Upgrading Java (11→21+) is Easy; Upgrading Spring (4→6) and Hibernate (5→6) is NOT

Marek Dominiak

Marek Dominiak

I run training in EventStorming, Architecture, DDD, and TDD

18 years of professional experience

Co-owner and instructor at trainitek

Hands-on Architect and Team Lead at MINTRA

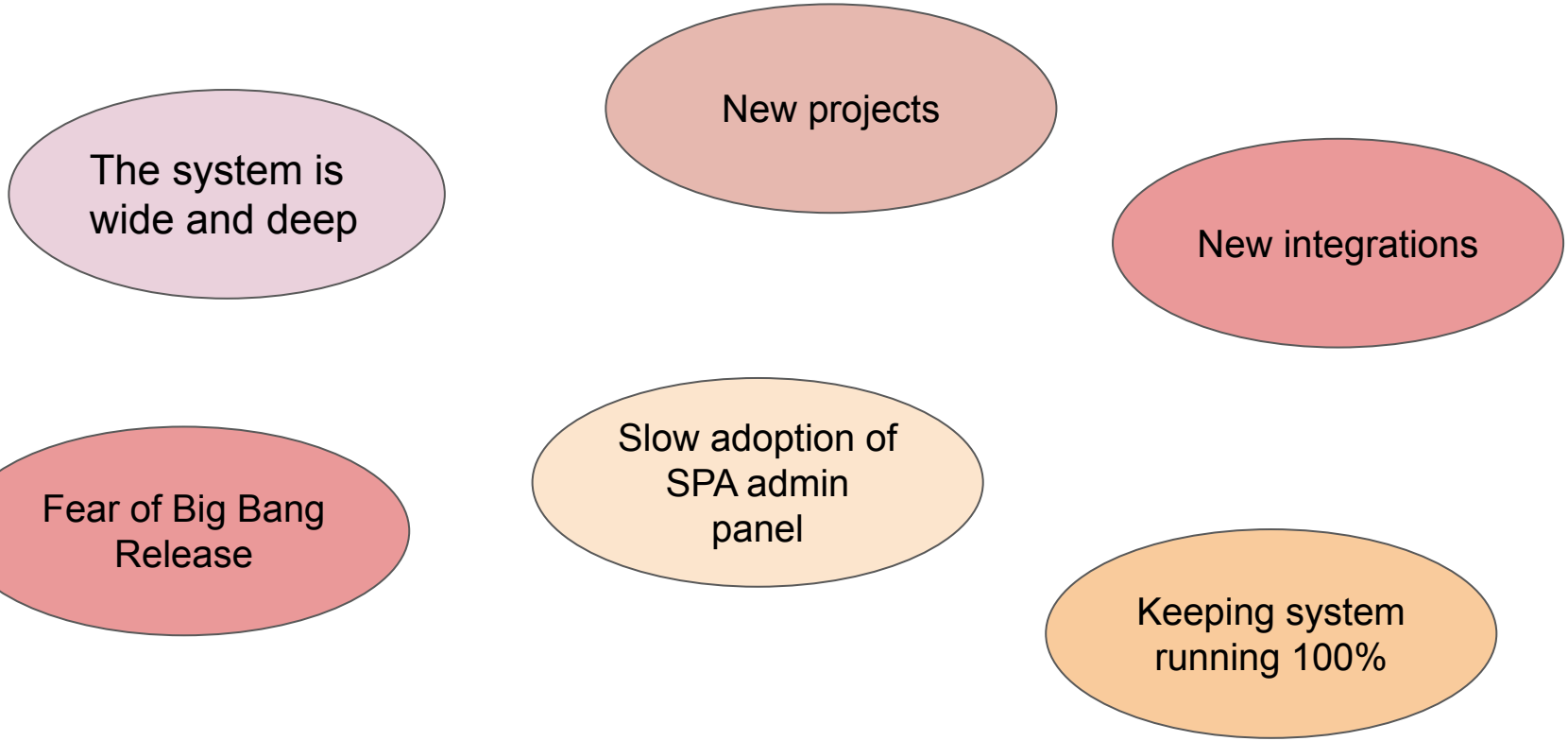
https://www.linkedin.com/in/marekdominiak

Agenda

- Context of the migration
- Plan for migration (Theory and Practice)
- Easy and Hard Migrations
- Benefits
- Lessons learned
- Q/A

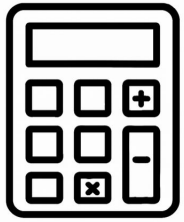
Context of the migration

What did hold us back?



Numbers

- The system developed over the last 16 years
- Team size ~25-30 people during this time
- ~13 different services
- Number of files: ~64K
- Lines of code: > 2M
- Number of user accounts: > 4.2M



Why did we need to upgrade?

- Bad Developer Experience

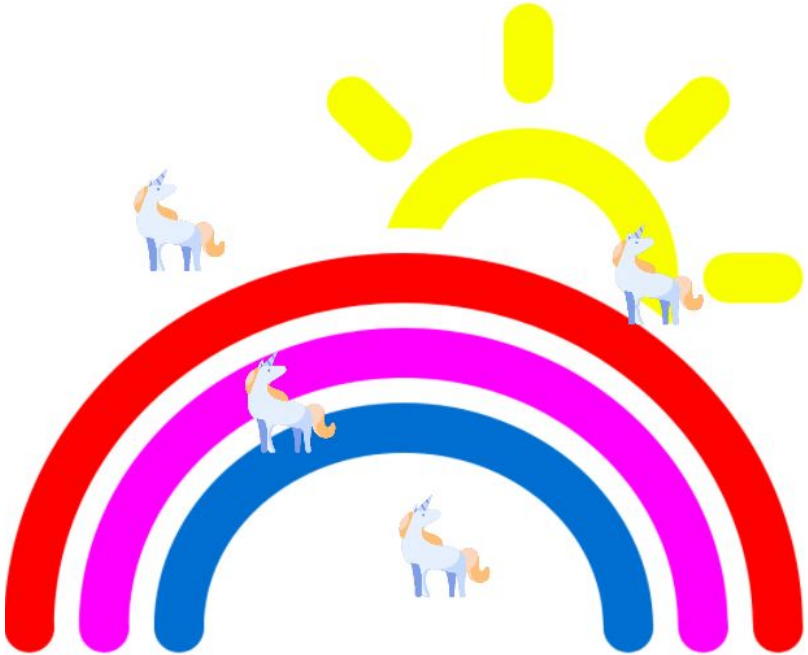
- Increased cost of keeping things secure

- Benefits from JVM upgrade

- Costs related to running outdated versions on Cloud (e.g. Mysql on AWS)

