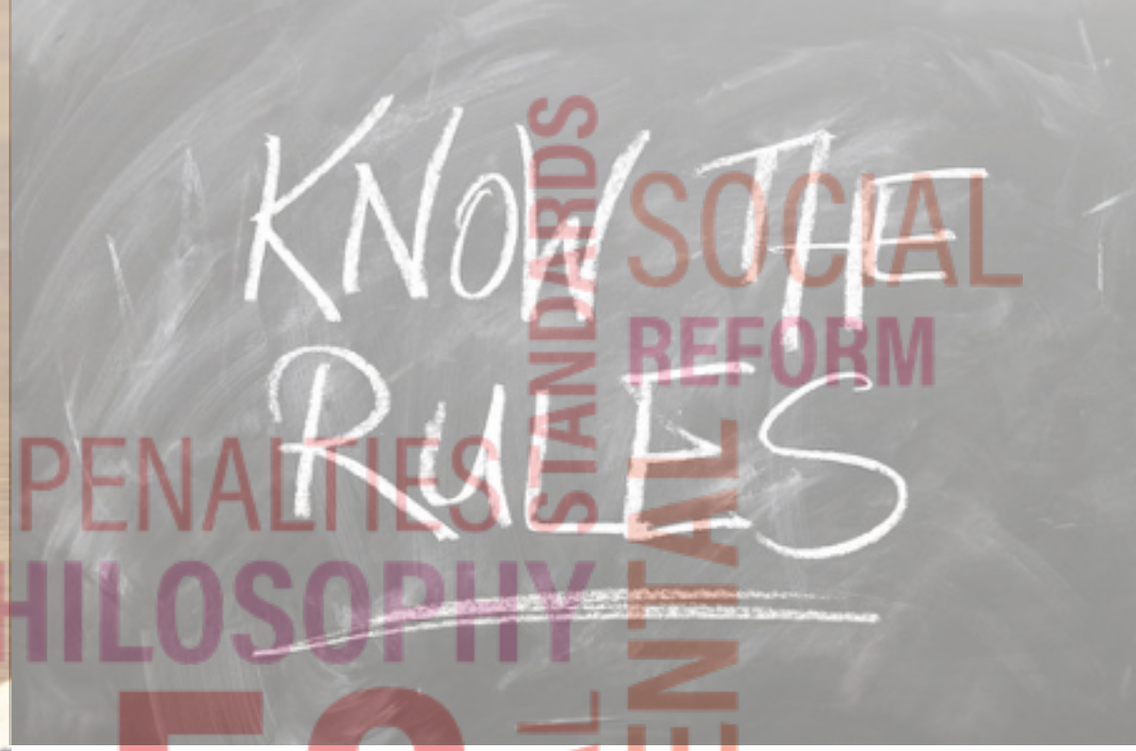
Rules as an Architectural Pattern For Development

