still
  - Helper methods, nested delegation
- Ejb3Configuration is still not great

Good practices

- Use your APIs when designing them
  - TDD is not enough
- Think beyond the code
  - semantic
- Make DSLs
- Make the most used case easy
- Read “Effective Java” by Joshua Bloch

DSL what?

- A big word for readable code
  - internal DSL
  - Reads like English
  - names short and explicit

```
mapping
 .fullTextFilterDef("security", SecurityFilterFactory.class)
   .cache(FilterCacheModeType.INSTANCE_ONLY)

 .analyzerDef( "ngram", StandardTokenizerFactory.class )
   .filter( LowerCaseFilterFactory.class )
   .filter( NGramFilterFactory.class )
     .param( "minGramSize", "3" )
     .param( "maxGramSize", "3" )

 .entity( Address.class )
   .indexed()
   .property("addressId", FIELD)
     .documentId().name( "id" )
 .property("street1", FIELD)
   .field()
   .field().analyzer("ngram").store(Store.YES)
 .property(“zipcode”, FIELD)
   .field()
     .bridge( ZipCodeBridge.class )
       .param( ZipCode.DEPT, 2 );
```

SearchMapping

```
mapping
 .fullTextFilterDef("security", SecurityFilterFactory.class)
   .cache(FilterCacheModeType.INSTANCE_ONLY)

 .analyzerDef( "ngram", StandardTokenizerFactory.class )
   .filter( LowerCaseFilterFactory.class )
   .filter( NGramFilterFactory.class )
     .param( "minGramSize", "3" )
     .param( "maxGramSize", "3" )

.entity( Address.class )
  .indexed()
  .property("addressId", FIELD)
    .documentId().name( "id" )
  .property("street1", FIELD)
    .field()
    .field().analyzer("ngram").store(Store.YES)
  .property(“zipcode”, FIELD)
    .field()
      .bridge( ZipCodeBridge.class )
        .param( ZipCode.DEPT, 2 );
```

EntityMapping

```
mapping
  .fullTextFilterDef("security", SecurityFilterFactory.class)
    .cache(FilterCacheModeType.INSTANCE_ONLY)

  .analyzerDef( "ngram", StandardTokenizerFactory.class )
    .filter( LowerCaseFilterFactory.class )
    .filter( NGramFilterFactory.class )
      .param( "minGramSize", "3" )
      .param( "maxGramSize", "3" )

  .entity( Address.class )
.indexed()
.property("addressId", FIELD)
  .documentId().name( "id" )
.property("street1", FIELD)
  .field()
  .field().analyzer("ngram").store(Store.YES)
.property(“zipcode”, FIELD)
  .field()
    .bridge( ZipCodeBridge.class )
      .param( ZipCode.DEPT, 2 );
```

Power without clutter

- Sensitive defaults
- Overridable
  - without blowing the API

```
Validator v =
  vf.getValidator();

Validator v =
  vf.usingContext()
      .messageInterpolator(jsfMI)
      .traversableResolver(AconcalTR)
      .getValidator();
```

```
interface ValidatorContext {
  ValidatorContext msgInterpolator(MsgInterp mi);
  ...
  Validator getValidator();
}
interface ValidatorFactory {
  Validator getValidator();
  ValidatorContext usingContext();
}
```

Power without clutter
Contract edition

- The contract might evolve
- Breaking the contract should be avoided

```
public interface ConstraintValidator<A extends Annotation, T> {
    void initialize(A constraintAnnotation);

    boolean isValid(T value, ConstraintValidatorContext context);
}

public interface ConstraintValidatorContext {
    void disableDefaultConstraintViolation();
    String getDefaultConstraintMessageTemplate();
    ConstraintViolationBuilder buildConstraintViolationWithTemplate
(String messageTemplate);
    interface ConstraintViolationBuilder {
        NodeBuilderDefinedContext addNode(String name);
        ConstraintValidatorContext addConstraintViolation();
        interface NodeBuilderDefinedContext {
            NodeBuilderCustomizableContext addNode(String name);
            ConstraintValidatorContext addConstraintViolation();
        }
        interface NodeBuilderCustomizableContext {
            NodeContextBuilder inIterable();
            NodeBuilderCustomizableContext addNode(String name);
            ConstraintValidatorContext addConstraintViolation();
        }
        interface NodeContextBuilder {
            NodeBuilderDefinedContext atKey(Object key);
            NodeBuilderDefinedContext atIndex(Integer index);
            NodeBuilderCustomizableContext addNode(String name);
            ConstraintValidatorContext addConstraintViolation();
        }
    }
}
```