Upgrade: In Theory

In Theory



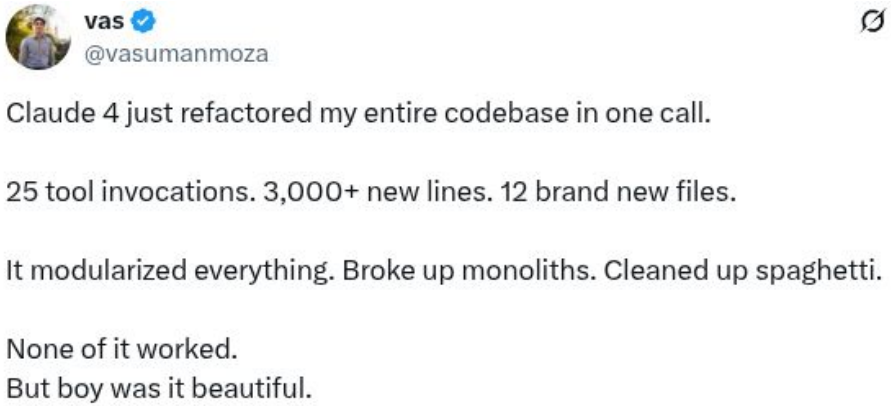
Ref: https://www.linkedin.com/posts/vladlarichev_genai-amazonq-generativeai-activity-7233767814497472512-ldh0

What's the difference between Theory and Practice?

In theory, there is no difference - but in practice, there is.

Theory vs Practice

- In just a few minutes/hours we should be done.
- Too good to be true?
- Yes, it is :)



- Using AI for our upgrade would lead to more time spent than doing it by ourselves.

Ref: https://x.com/vasumanmoza/status/1926487201463832863
Ref 2: https://www.reddit.com/r/aws/comments/1jhijd9/aws_q_was_great_untill_it_started_lying/
Ref 3: https://github.com/spring-projects/spring-boot/pull/39754/
Ref 4: https://medium.com/@sobyx/the-ais-existential-crisis-an-unexpected-journey-with-cursor-and-gemini-2-5-pro-7dd811ba7e5e

Upgrade: In Practice

Parts where migration was easy

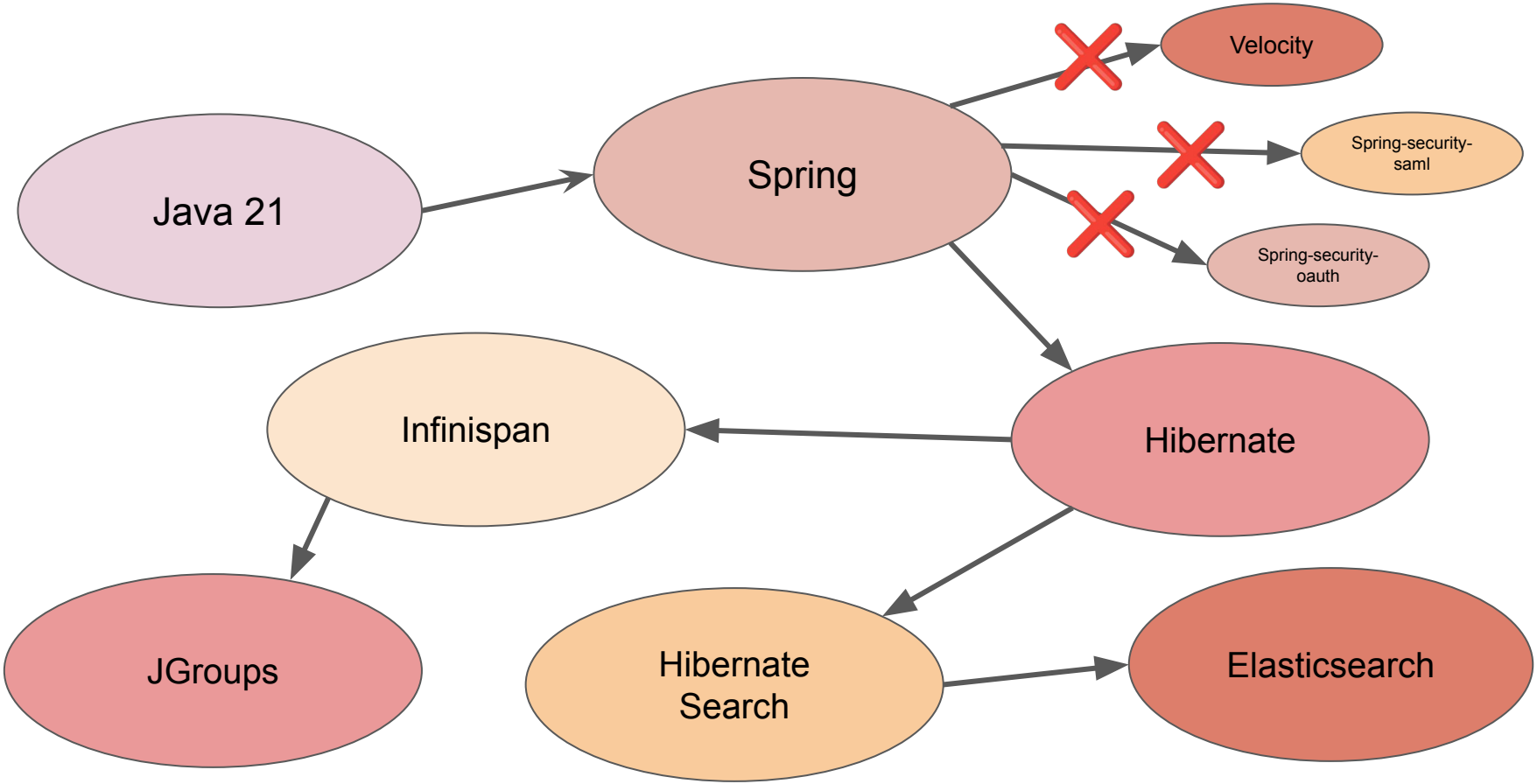
“Easy” upgrades

- Java 11 to Java 21
- Tomcat 9 to Tomcat 11
- Spring Framework (Core, MVC) 4.3.x to 6.x

