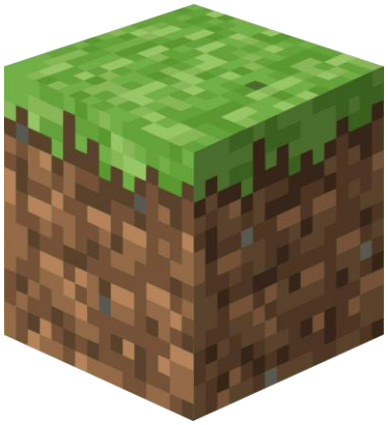
NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

Software Engineering Report

1st Semester 2024/2025

Minecraft World Edit



Nuno Duarte	60593
Patrícia Costa	62111
Guilherme Simões	62909
Daniel Agostinho	65330
Miguel Flor	65969
Nelson Martins	66081

Índice

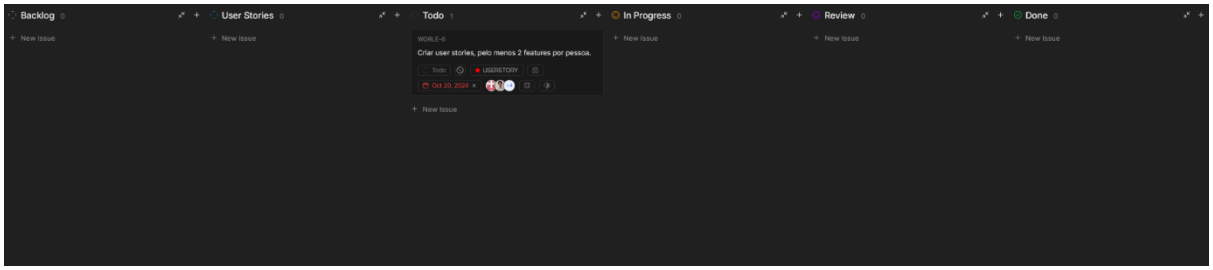
PROJECT MANAGEMENT

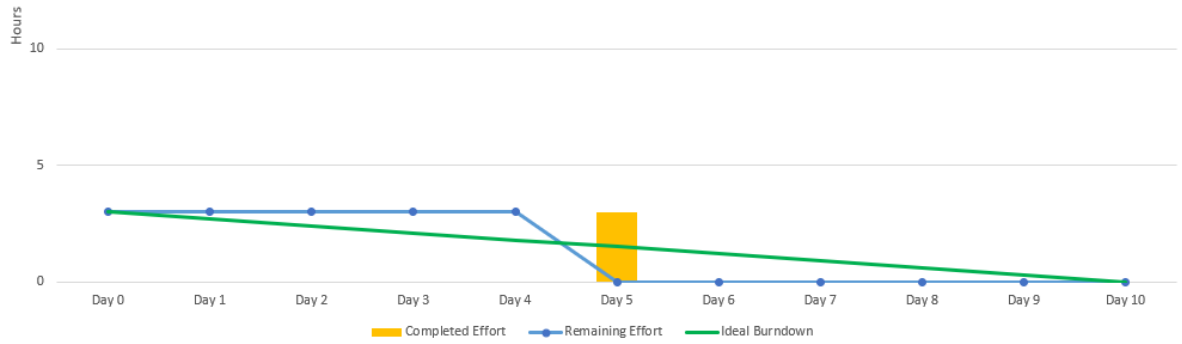
Scrum Master Identification

Week 1	Daniel Agostinho (65330)
Week 2	Miguel Flor (65969)
Week 3	Patrícia Costa (62111)
Week 4	Nelson Martins (66081)
Week 5	Guilherme Simões (62909)
Week 6	Nuno Duarte (60593)
Week 7	Miguel Flor (65969)
Week 8	Nelson Martins (66081)

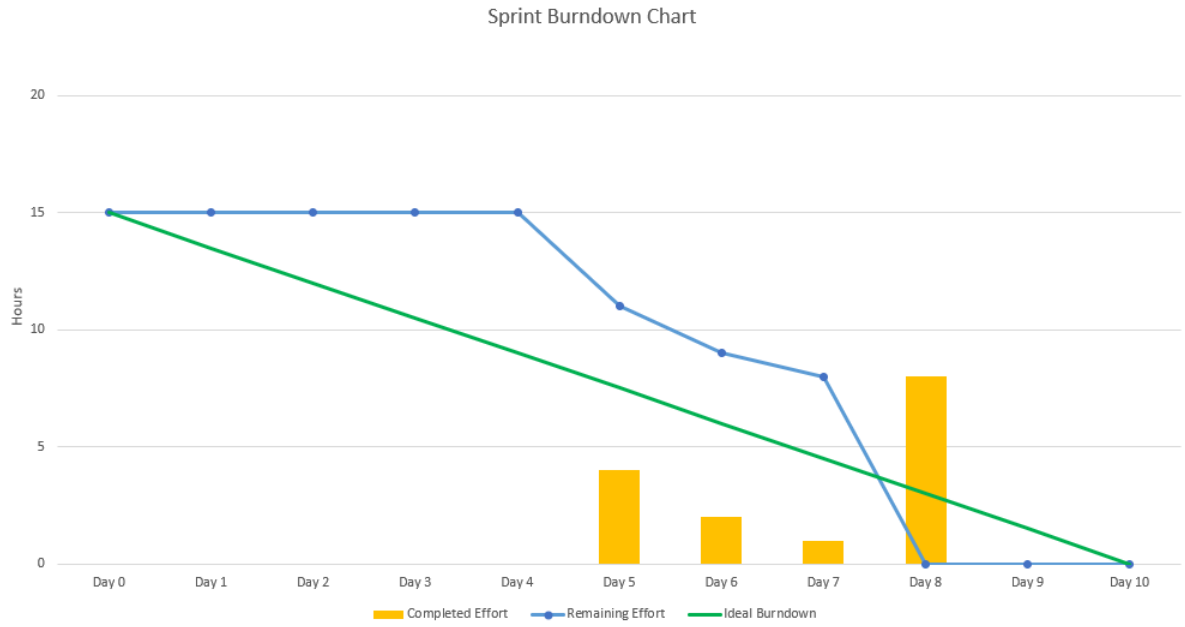
Burndown Chart and Scrum Board Evolution