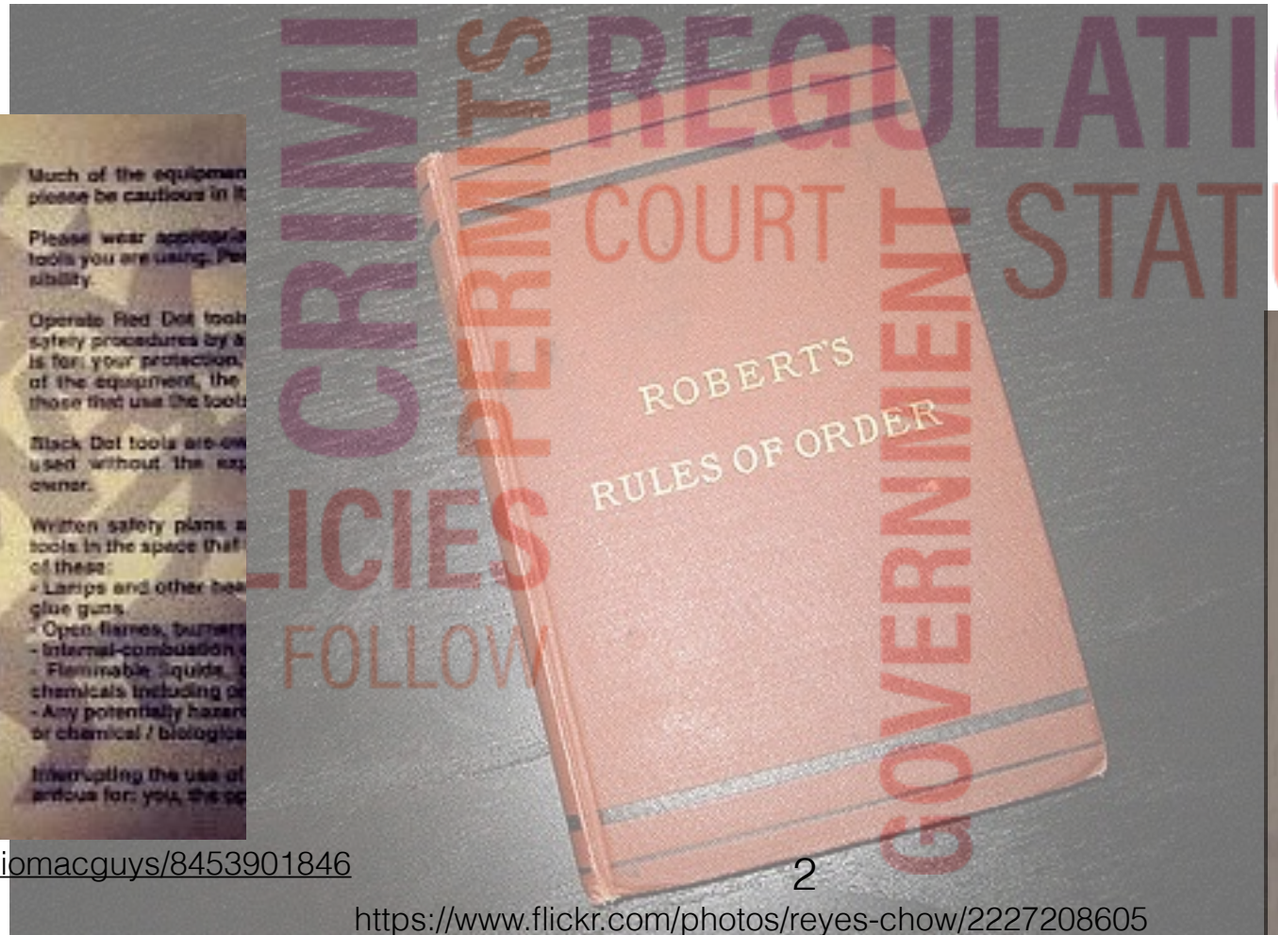
Steve Swing
@sswing

https://github.com/steveswing/cere

http://www.thebluediamondgallery.com/scrabble/r/rules.html

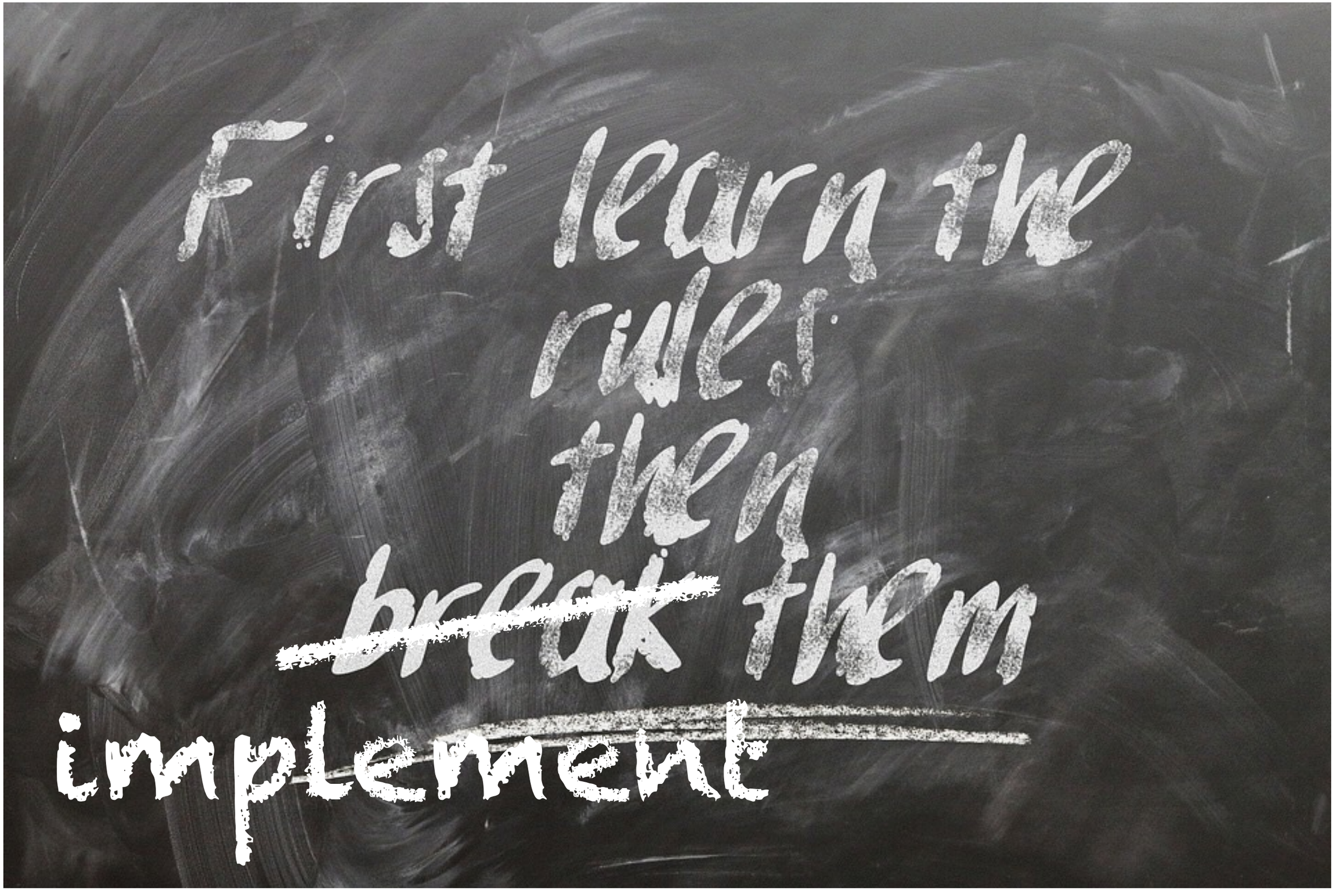


https://www.flickr.com/photos/radiomacguys/8453901846



https://www.flickr.com/photos/reyes-chow/2227208605

http://i.imgur.com/qu68T.jpg


First learn the rules
then
break them
implement

Disclaimer

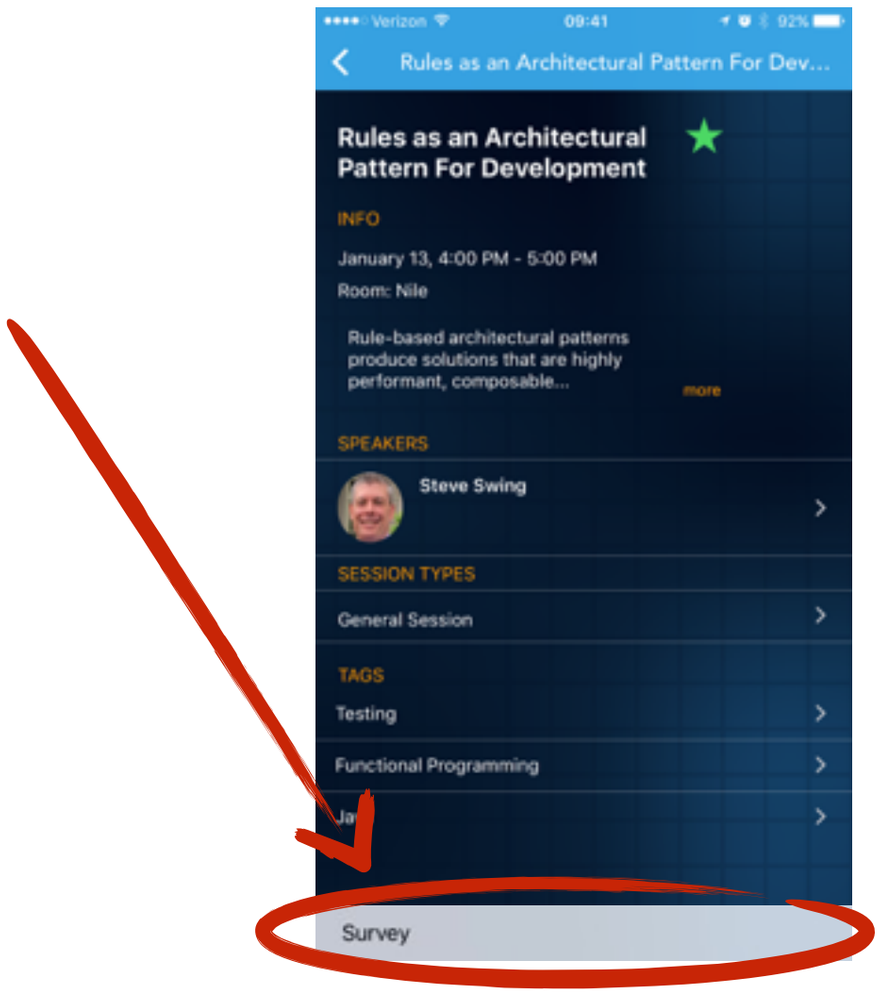
- Already using a rule engine such as Drools? Carry on.
- Don't agree? Learn Drools or other rule engine you like
- However…
  - Use rule design patterns any where
  - Training constraints prevent Drools/other inference engine
  - Architectural approval denied
  - External dependencies/platform constraints
  - Control freak