Parts where migration was NOT easy

Challenge 1 - Chained Dependencies

Chained Dependencies

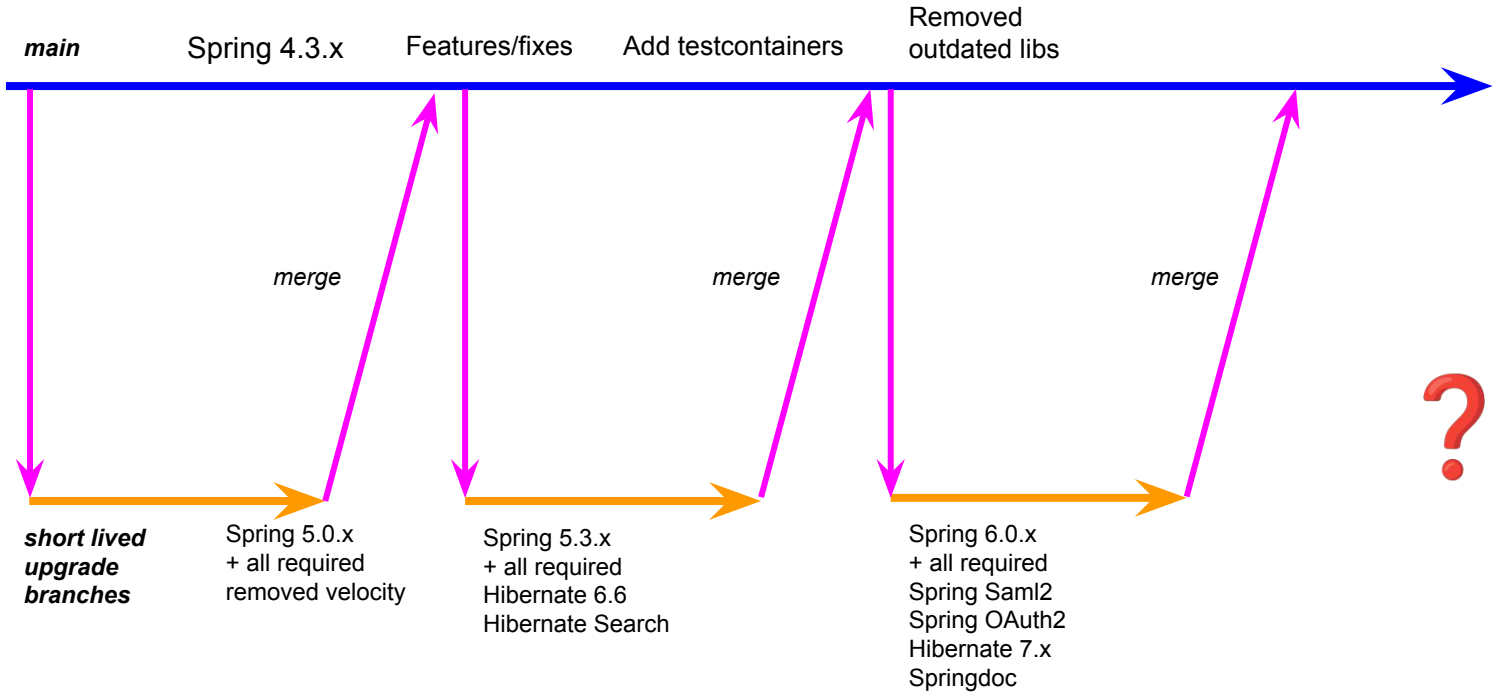


What were our options for upgrade?

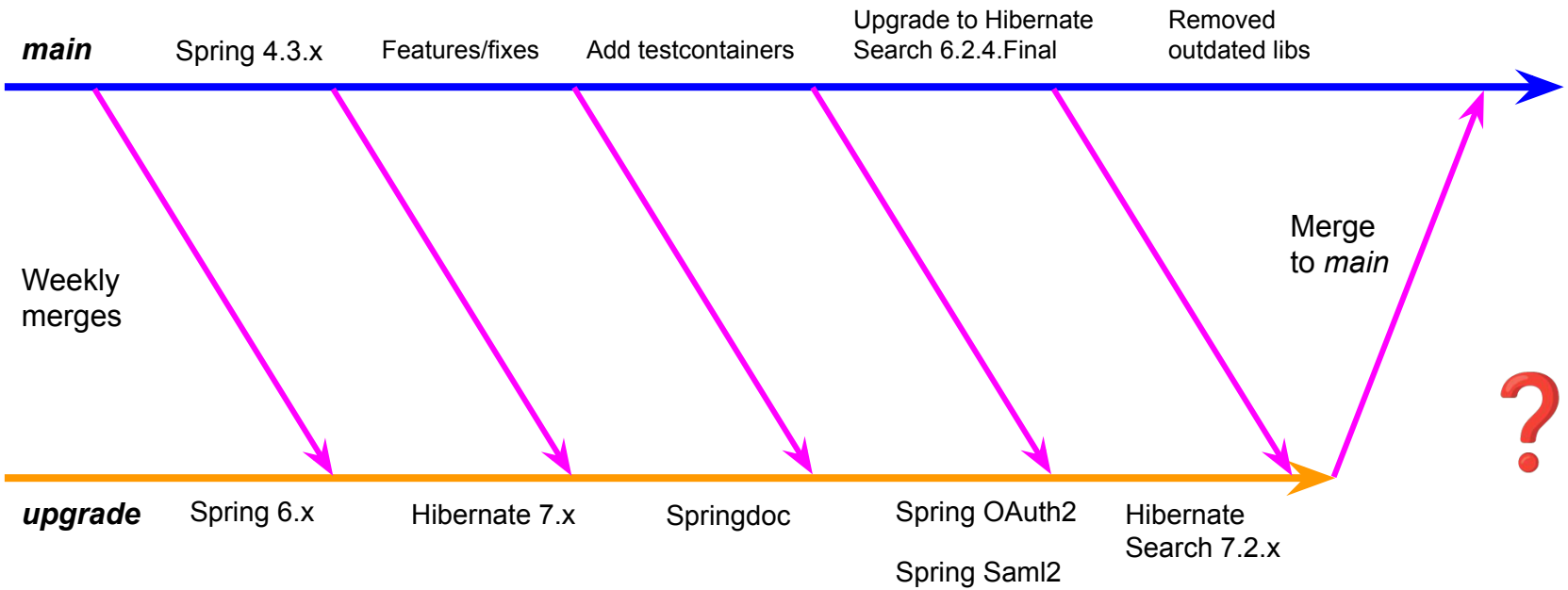
1. Gradual upgrade from:
    a. Spring 4.3.x to Spring 5.0.x
    b. Spring 5.0.x to Spring 5.3.x
    c. Spring 5.3.x to Spring 6.0.x.
    d. **Short-lived branches**, same with other frameworks.