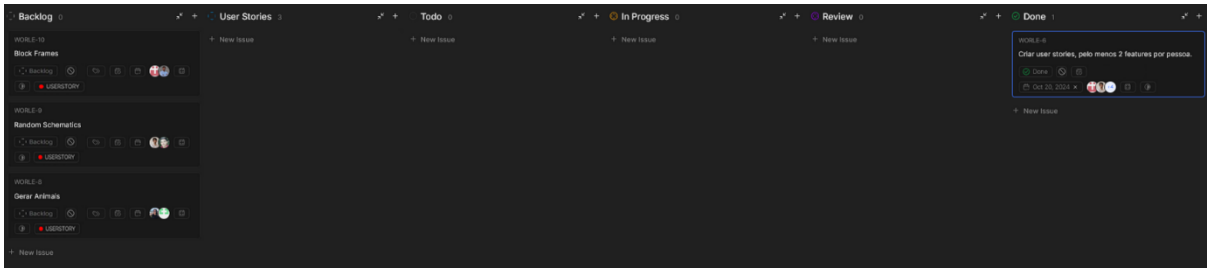
Week 1: Daniel Agostinho (65330)



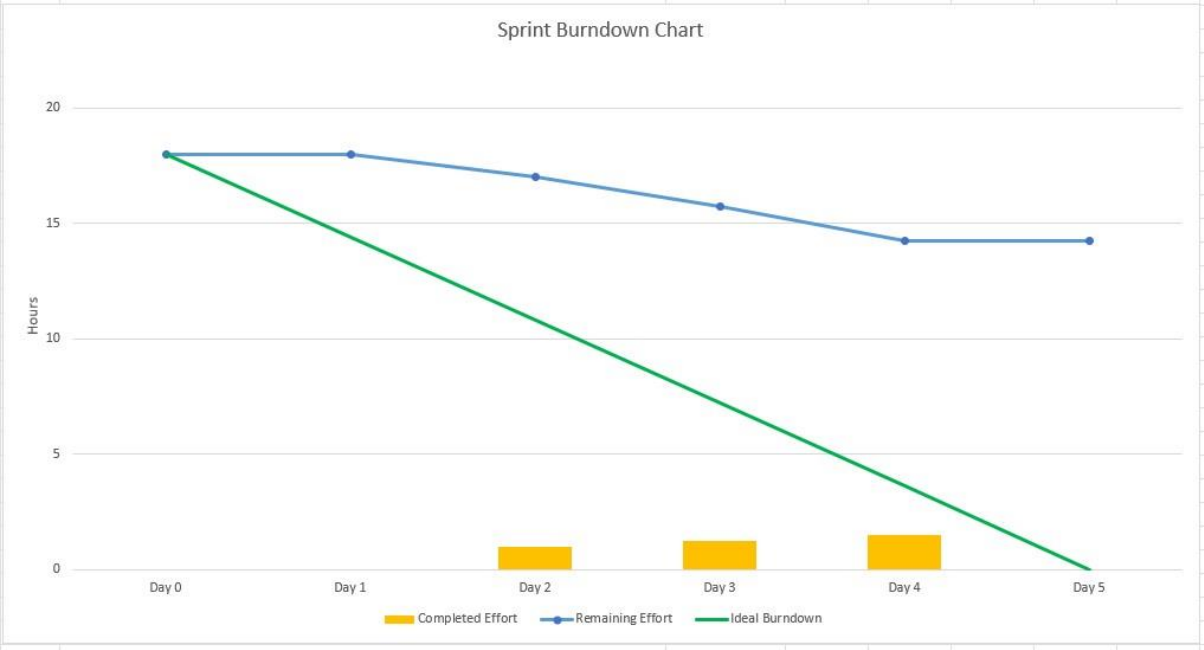


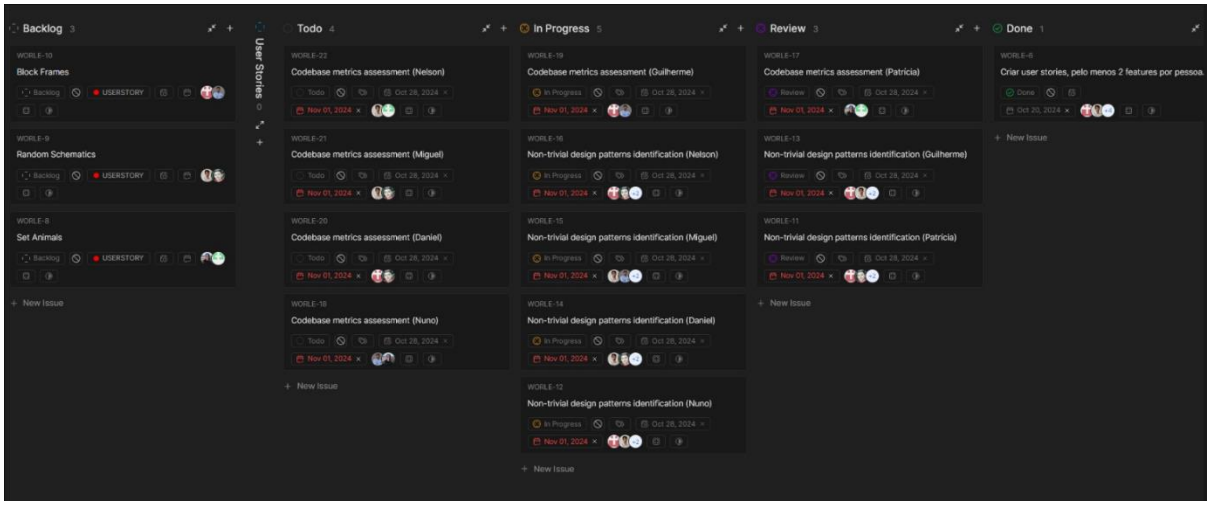
Week 2: Miguel Flor (65969)