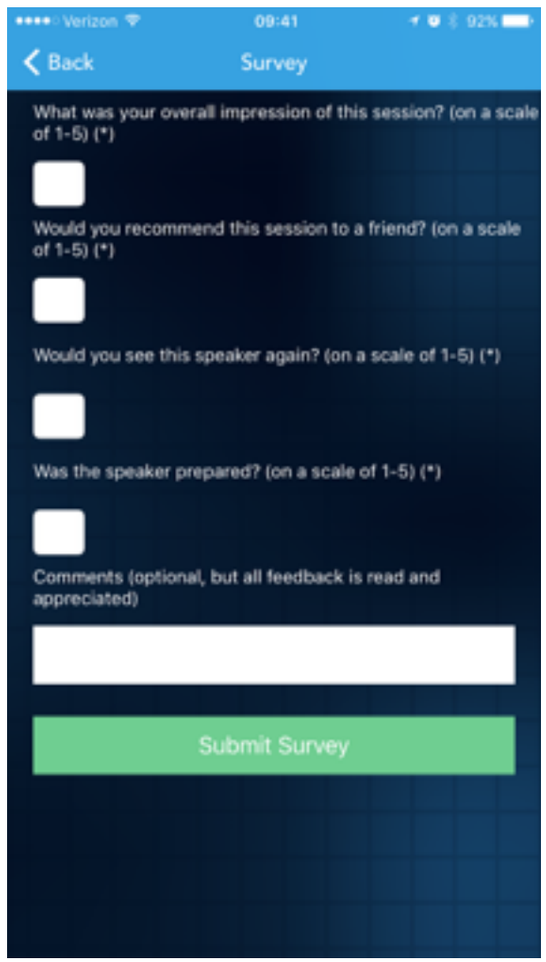
“If you can't be a good example, then you'll just have to be a horrible warning.”