2. Long lived branch for upgrade from **Spring 4.3.x** to **Spring 6.x**, all libs and frameworks in one sweep.

Options for upgrade 1/2 (main + “short” lived branches)



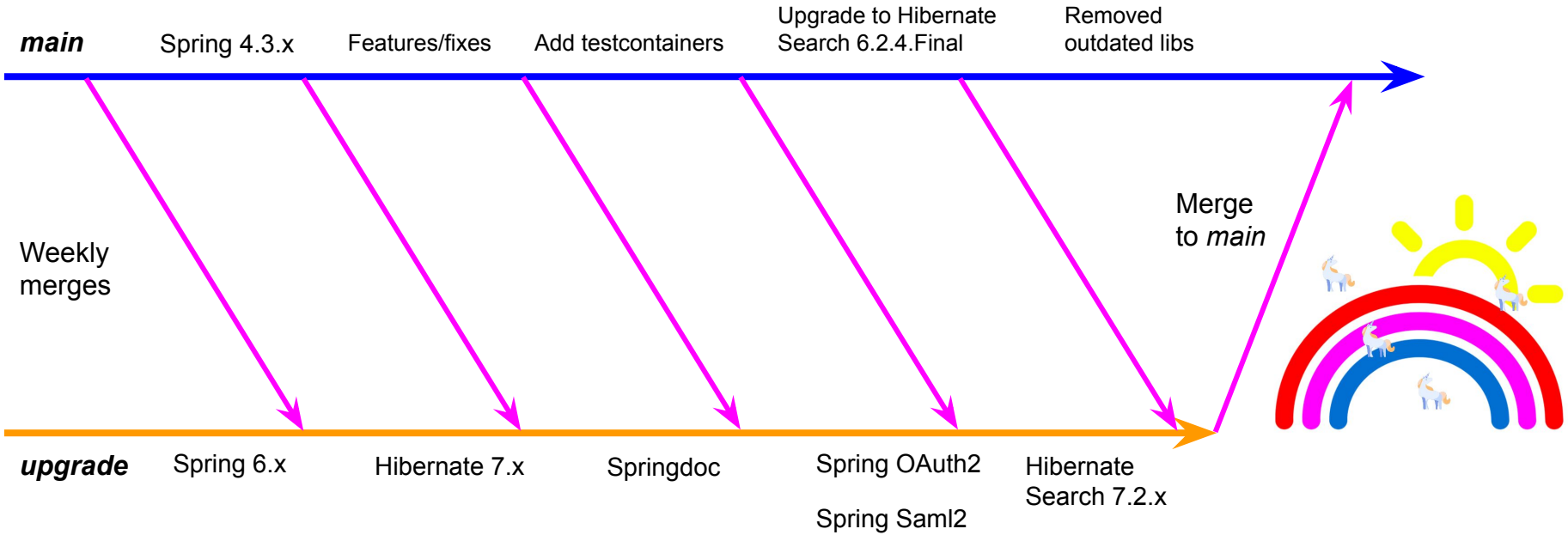
Options for upgrade 2/2 (main + long lived remote branch)



What did we do?

- Because:
  - There were **6 major/minor** releases between 4.3.x and 6.0.x, and **hundreds** of fixed issues.
  - We would have been affected by those issues during the time between 4.3.x and 6.0.x.
  - It would take more time and might not even be possible.
  - So choosing **Option 1 (short lived branches)** was **NO - GO.** ❌

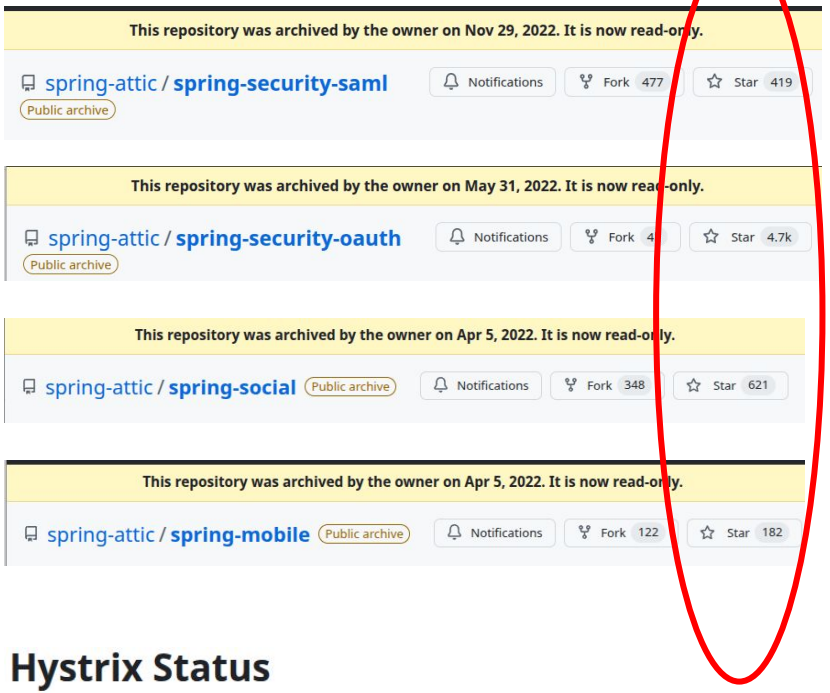
We chose option 2



Challenge 2 - Libraries that disappeared

Libraries that disappeared

- Spring-security-saml
- Spring-security-oauth
- Spring-social
- Spring-mobile
- Springockito
- Hystrix
- Fongo
- Spring-Velocity



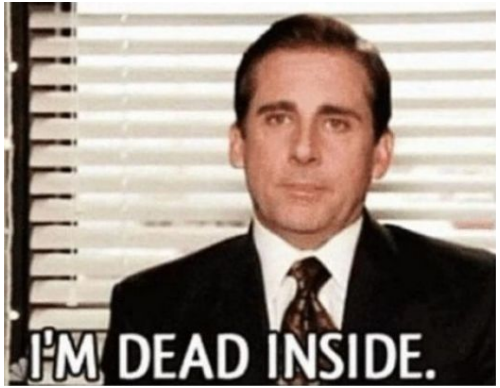
Libraries that disappeared - replacements

- `Spring-security-saml` → `spring-security-saml2-service-provider`
- `Spring-security-oauth` → `spring-security-oauth2-client`
- `Spring-social` → `removed in the main branch`
- `Spring-mobile` → `removed in the main branch`
- `Springockito` → we removed it and replaced with custom implementation of a `TestExecutionListener`, as `@MockBean` isn't available in the base Spring
- `Hystrix` → `resilience4j`
- `Fongo` → `testcontainers`
- `Spring-velocity` → `removed, replaced with SPA`