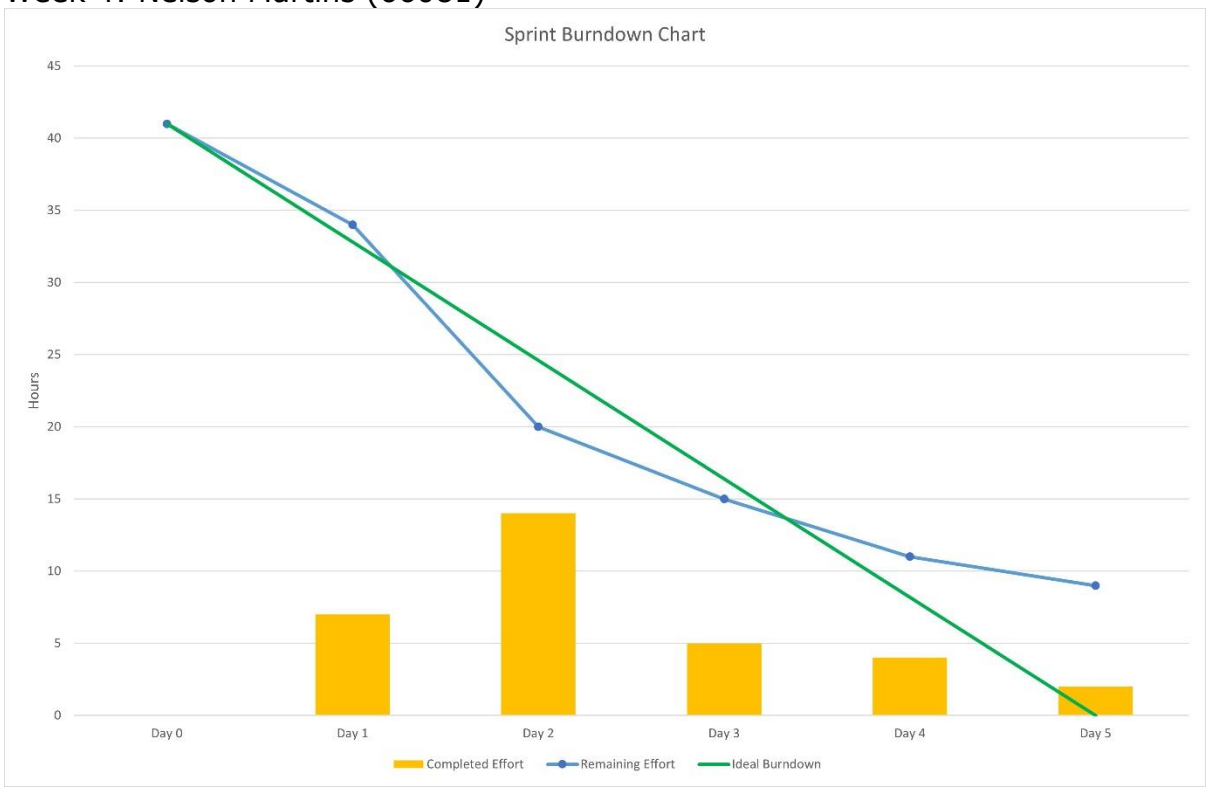


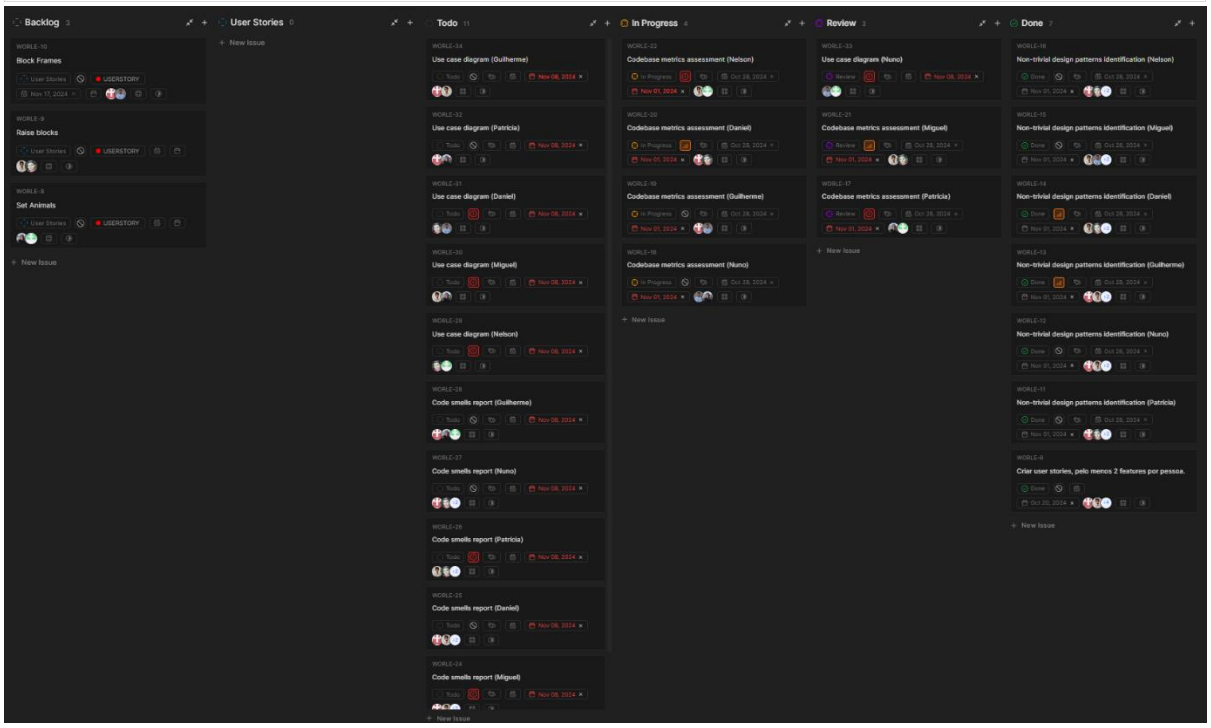
Week 3: Patrícia Costa (62111)