– Catherine Aird

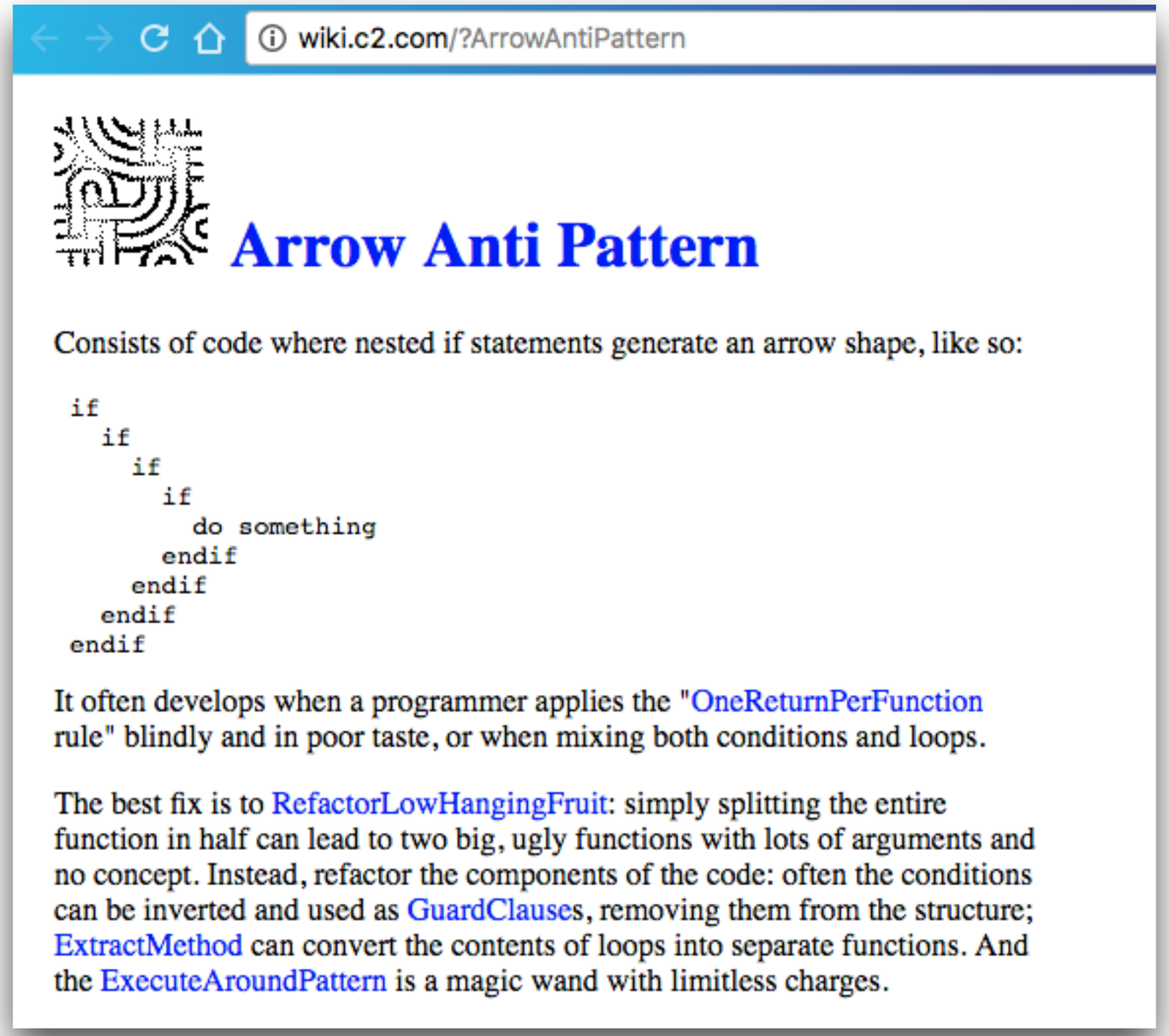
EventsXD Survey

- 5 = Good Example
- 1 = Horrible Warning
- https://github.com/steveswing/cere

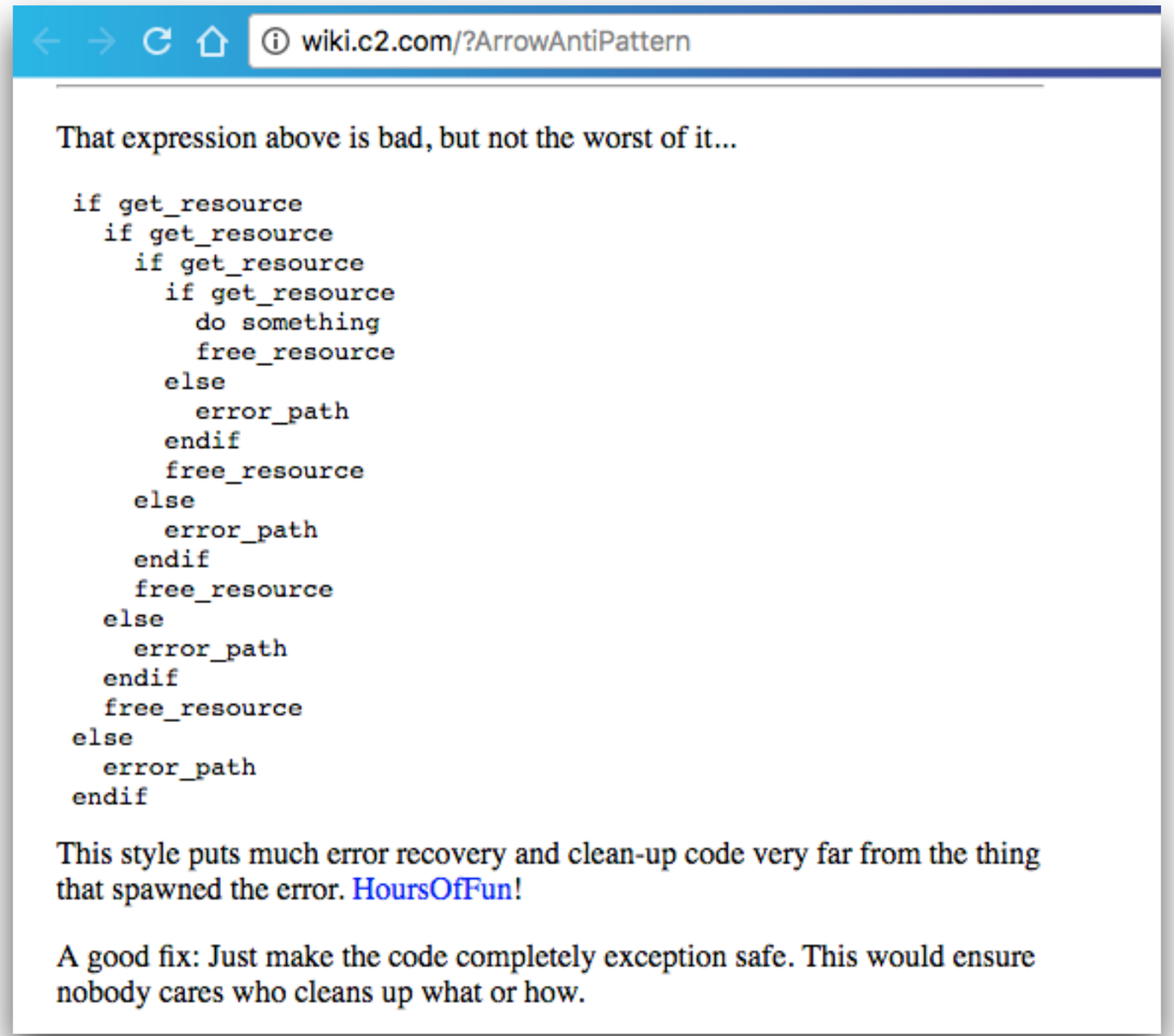




Arrow Anti-Pattern



Arrow Anti-Pattern Worse



Java Tutorial If Else

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html

The `if-then-else` Statement

The `if-then-else` statement provides a secondary path of execution when an "if" clause evaluates to `false`. You could use an `if-then-else` statement in the *applyBrakes* method to take some action if the brakes are applied when the bicycle is not in motion. In this case, the action is to simply print an error message stating that the bicycle has already stopped.

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

The following program, `IfElseDemo`, assigns a grade based on the value of a test score: an A for a score of 90% or above, a B for a score of 80% or above, and so on.

```
class IfElseDemo {
    public static void main(String[] args) {

        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

The output from the program is:

```
Grade = C
```

Java Tutorial If Else (Test)

```java
import java.util.Arrays;
import java.util.Collection;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;

import static org.junit.Assert.assertEquals;

@RunWith(Parameterized.class)
public class IfElseDemoTest {
    private int input;
    private char expected;

    public IfElseDemoTest(int input, char expected) {
        this.input = input;
        this.expected = expected;
    }

    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(new Object[][]{
                {100, 'A'}, {91, 'A'}, {90, 'A'}, {89, 'B'}, {81, 'B'}, {80, 'B'}, {79, 'C'},
                {71, 'C'}, {70, 'C'}, {69, 'D'}, {61, 'D'}, {60, 'D'}, {59, 'F'}, {0, 'F'}});
    }

    @Test
    public void testScores() {
        int testscore = input;
        char grade;
        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }

        assertEquals(String.format("expected match for %d", input), expected, grade);
    }
}
```

If Else (Predicates)