Challenge 3 - Changed APIs

Changed APIs

- javax -> jakarta
- Hibernate, Hibernate Search
- Spring Security Core
- Infinispan
- Mockito (we removed PowerMock and replaced it using Mockito features)
- Apache Tiles
- Spring-Webflow
- Apache HttpClient, Apache HttpCore



javax.* -> jakarta.*

javax.* -> jakarta.*

- It is tougher than it seems
- APIs that we had to replace
  - Jakarta Persistence API
  - Jakarta Servlet API
  - Jakarta Mail API
  - Jakarta Validation API
  - Jakarta Annotation API
  - Jakarta Transaction API
  - Jakarta EL API
  - Jakarta JMS API
- The hardest were:
  - Jakarta Servlet API
  - Jakarta Persistence API

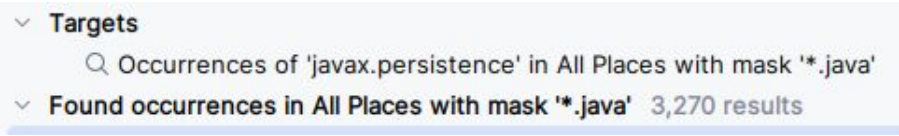
Jakarta Servlet API

- A lot of usages

- Servlet API being a transitive dependency in a lot of our dependencies.
  - This meant we needed to upgrade them …
  - To name some:
    - Hystrix
    - CXF
    - HttpClient, HttpCore
    - and so on …

Jakarta Persistence API

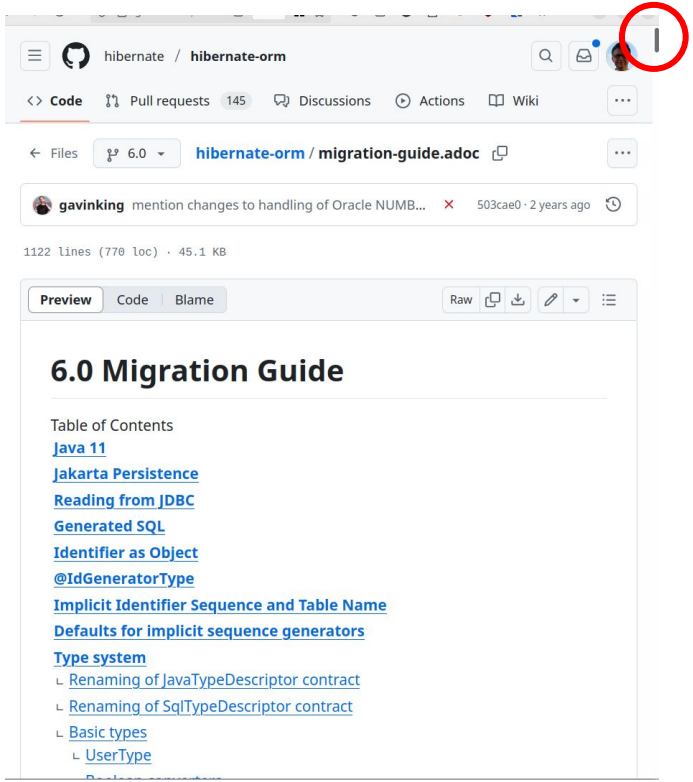
- A lot of usages



- Some of the libraries have duplicate versions for jakarta and javax!
- `infinispan-core` for javax
- `infinispan-core-jakarta` for jakarta
- which caught us by surprise.

Hibernate 1/6

- Where do I even start?
- There is this mythical thing called documentation.
- Migration guide is your friend https://github.com/hibernate/hibernate-orm/blob/6.0/migration-guide.adoc#instant-mapping-changes



Hibernate 2/6

- Deprecated Criteria API -> JPA Criteria API

Old:

```
private Long getaCoursePaymentsCount(Course course) {
   Criteria criteria = getCriteria(CoursePayment.class);
   criteria.add(eq(FieldName.course.toString(), course));
   criteria.add(isNotNull(FieldName.dateOfPayment.toString()));

   return (Long) criteria.setProjection(Projections.rowCount())
               .uniqueResult();
}
```

```
private Criteria getCriteria(Class clazz) {
   return getHibernateSession().createCriteria(clazz);
}

private Session getHibernateSession() {
   final Session session =
       (org.hibernate.Session) entityManager.getDelegate();
   return session;
}
```

New:

```
private Long getCoursePaymentsCount(Course course) {
   CriteriaBuilder cb = entityManager.getCriteriaBuilder();
   CriteriaQuery<Long> query = cb.createQuery(Long.class);
   Root<CoursePayment> root = query.from(CoursePayment.class);
   query.select(cb.count(root));
   query.where(cb.equal(root.get(FieldName.course.toString()), course),
           cb.isNotNull(root.get(FieldName.dateOfPayment.toString())));
   return entityManager.createQuery(query).getSingleResult();
}
```

Hibernate 3/6

- Changes in Mappings:
  - Ref: https://github.com/hibernate/hibernate-orm/blob/6.0/migration-guide.adoc#instant-mapping-changes
  - In Hibernate 5.7 this mapping:

    Entity:

    ```
    @Temporal(TemporalType.TIMESTAMP)
    private Date appliedOn;
    ```

    Table:

    ```
    `appliedOn` DATE
    ```

    worked without any issues.
  - It didn't work in Hibernate 6.x, so we decided to change the type on the DB level

    Table:

    ```
    `appliedOn` DATETIME(6)
    ```
  - We decided to set:

    ```
    <prop key="hibernate.type.preferred_instant_jdbc_type">DATE</prop>
    ```

Hibernate 4/6

- In Hibernate 5.7 mapping like:

```
@Enumerated
private QuestionAttachmentStatus questionAttachmentStatus;
```

  could be mapped to an INT column.

- Now it needed to be mapped to TINYINT.
- What is good as it makes our system a tiny bit more optimal.

Hibernate 5/6

- Stronger validation of native queries.
- Native queries return Long instead of BigInteger

  Old:

```
return ((BigInteger) query.getSingleResult());
```

  New:

```
return ((Long) query.getSingleResult());
```

- Better validation of native queries for entities with secondary tables

  Old:

```
Query query = entityManager.createNativeQuery(
       "SELECT c.*, crp.* FROM CourseRegistration c " +
       " LEFT JOIN courseregistration_price crp ON crp.id = c.id", CourseRegistration.class);
```

  New:

```
Query query = entityManager.createNativeQuery(
       "SELECT c.*, crp.currency_id, crp.amount FROM CourseRegistration c " +
       " LEFT JOIN courseregistration_price crp ON crp.id = c.id)", CourseRegistration.class);
```

Hibernate 6/6

- Identifier generation default has been changed

> As of 6.0, Hibernate by default creates a sequence per entity hierarchy instead of a single sequence `hibernate_sequence`.
>
> Due to this change, users that previously used `@GeneratedValue(strategy = GenerationStrategy.AUTO)` or simply `@GeneratedValue` (since `AUTO` is the default), need to ensure that the database now contains sequences for every entity, named `<entity name>_seq`. For an entity `Person`, a sequence `person_seq` is expected to exist. It's best to run hbm2ddl (e.g. by temporarily setting `hbm2ddl.auto=create`) to obtain a list of DDL statements for the sequences.