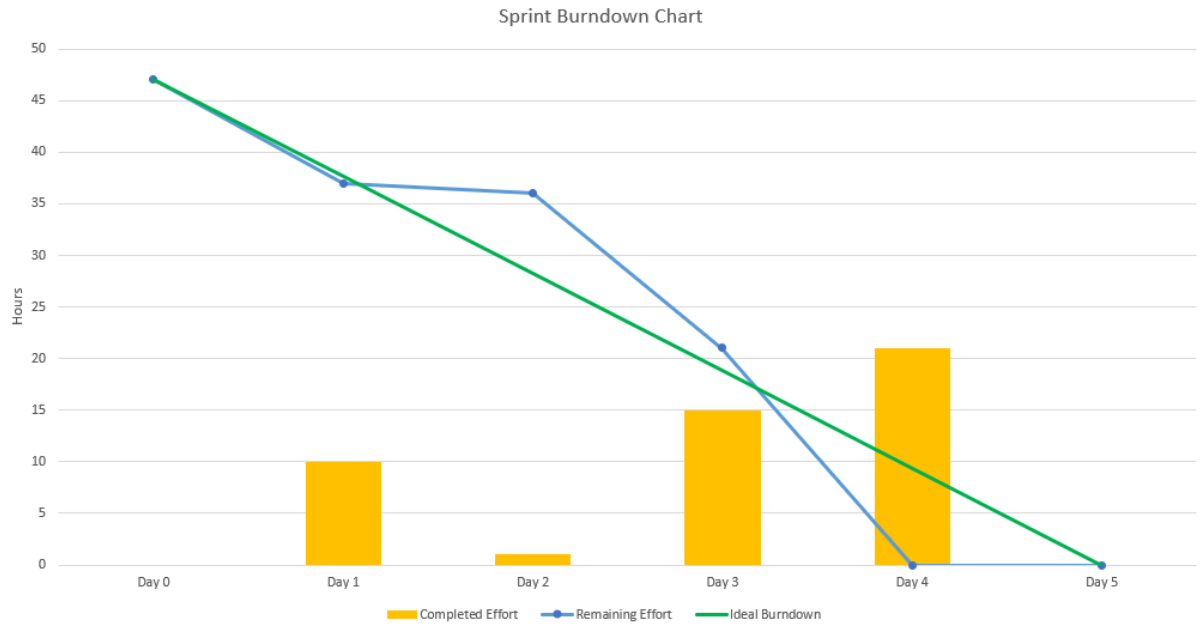


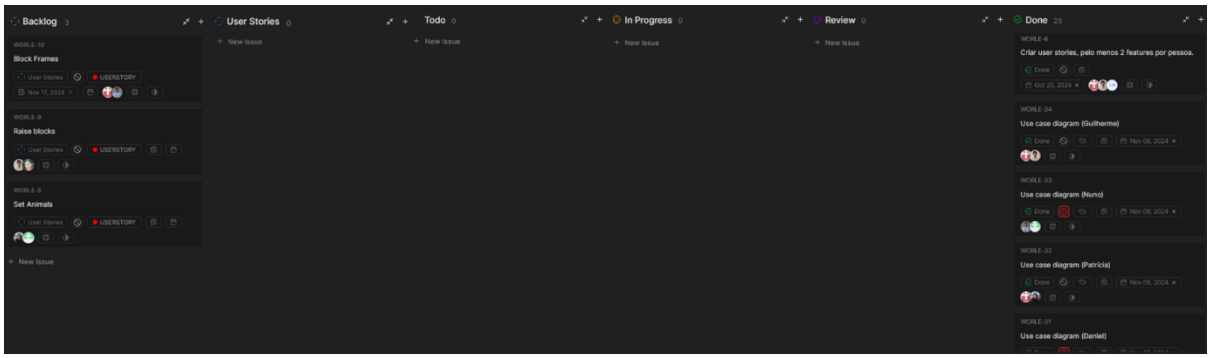
Week 4: Nelson Martins (66081)