Power without clutter extend and specialize

- Provide more without polluting the simple API
- `<T extends Base> T specializes(Class<T> type) {}`


ConstraintValidatorFactory
getInstance(Class<T>) T
TraversableResolver
isCascadable(Object, Node, Class<?>, Path, ElementType) boolean
isReachable(Object, Node, Class<?>, Path, ElementType) boolean
MessageInterpolator
interpolate(String, Context) String
interpolate(String, Context, Locale) String
ValidatorFactory
constraintValidatorFactory ConstraintValidatorFactory
traversableResolver TraversableResolver
validator Validator
messageInterpolator MessageInterpolator
unwrap(Class<T>) T
usingContext() ValidatorContext
ConfigurationState
ignoreXmlConfiguration boolean
constraintValidatorFactory ConstraintValidatorFactory
traversableResolver TraversableResolver
mappingStreams Set<InputStream>
messageInterpolator MessageInterpolator
properties Map<String, String>
Configuration<...>
defaultConstraintValidatorFactory ConstraintValidatorFactory
defaultMessageInterpolator MessageInterpolator
defaultTraversableResolver TraversableResolver
addMapping(InputStream) T
addProperty(String, String) T
buildValidatorFactory() ValidatorFactory
constraintValidatorFactory(ConstraintValidatorFactory) T
ignoreXmlConfiguration() T
messageInterpolator(MessageInterpolator) T
traversableResolver(TraversableResolver) T
BootstrapState
validationProviderResolver ValidationProviderResolver
defaultValidationProviderResolver ValidationProviderResolver
ValidationProvider<...>
buildValidatorFactory(ConfigurationState) ValidatorFactory
createGenericConfiguration(BootstrapState) Configuration<?>
createSpecializedConfiguration(BootstrapState) T
ValidationProviderResolver
validationProviders List<ValidationProvider<?>>
GenericBootstrap
configure() Configuration<?>
providerResolver(ValidationProviderResolver) GenericBootstrap
ProviderSpecificBootstrap<...>
configure() T
providerResolver(ValidationProviderResolver) ProviderSpecificBootstrap<T>
Validation
buildDefaultValidatorFactory() ValidatorFactory
byDefaultProvider() GenericBootstrap
byProvider(Class<U>) ProviderSpecificBootstrap<T>

```
public interface Configuration<T extends Configuration<T>> {
    T ignoreXmlConfiguration();
    T messageInterpolator(MessageInterpolator interpolator);
    T traversableResolver(TraversableResolver resolver);
    T constraintValidatorFactory(ConstraintValidatorFactory
constraintValidatorFactory);
    T addMapping(InputStream stream);
    T addProperty(String name, String value);
    MessageInterpolator getDefaultMessageInterpolator();
    TraversableResolver getDefaultTraversableResolver();
    ConstraintValidatorFactory getDefaultConstraintValidatorFactory();
    ValidatorFactory buildValidatorFactory();
}
```

```
public interface HVConf extends Configuration<HVConf> {
    HVConf enableLegacyAnnotations();
    HVConf addConstraint(ConstraintDescriptor constraint);
}
```

```
ValidatorFactory factory = Validation
  .byProvider(HibernateValidator.class)
  .configure()
    .ignoreXmlConfiguration()
    .enableLegacyAnnotations()
    .addConstraint( myHVConstraint )
    .buildValidatorFatory();


public interface ValidationProvider<T extends Configuration<T>>
  { ... }
```

```
public class HibernateValidator implements
  ValidationProvider<HVConf> { ... }
```

Other tips

- Abstract class vs interface
  - API or extension contract
- Enums and interfaces

.close()

- buildAndTry() { return buildAndTry(); }
- Think about the user
- Think about the future
- Utilise Emmanuel Bernard

Questions

- http://in.relation.to
- http://blog.emmanuelbernard.com