- We decided to depend on our Mysql to generate Identifiers.

From:

```
@Id
@GeneratedValue
private Integer id;
```

To:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer id;
```

Hibernate - Summary

- Hibernate solidified their query and mapping validation.
- Many bugs were solved.
- Removed deprecated APIs.

Hibernate Search - plan

- We wanted to be able to use newer versions of Elasticsearch.

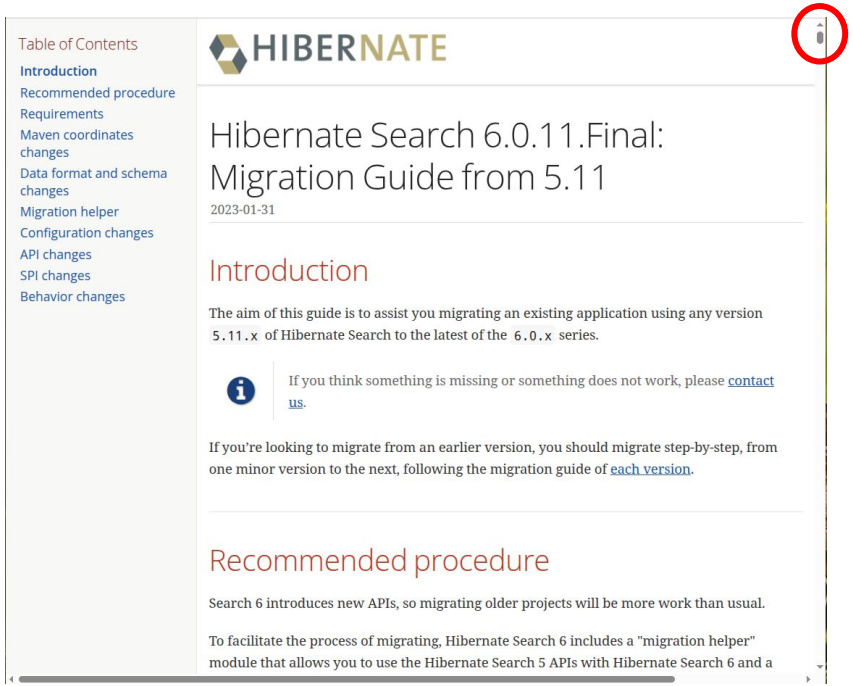
Hibernate Search Version	Compatible Elasticsearch Version
5.9.3.Final	2.0 – 5.6
6.2.4.Final	7.10 – 8.12
7.2.2.Final	8.14 – 8.15

- **Plan**:
  - 5.9.3.Final (Lucene API supported, default backend)
  - -> **6.2.4.Final** - **main branch**
    (Lucene types removed from public API,
    Hibernate 6.2.+, Elasticsearch: 7.x.+)
  - -> **7.2.2.Final - upgrade branch**
    (API is fully backend-agnostic,
    Hibernate 6.6.+, Elasticsearch: 8.x.+)

Hibernate Search Version	Compatible Hibernate ORM Versions
5.9.3.Final	5.2.3.Final – 5.2.x
6.2.4.Final	6.2.24.Final
7.2.2.Final	6.6.0.Final

Hibernate Search - 1/6

- Start with: https://hibernate.org/search/documentation/migrate/
- 5.x -> 6.x: https://docs.jboss.org/hibernate/search/6.0/migration/html_single/



Hibernate Search - 2/6

- Mapping Annotation changes
- **Old**:

```
@org.hibernate.search.annotations.Indexed
public class ContentDocument {
    // ...
    @Field(name = "contentType")
    @SortableField(forField = "contentType")
    @FieldBridge(impl = ContentTypeSortFieldBridge.class)
    public ContentType getType() {
        return ContentType.ContentDocument;
    }

  }
```

- **New**:

```
@org.hibernate.search.mapper.pojo.mapping.definition.annotation.Indexed
public class ContentDocument {
    // ...
    @Field(name = "contentType")
    @IndexingDependency(reindexOnUpdate = ReindexOnUpdate.NO)
    public ContentType getType() {
        return ContentType.ContentDocument;
    }

  }
```

Hibernate Search - 3/6

- Annotation changes (Ref: https://docs.jboss.org/hibernate/search/6.0/migration/html_single/)

Summary: Don't read it :) Use when needed

1. **Complete rewrite of Bridge API.**
2. **Basic Annotations**:
   - `@org.hibernate.search.annotations.Indexed` → `@org.hibernate.search.mapper.pojo.mapping.definition.annotation.Indexed`
   - `@Field` → Split into multiple specialized annotations:
     - `@FullTextField`: For analyzed text fields (full-text search)
     - `@KeywordField`: For non-analyzed fields (exact matching, sorting, aggregations)
     - `@GenericField`: For non-text fields (numbers, dates, etc.)
3. **Field Customization**:
   - `@SortableField` → `sortable = Sortable.YES` parameter in field annotations
   - `@FieldBridge` → `@ValueBridgeRef` or `@TypeBinderRef`
   - `@Analyzer` → `analyzer` parameter in `@FullTextField`
4. **New Annotations**:
   - `@IndexingDependency`: Controls when fields are reindexed
   - `@ObjectPath` and `@PropertyValue`: Define paths to dependent properties
   - `@TypeBinding`: Binds a custom type to an entity

Hibernate Search - 4/6

- Indexing API changes:

```
FullTextSession fullTextSession = prepareFullTextSession();
fullTextSession.index(entityToCreateIndex);
```

```
SearchSession searchSession = getSearchSession();
searchSession.indexingPlan().addOrUpdate(entityToCreateIndex);
```

Hibernate Search - 5/6

- Search API changes:

```
FullTextEntityManager fullTextEntityManager = Search.getFullTextEntityManager(entityManager);
QueryBuilder queryBuilder = fullTextEntityManager.getSearchFactory()
        .buildQueryBuilder().forEntity(Course.class).get();
Query luceneQuery = queryBuilder.keyword().onField("title")
        .matching(searchTerm).createQuery();
FullTextQuery fullTextQuery = fullTextEntityManager.createFullTextQuery(luceneQuery, Course.class);
```