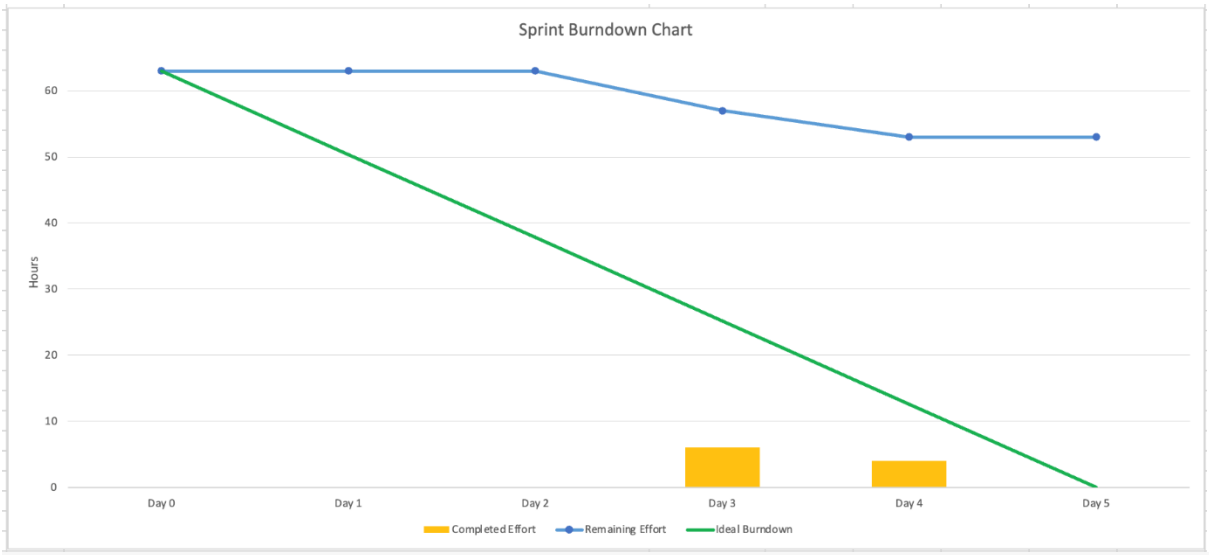


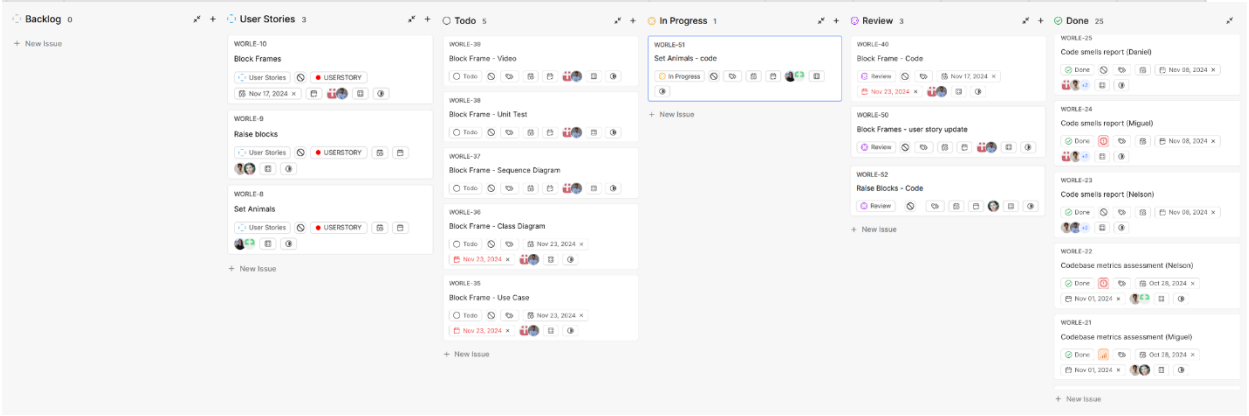
Week 5: Guilherme Simões (62909)


Sprint Burndown Chart
Hours
50
45
40
35
30
25
20
15
10
5
0
Day 0
Day 1
Day 2
Day 3
Day 4
Day 5
Completed Effort
Remaining Effort
Ideal Burndown