```
2
3     import ...
19
20    @RunWith(Parameterized.class)
21    public class IfElseDemoPredicates {
22        private int input;
23        private char expected;
24        private Predicate<Integer> scored90 = score -> score >= 90;
25        private Predicate<Integer> scored80 = score -> score >= 80;
26        private Predicate<Integer> scored70 = score -> score >= 70;
27        private Predicate<Integer> scored60 = score -> score >= 60;
28        private Predicate<Integer> scored0 = score -> score >= 0;
29        private Predicate<Integer> scoredA = scored90;
30        private Predicate<Integer> scoredB = scored80.and(scored90.negate());
31        private Predicate<Integer> scoredC = scored70.and(scored80.negate());
32        private Predicate<Integer> scoredD = scored60.and(scored70.negate());
33        private Predicate<Integer> scoredF = scored0.and(scored60.negate());
34        private List<Function<Integer, Character>> rules = initialize();
35
36        public IfElseDemoPredicates(final int input, final char expected) {...}
40
41        @Parameters
42    @   public static Collection<Object[]> data() {
43            return Arrays.asList(new Object[][]{{100, 'A'}, {91, 'A'}, {90, 'A'}, {89, 'B'}, {81, 'B'}, {80, 'B'},
44                    {79, 'C'}, {71, 'C'}, {70, 'C'}, {69, 'D'}, {61, 'D'}, {60, 'D'}, {59, 'F'}, {0, 'F'}});
45        }
46
47        private List<Function<Integer, Character>> initialize() {
48            final List<Function<Integer, Character>> result = new ArrayList<>();
49            result.add((score) -> scoredA.test(score) ? 'A' : null);
50            result.add((score) -> scoredB.test(score) ? 'B' : null);
51            result.add((score) -> scoredC.test(score) ? 'C' : null);
52            result.add((score) -> scoredD.test(score) ? 'D' : null);
53            result.add((score) -> scoredF.test(score) ? 'F' : null);
54            return result;
55        }
56
57        @Test
58        public void testScores() {...}
75
76        @Test
77        public void alternativeScoring() throws Exception {
78            final Optional<Character> first = rules.stream().map(f -> f.apply(input)).filter(Objects::nonNull).findFirst();
79            assertTrue( message: "expected true", first.isPresent());
80            final char grade = first.get();
81            assertEquals( message: "expected match", expected, grade);
82        }
83    }
84
```

Simple Implementation

- **Use Predicates to replace complex if then else logic**

```
final CustomerType customerType = customer.getType();
final int totalSales = customer.getTotalSales();
if (CustomerType.SMALL.equals(customerType)) {
    if (totalSales > 1000) {
        // SMALL VIP Customer
    }
} else if (CustomerType.MEDIUM.equals(customerType)) {
    if (totalSales > 5000) {
        // MEDIUM VIP Customer
    }
} else if (CustomerType.LARGE.equals(customerType)) {
    if (totalSales > 10000) {
        // LARGE VIP Customer
    }
}
```

Predicates

```
public class Predicates {
    public Predicate<Customer> small = c -> CustomerType.SMALL.equals(c.getType());

    public Predicate<Customer> medium = c -> CustomerType.MEDIUM.equals(c.getType());

    public Predicate<Customer> large = c -> CustomerType.LARGE.equals(c.getType());
```

Predicates

```
public class Predicates {
    public Predicate<Customer> small = c -> c.getType() == CustomerType.SMALL;

    public Predicate<Customer> medium = c -> c.getType() == CustomerType.MEDIUM;

    public Predicate<Customer> large = c -> c.getType() == CustomerType.LARGE;

    public Predicate<Customer> smallOrLarge = small.or(large);

    public Predicate<Customer> notMedium = medium.negate();
```

Predicates

```
public class Predicates {
    private static final Map<CustomerType, Integer> salesThresholds = initializeSalesThresholds();
    private static final Instant sixMonthsAgo = offset(systemDefaultZone(), of(-6L, MONTHS)).instant();
    public Predicate<Customer> small = c -> CustomerType.SMALL.equals(c.getType());
    public Predicate<Customer> medium = c -> CustomerType.MEDIUM.equals(c.getType());
    public Predicate<Customer> large = c -> CustomerType.LARGE.equals(c.getType());
    public Predicate<Customer> smallOrLarge = small.or(large);
    public Predicate<Customer> notMedium = medium.negate();
    public Predicate<Customer> vip = c -> c.getTotalSales() > salesThresholds.getOrDefault(c.getType(), MAX_VALUE);
    public Predicate<Customer> mediumVip = medium.and(vip);
    public Predicate<Customer> recentSales = c -> sixMonthsAgo.isBefore(asInstant(c.getRecentSalesDate()));
    public Predicate<Customer> currentMedVip = mediumVip.and(recentSales);

    private static Map<CustomerType, Integer> initializeSalesThresholds() {
        final Map<CustomerType, Integer> result = new TreeMap<>();
        result.put(CustomerType.SMALL, 1000);
        result.put(CustomerType.MEDIUM, 5000);
        result.put(CustomerType.LARGE, 10000);
        return result;
    }

    private Instant asInstant(final LocalDateTime d) { return d.toInstant(of(systemDefault().getId())); }
}
```

Predicates

```java
/**...*/
@FunctionalInterface
public interface Predicate<T> {

    /**...*/
    boolean test(T t);

    /**...*/
    default Predicate<T> and(Predicate<? super T> other) {
        Objects.requireNonNull(other);
        return (t) -> test(t) && other.test(t);
    }

    /**...*/
    default Predicate<T> negate() { return (t) -> !test(t); }

    /**...*/
    default Predicate<T> or(Predicate<? super T> other) {
        Objects.requireNonNull(other);
        return (t) -> test(t) || other.test(t);
    }

    /**...*/
    static <T> Predicate<T> isEqual(Object targetRef) {
        return (null == targetRef)
                ? Objects::isNull
                : object -> targetRef.equals(object);
    }
}
```

What is a rule?

- "When... then..."
- A rule is a list of conditions and a list of actions
- If all conditions evaluate to true the actions are performed
- Pattern match (simple to extremely complex)

What is a rule?

```java
@FunctionalInterface
public interface Rule<T> extends Serializable {
    boolean fire(T t);
}
```

What is a rule?

```
 2
 3    import ...
 8
 9    public interface Rule<T> extends Comparable<Rule<T>>, Serializable {
10        RuleType getRuleType();
11
12        int getId();
13
14        String getName();
15
16        Collection<Condition<T>> getConditions();
17
18        Collection<Action<T>> getActions();
19
20        Rule<T> add(Condition<T> c);
21
22        Rule<T> add(Action<T> a);
23
24        boolean fire(T t);
25    }
26
```

What is a rule (impl)?

```java
public class RuleImpl<T> implements Rule<T> {
    private Collection<Condition<T>> conditions = new ArrayList<>();
    private Collection<Action<T>> actions = new ArrayList<>();

    public Collection<Condition<T>> getConditions() {
        return Collections.unmodifiableCollection(conditions);
    }

    public Collection<Action<T>> getActions() {
        return Collections.unmodifiableCollection(actions);
    }

    @Override
    public boolean fire(final T t) {
        if (!conditions.stream().allMatch(c -> c.test(t))) {
            return false;
        }

        actions.forEach(a -> a.perform(t));

        return true;
    }
```

What is a rule (impl)?