```
SearchSession searchSession = org.hibernate.search.mapper.orm.Search.session(entityManager);
SearchResult<Course> result = searchSession.search(Course.class)
      .where(f -> f.match().field("title").matching(searchTerm)).fetch(20);
```

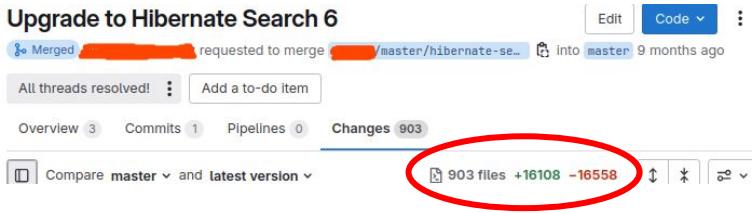
Hibernate Search - 6/6

- Key Search API changes:
- `FullTextEntityManager` → `SearchSession`
- `QueryBuilder` → Lambda-based search DSL
- `FullTextQuery` → `SearchResult`
- New fluent API for defining queries

Hibernate Search - summary

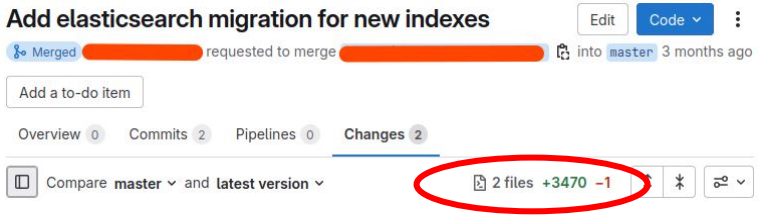
- There were a lot of usages.

- We have done this on the main branch.



- Searches across the whole application had to be tested, and that was a **huge effort.**

- **(Almost) All indexes** had to be recreated, that complicated our deployment.



- APIs of **6.2.4.Final** are mostly the same as in **7.2.x**, we can postpone upgrade to **7.2.4.Final** to after the migration.

Spring security 1/4

- Spring Security Core (3.x -> 6.x) - changes in:
  - Authentication providers
  - Password encoders
  - XML / Java based configuration
  - CSRF filters
  - Authorization

```
<sec:authorize ifAllGranted="ROLE_ADMIN,ROLE_USER">
        <p>Must have ROLE_ADMIN and ROLE_USER</p>
</sec:authorize>
```

```
<sec:authorize access="hasRole('ROLE_ADMIN') and hasRole('ROLE_USER')">
       <p>Must have ROLE_ADMIN and ROLE_USER</p>
</sec:authorize>
```

- And all would be fine if those were the only changes …

Ref: https://docs.spring.io/spring-security/site/migrate/current/3-to-4/html5/migrate-3-to-4-jc.html

Spring security 2/4

This repository was archived by the owner on Nov 29, 2022. It is now read-only.

spring-attic / **spring-security-saml**

Public archive

Notifications | Fork 477 | Star 419

This repository was archived by the owner on May 31, 2022. It is now read-only.

spring-attic / **spring-security-oauth**

Public archive

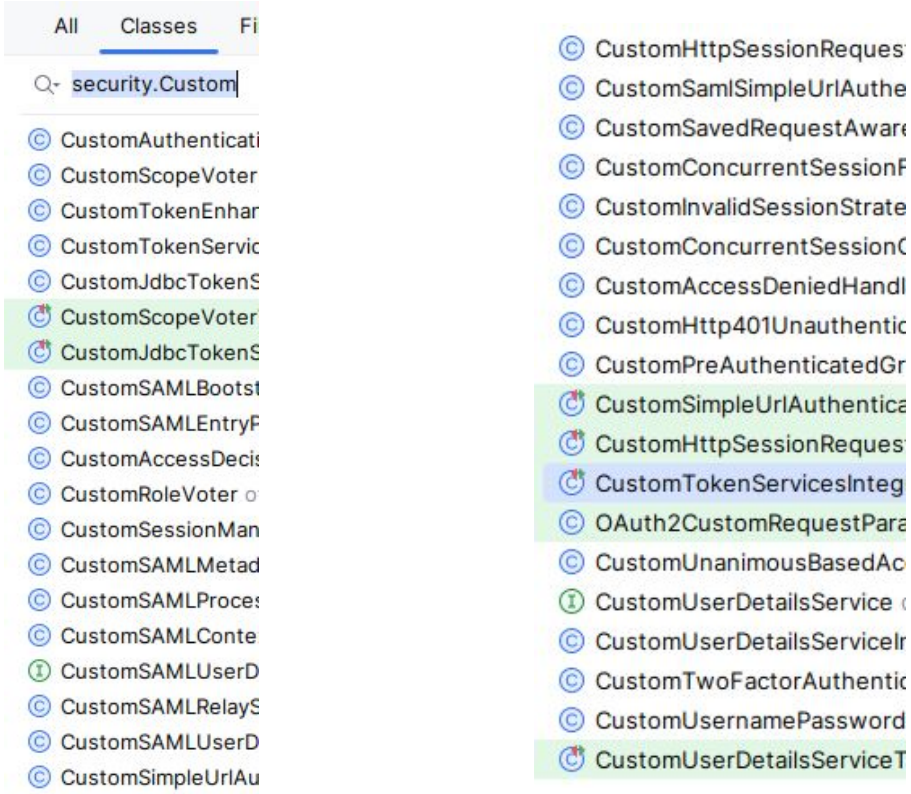
Notifications | Fork 4k | Star 4.7k

… needed **complete** rewrite, around 7000 LOC to rewrite ….

Spring security 3/4

How to assess effort needed to upgrade of Spring Security and related libraries?

Simple heuristic:
The higher the number of classes with name starting with “Custom” related to Spring Security you have, the harder it will be.

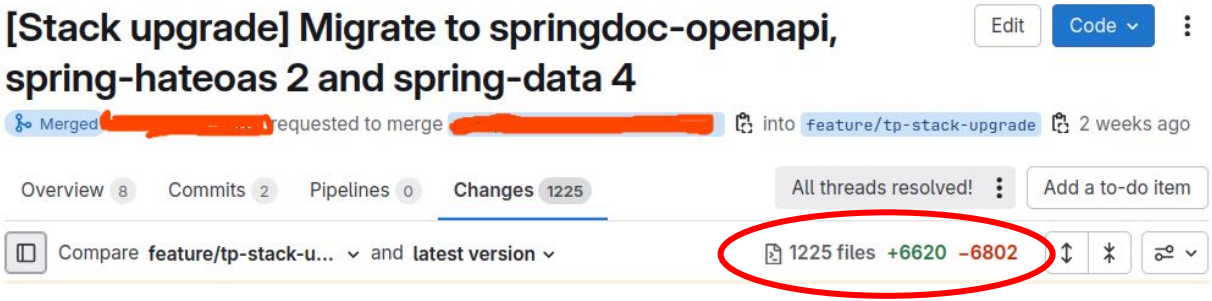


Spring security 4/4

This is still a **Work In Progress …**

Springfox -> Springdoc 1/2

- Springfox has never been recognized by the Spring team, but there was no alternative.
- Finally, we have got springdoc-openapi and spring-hateoas 2.



Springfox -> Springdoc 2/2

- Mainly, simple annotation changes.
- The whole configuration had to be changed.
- **Expected** changes in **generated OpenAPI documentation**, that can lead to **different generated TypeScript types** used on Frontend applications.

This is still a **Work In Progress …**

Infinispan

- Migration from 8.x to 14.x
- The whole xml configuration had to be changed.