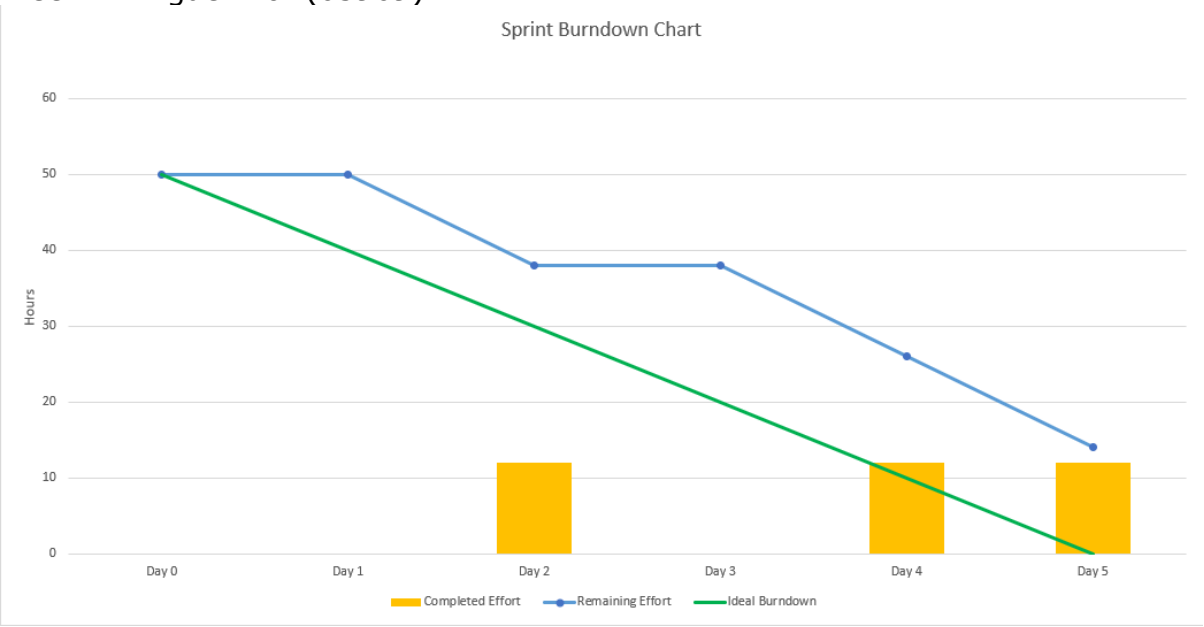

Backlog
User Stories
Todo
In Progress
Review
Done
Block Frames
Raise blocks
Set Animals
Criar user stories, pelo menos 2 features por pessoa.
Use case diagram (Guilherme)
Use case diagram (Nuno)
Use case diagram (Patricia)
Use case diagram (Daniel)

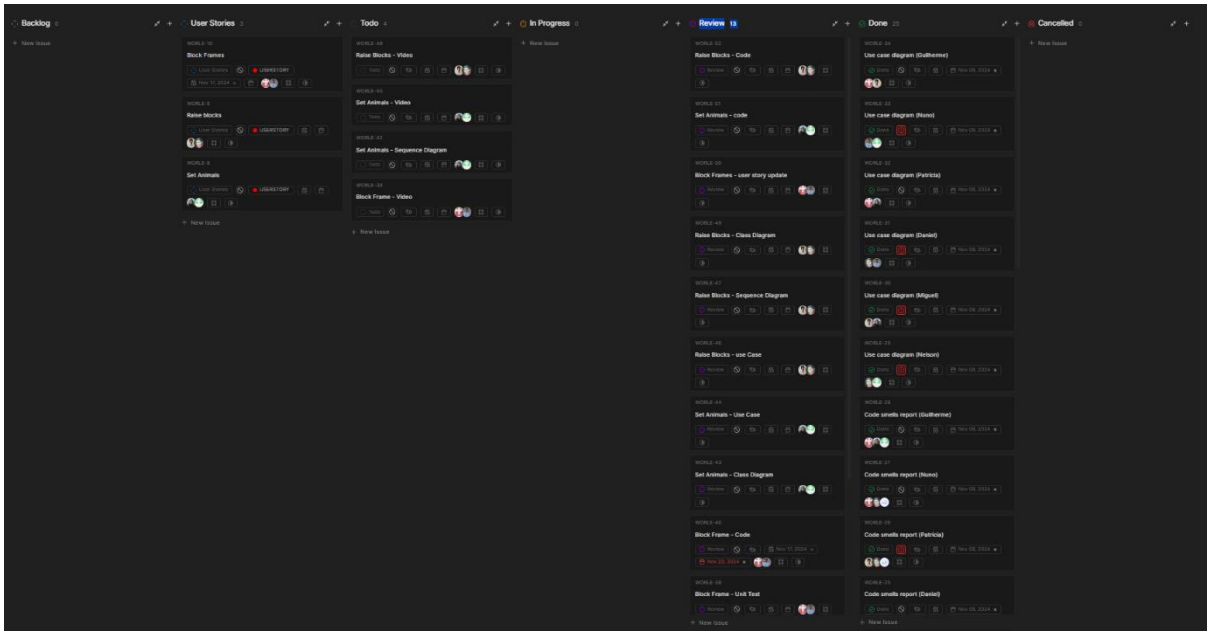
Week 6: Nuno Duarte (60593)