```
public class RuleImpl<T> implements Rule<T> {
    public static final long serialVersionUID = 1L;
    private String id;
    private Collection<Condition<T>> conditions = new ArrayList<>();
    private Collection<Action<T>> actions = new ArrayList<>();
    private RuleType ruleType = RuleType.STANDARD;

    public Collection<Condition<T>> getConditions() { return Collections.

    public Collection<Action<T>> getActions() { return Collections.unmodi

    @Override
    public RuleType getRuleType() { return ruleType; }

    @Override
    public boolean fire(final T t) {
        if (conditions.stream().allMatch(c -> c.test(t))) {
            actions.forEach(a -> a.perform(t));
            return true;
        }
        return false;
    }
```

Conditions & Predicates

```
@FunctionalInterface
public interface Condition<T> {
    boolean test(T t);
}
```

Conditions & Predicates

```java
/**...*/
@FunctionalInterface
public interface Predicate<T> {

    /**...*/
    boolean test(T t);

    /**...*/
    default Predicate<T> and(Predicate<? super T> other) {
        Objects.requireNonNull(other);
        return (t) -> test(t) && other.test(t);
    }

    /**...*/
    default Predicate<T> negate() { return (t) -> !test(t); }

    /**...*/
    default Predicate<T> or(Predicate<? super T> other) {
        Objects.requireNonNull(other);
        return (t) -> test(t) || other.test(t);
    }

    /**...*/
    static <T> Predicate<T> isEqual(Object targetRef) {
        return (null == targetRef)
                ? Objects::isNull
                : object -> targetRef.equals(object);
    }
}
```

BaseCondition

```java
public class BaseCondition<T> implements Condition<T> {
    public static final long serialVersionUID = 1L;
    protected boolean affirmative;

    public BaseCondition() { this( affirmative: true); }

    public BaseCondition(final boolean affirmative) { this.affirmative = affirmative; }

    @Override
    public boolean test(final T t) { return !affirmative; }

    @Override
    public boolean equals(final Object o) {
        if (this == o) {
            return true;
        }
        if (o == null || getClass() != o.getClass()) {
            return false;
        }
        final BaseCondition<?> that = (BaseCondition<?>)o;
        return affirmative == that.affirmative;
    }

    @Override
    public int hashCode() { return Objects.hash(affirmative); }
}
```

Actions & Functions

```
@FunctionalInterface
public interface Action<T> extends Serializable {
    void perform(T t);
}
```

Actions & Functions

```java
@FunctionalInterface
public interface Consumer<T> {

    /**...*/
    void accept(T t);

    /**...*/
    default Consumer<T> andThen(Consumer<? super T> after) {
        Objects.requireNonNull(after);
        return (T t) -> { accept(t); after.accept(t); };
    }
}
```

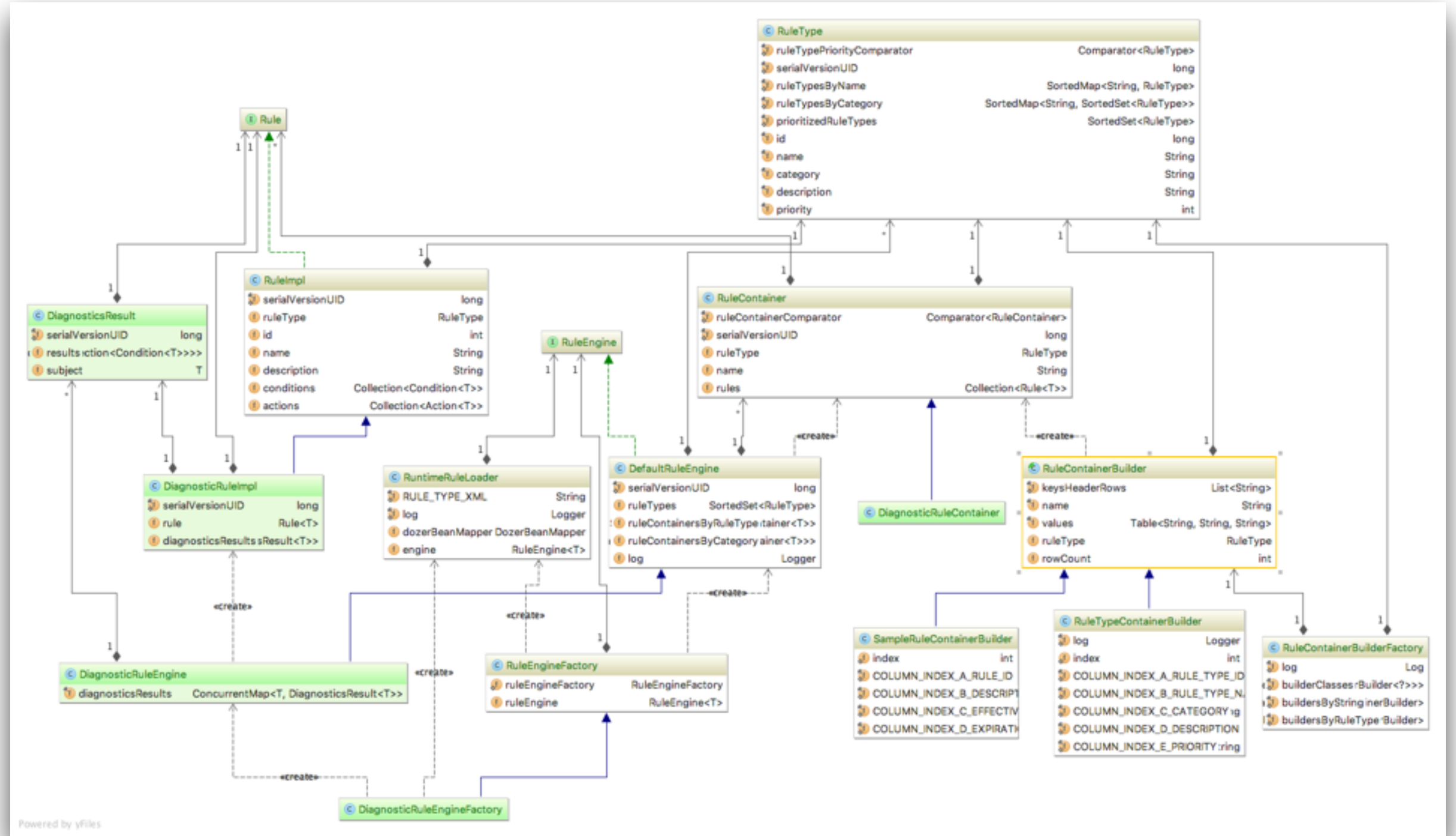
Actions & Functions

```java
/**...*/
@FunctionalInterface
public interface Function<T, R> {

    /**...*/
    R apply(T t);

    /**...*/
    default <V> Function<V, R> compose(Function<? super V, ? extends T> before) {
        Objects.requireNonNull(before);
        return (V v) -> apply(before.apply(v));
    }