```
<invalidation-cache name="enrollments">
    <eviction size="5000000" strategy="LIRS" type="COUNT"/>
    <expiration max-idle="-1" lifespan="1800000" interval="5000"/>
</invalidation-cache>
```

```
<invalidation-cache name="enrollments">
    <memory max-count="5000000" when-full="REMOVE"/>
    <expiration max-idle="-1" lifespan="1800000" interval="5000"/>
</invalidation-cache>
```

- Finally better control over **the memory use**.
- We expect some issues once deploying to AWS.
- All in all, not that hard but remember, that there are two artifacts in Mvn repo.

Home » org.infinispan » infinispan-core

Infinispan Core

Infinispan core module

Home » org.infinispan » infinispan-core-jakarta

Infinispan Core Jakarta EE

Infinispan core module for Jakarta EE

Some smaller API changes

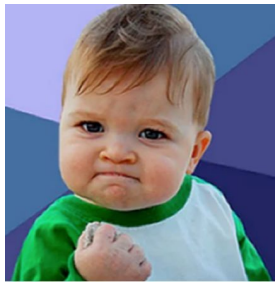
- MockServer → WireMock
- PowerMock → Mockito
- Apache Tiles → removed
- Apache HttpClient, Apache HttpCore
- many others …

Benefits

Benefits

Server startup cut to 50%

Better resource utilization



Improved processes in company

New features! Java, Spring, Hibernate

Better vulnerability management

Developer Experience

Lessons learned

Don’t allow to accumulate Tech Debt

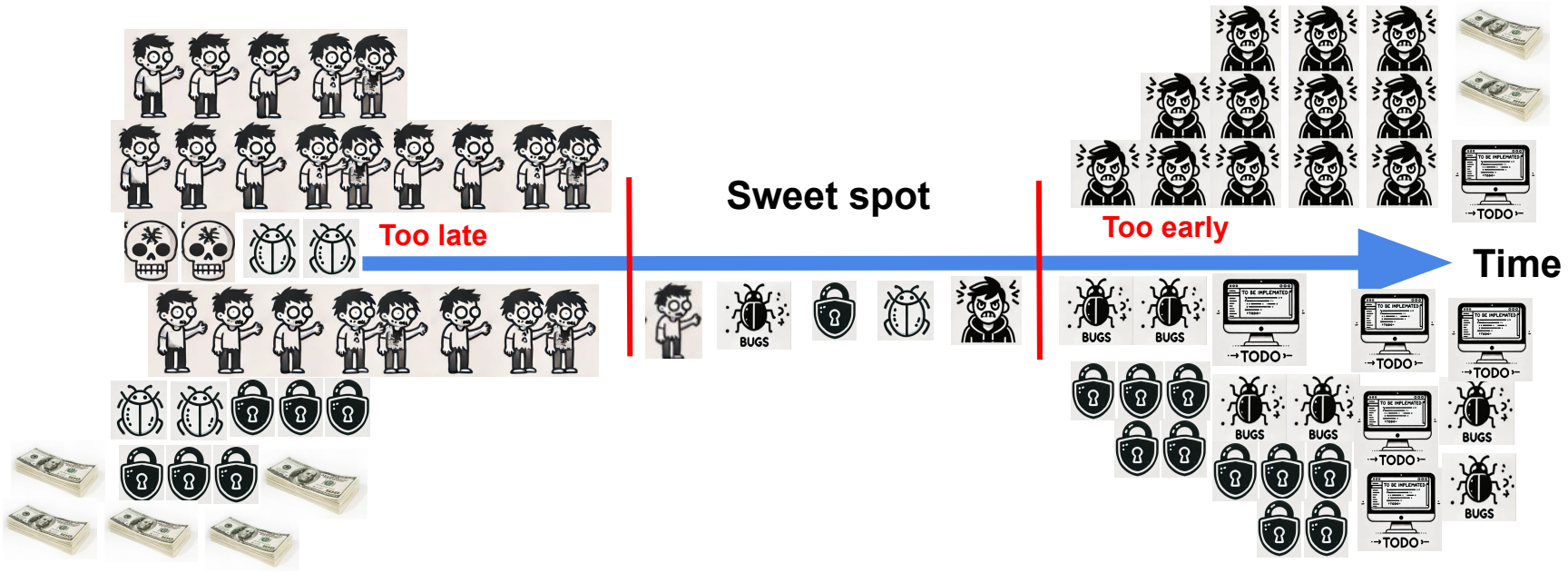
Avoid those “strategies” to deal with Tech Debt:

Ref: https://www.linkedin.com/posts/ralph-aboujaoude-diaz-40838313_technology-leader-technical-activity-7207774652469854208-GZvh/?utm_source=share&utm_medium=member_android

Find your sweet spot for the Upgrade window

The Sweet spot for the Upgrade window

- Find your sweet spot between lagging behind and upgrading too early.
- Martin Fowler's advice: "*If it's painful, do it more often*".



Last notes

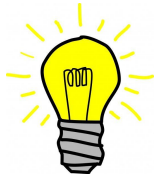
- Remember, upgrading a Monolith is hard work.

- Neither Spring nor Hibernate are bad, accumulating Technical Debt is!

- You don’t need to do anything, Upgrades and Tech Debt will find you at some point :)

- Having a good tests coverage helps a lot.

Suggestions



- Include as many people from the team to help with migration.

- Spend effort on planning migration.
- Require your team to keep versions of your frameworks/libs up to date.
- Add recurring tasks for keeping upgrades in check.
  Automate security upgrades if possible.

- Improve processes in your workplace.

Q/A

Thank You!

Presentation Slides

marekdominiak