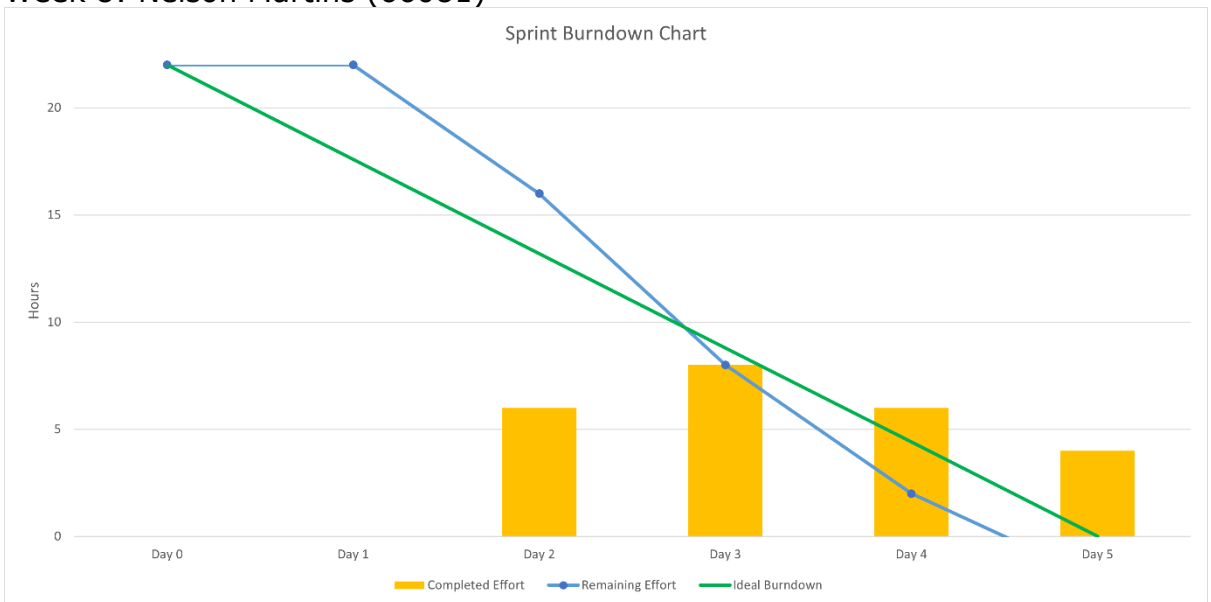


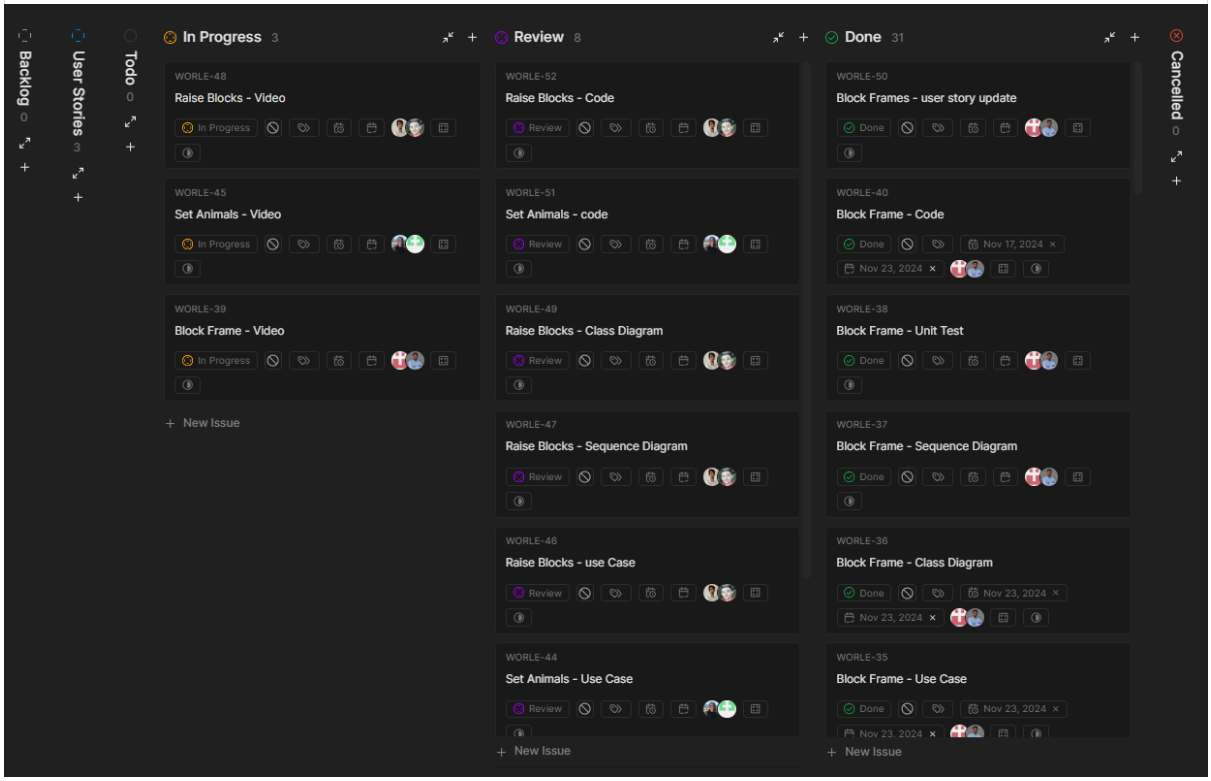
Week 7: Miguel Flor (65969)





Week 8: Nelson Martins (66081)





MILESTONE 1

User Stories Description

Frame-like Walls:

As an architect, **I want** a tool that fills the edges of a cuboid region and optionally fills specific faces, **so that** I can create structural frames, such as for windows, doors, and decorative trims, while retaining the flexibility to design additional face features as needed.
[Nuno Duarte 60593; Guilherme Simões 62909]

Raise Blocks:

As a builder, **I want** to have a tool that allows me to raise blocks to a certain height, in a given area, **so that** I can build walls and other structures more efficiently.
[Daniel Agostinho 65330; Miguel Flor 65969]