    /**...*/
    default <V> Function<T, V> andThen(Function<? super R, ? extends V> after) {
        Objects.requireNonNull(after);
        return (T t) -> after.apply(apply(t));
    }

    /**...*/
    static <T> Function<T, T> identity() { return t -> t; }
}
```

Actions & Functions

```
public Function<Customer, Customer> applyDiscount = c -> c;
public Function<Customer, Customer> freeShipping = c -> c;

private Map<Predicate<Customer>, Function<Customer, Customer>> initializeActions() {
    final Map<Predicate<Customer>, Function<Customer, Customer>> result = new TreeMap<>();
    result.put(currentMedVip, applyDiscount);
    result.put(recentSales, freeShipping);
    return result;
}
private void applyActions() {
    final Map<Predicate<Customer>, Function<Customer, Customer>> actions = initializeActions();
    final List<Customer> customers = getCustomers();
    for (final Customer customer : customers) {
        actions.entrySet().stream().filter(e -> e.getKey().test(customer)).forEach(e -> e.getValue().apply(customer));
    }
}
```

Moderate Complexity



Moderate Complexity

- Small number of rules
- Number of objects in memory is large
- “Facts” known up front
- Custom Embedded Rule Engine
- Brute Force
- Not Rete (non-inference)
- In memory

Extreme Complexity

- Large number of rules and small number of objects in memory
- Use an inference engine with Rete Algorithm:
  - Open Source
    - drools.org ($)
    - Jess (JSR 94: Java Rule Engine API)
  - Commercial ($$$)
    - IBM WebSpere ILOG JRules
    - FICO Blaze Advisor
    - PegaSystems PegaRULES

Rules of (CERE) Rules

Guiding Principles

- **Immutable rules, conditions, actions**
- **First and final**
- **Avoid chaining**
- **Seek rule & rule set independence**
- **Truth tables**
- **Short-circuit conditions**

Features

- **Testing conditions**
- **Testing Actions**
- **Tracer capabilities**
  - **Right answer for the wrong reasons**

Testing Conditions

```
public boolean test(final Object o) {
    final LocalDateTime now = LocalDateTime.now();
    return null == effectiveDate && null == expirationDate
            ? affirmative : (!effectiveDate.isAfter(now) && !expirationDate.isBefore(now)) == affirmative;
}
```

```
public String toString() {
    String result = "now must ";
    if (effectiveDate != null && expirationDate != null) {
        if (effectiveDate.equals(expirationDate)) {
            return format("%sbe %s%s", result, affirmative ? "exactly " : "any date except ", effectiveDate);
        } else {
            return format("%s%sbe between %s and %s (inclusive)", result,
                    affirmative ? "" : "not ", effectiveDate, expirationDate);
        }
    } else {
        if (effectiveDate == null && expirationDate == null) {
            return "current date may be any value.";
        } else {
            return format("%s%sbe on or %s%s", result, affirmative ? "" : "not ",
                    effectiveDate == null ? "before " : "after ", effectiveDate);
        }
    }
}
```

Testing Conditions

```
public class IsEffectiveTest {
    public Condition<?> _default;
    public Condition<?> affirmative;
    public Condition<?> negative;
    private LocalDateTime effective;
    private LocalDateTime expires;

    @Before
    public void setUp() throws Exception {
        _default = new IsEffective();
        effective = LocalDateTime.now().minusDays(1L);
        expires = LocalDateTime.now().plusDays(1L);
        affirmative = new IsEffective(effective, expires, affirmative: true);
        negative = new IsEffective(effective, expires, affirmative: false);
    }

    @Test
    public void testNull() throws Exception {
        assertTrue( message: "expected true", _default.test( t: null));
        assertTrue( message: "expected true", affirmative.test( t: null));
        assertFalse( message: "expected false", negative.test( t: null));
    }

    @Test
    public void testToString() throws Exception {
        final String expectedDefault = "current date may be any value.";
        assertEquals( message: "expected match", expectedDefault, _default.toString());
        final String expectedAffirmative = format("now must be between %s and %s (inclusive)", effective, expires);
        assertEquals( message: "expected match", expectedAffirmative, affirmative.toString());
        final String expectedNegative = format("now must not be between %s and %s (inclusive)", effective, expires);
        assertEquals( message: "expected match", expectedNegative, negative.toString());
    }
}
```

Serialized Rules

- **XStream**
  - **JSON**
  - **XML**
  - **Java Serialization**
  - **Other**

RuntimeRuleLoader

```java
public class RuntimeRuleLoader {
    private RuleEngine engine;

    public RuntimeRuleLoader(final RuleEngine engine) { this.engine = engine; }

    public void initialize() {
        XStream xStream = new XStream(new PureJavaReflectionProvider());
        final Collection<RuleType> ruleTypes = (Collection<RuleType>)xStream
                .fromXML(getClass().getResourceAsStream( name: "/RuleType.xml"));
        if (null != ruleTypes) {
            ruleTypes.stream().flatMap(new Function<RuleType, Stream<Rule>>() {
                @Override
                public Stream<Rule> apply(final RuleType rt) {
                    return ((Collection<Rule>)xStream
                            .fromXML(getClass().getResourceAsStream( name: "/" + rt.name() + ".xml")))
                            .stream();
                }
            }).forEach(r -> engine.add(r.getRuleType(), r));
        }
    }
}
```

Rules In Excel

rules.xlsx

	A	B	C	D	E	F
1						
2	**rule type id**	**rule type**	**category**	**description**	**priority (processing order)**	
3						
4	1	sampleRules	sample	some sample rules	1	
5	2	otherRules	other	other rules that aren't defined	2	
6						
7						
8						

ruleTypes | sampleRules | otherRules

Sheet 1 of 3 | PageStyle_ruleTypes | Sum=0 | 75%

rules.xlsx

	A	B	C	D	E	F	G
1							
2	**rule id**	**description**	**effective date**	**expiration date**			
3							
4	1	This rule expired		2015-01-14 00:59:59.999			
5	2	This rule is in effect	2015-01-14 01:00:00.000				
6							
7							
8							

ruleTypes | sampleRules | otherRules

Sheet 2 of 3 | PageStyle_sampleRules | Sum=0 | 75%

RuleTypes to XML

rules.xlsx

	A	B	C	D	E	F
1						
2	**rule type id**	**rule type**	**category**	**description**	**priority (processing order)**	
3						
4	1	sampleRules	sample	some sample rules	1	
5	2	otherRules	other	other rules that aren't defined	2	
6						
7						
8						

ruleTypes | sampleRules | otherRules

Sheet 1 of 3 | PageStyle_ruleTypes | Sum=0 | 75%

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rule-types>
    <rule-type id="1" name="sampleRules" category="sample" description="some sample rules" priority="1"/>
    <rule-type id="2" name="otherRules" category="other" description="other rules that aren't defined" priority="2"/>
</rule-types>

```

Rules to XML

rules.xlsx

	A	B	C	D	E	F	G
1							
2	rule id	description	effective date	expiration date			
3							
4	1	This rule expired		2015-01-14 00:59:59.999			
5	2	This rule is in effect	2015-01-14 01:00:00.000				
6							
7							
8							

ruleTypes | **sampleRules** | otherRules

Sheet 2 of 3 | PageStyle_sampleRules | Sum=0 | 75%

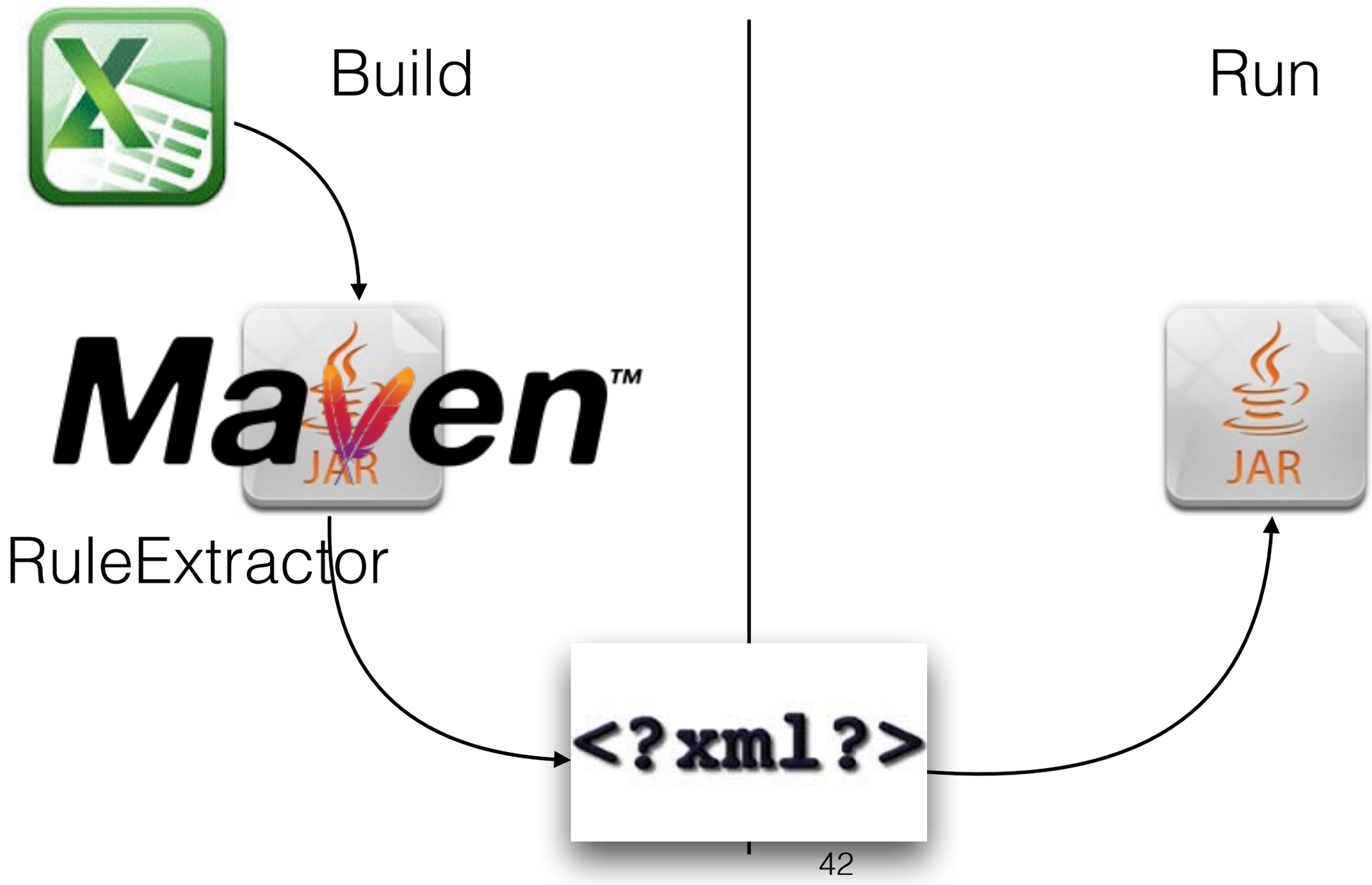
```xml
<?xml version="1.0" encoding="UTF-8"?>
<rules>
    <rule id="1" name="sampleRules" ruleType="sample" description="This rule expired">
        <actions/>
        <conditions>
            <effective affirmative="true" expires="2015-01-14T00:59:59.999-05:00"/>
        </conditions>
    </rule>
    <rule id="2" name="sampleRules" ruleType="sample" description="This rule is in effect">
        <actions/>
        <conditions>
            <effective affirmative="true" effective="2015-01-14T01:00:00.000-05:00"/>
        </conditions>
    </rule>
</rules>
```

Excel Rule Extractor

- Don't recommend reading directly from Excel at runtime in production.
- Parse Excel, transform into conditions, actions, rules, rule sets in memory.
- Serialize object graph to a suitable runtime storage format.
- Re-read serialized object graph for functional tests.
- Version controlled serialized object graph.

Maven Plugin

- **Integrates as part of build step**



Performance Considerations

- **Faster to be brute force and repeat the same conditions.**

- **Focus on fast-fail short-circuiting.**

- **Build rule object graph so more expensive Predicates are tested later.**

- **Organize rules into rule-sets so first winner skips remaining rules in the set.**

- **Avoid multiple potential winners if possible.**

Performance Cont.

- Parallelize Predicate test()
  - Defeats many short-circuit optimizations.
- Parallelize firing rules simultaneously on model objects.
- Parallelize firing rules of different rule sets on the same object simultaneously.
  - Defeats rule set precedences
- Possible to build an acyclic directed graph of rules connected by common Predicates.
  - …but now you're building an inference engine… don't do this!

References

http://www.martinfowler.com/bliki/RulesEngine.html

https://www.drools.org/

http://www.jessrules.com/

https://jcp.org/en/jsr/detail?id=94

http://www.sparklinglogic.com/category/business-rules/

https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html

https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html

https://docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html

EventsXD Survey

- 5 = Good Example
- 1 = Horrible Warning
- https://github.com/steveswing/cere