Set Animals:

As an animal lover, **I want** a tool that allows me to spawn and place specific animals in a selected area, **so that** I can create themed environments or ecosystems with precise control over animal placement and population density.
[Patrícia Costa 62111; Nelson Martins 66081]

MILESTONE 2

Non-trivial Design Patterns

Singleton [identification: 62111; review: 60593]

Lines 118, 140 and 153 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/WorldEdit.java
---

```
  private static final WorldEdit instance = new WorldEdit();
private WorldEdit() {
    eventBus.register(new SchematicsEventListener());
}
public static WorldEdit getInstance() {
    return instance;
}
```

Explanation:

The Singleton pattern is identifiable here because WorldEdit restricts instantiation to a single object accessible across the application.
This is achieved through a **private static instance** and a **private constructor** that prevents additional instantiations.
The getInstance() method ensures that any part of the code accessing WorldEdit will use this single instance.
Singleton is ideal here because WorldEdit manages game-wide resources and settings, and multiple instances could cause inconsistencies or unexpected behavior across the application.
On the other hand, note that this implementation *lacks lazy initialization*, which could cause the program to be less efficient.

Strategy [identification: 62111; review: 62909]

Line 38 of the file worldedit-core/src/main/java/com/sk89q/worldedit/function/mask/Mask.java

---

```
boolean test(BlockVector3 vector);
```

Line 96 of the file worldedit-core/src/main/java/com/sk89q/worldedit/function/mask/BlockMask.java

---

```
@Override
public boolean test(BlockVector3 vector) {
    BlockState block = getExtent().getBlock(vector);
    for (BaseBlock testBlock : blocks) {
        if (testBlock.equalsFuzzy(block)) {
            return true;
        }
    }

    return false;
}
```

Line 93 of the file worldedit-core/src/main/java/com/sk89q/worldedit/function/mask/BiomeMask.java

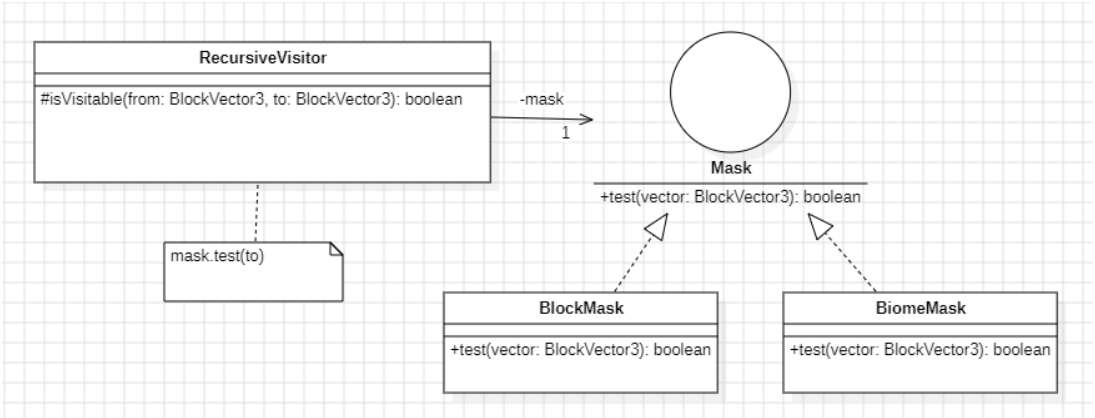
---

```
@Override
public boolean test(BlockVector3 vector) {
    BiomeType biome = extent.getBiome(vector);
    return biomes.contains(biome);
}
```

Lines 49 of the file worldedit-core/src/main/java/com/sk89q/worldedit/function/visitor/RecursiveVisitor.java

---

```java
@Override
protected boolean isVisitable(BlockVector3 from, BlockVector3 to) {
    return mask.test(to);
}
```



Explanation:

This implementation demonstrates the **Strategy pattern** by allowing RecursiveVisitor to act as the **context** that delegates the decision-making for block visitation to the Mask interface, which acts as a **strategy**. By using different Mask **implementations** (e.g. BlockMask, BiomeMask), RecursiveVisitor can apply various criteria for determining if a block should be visited without changing its own logic.
This setup allows RecursiveVisitor to adapt its behavior dynamically by swapping out Mask **strategies**, making it flexible and easy to extend with new visitation criteria.

Observer [identification: 62111; review: 65330]

Lines 54, 88, 120, 140, 150, 161 and 191 of the file worldedit-core/src/main/java/com/sk89q/worldedit/util/eventbus/EventBus.java

---

```java
private final SetMultimap<Class<?>, EventHandler> handlersByType =
    HashMultimap.create();
```

```java
public void subscribeAll(Multimap<Class<?>, EventHandler> handlers) {
    checkNotNull(handlers);
    lock.writeLock().lock();
    try {
        handlersByType.putAll(handlers);
    } finally {
        lock.writeLock().unlock();
    }
}
```

```java
public void unsubscribeAll(Multimap<Class<?>, EventHandler> handlers) {
    checkNotNull(handlers);
    lock.writeLock().lock();
    try {
        for (Map.Entry<Class<?>, Collection<EventHandler>> entry : handlers.asMap().entrySet()) {
            handlersByType.get(entry.getKey()).removeAll(entry.getValue());
        }
    } finally {
        lock.writeLock().unlock();
    }
}
```

```java
public void register(Object object) {
    subscribeAll(finder.findAllSubscribers(object));
}
```

```java
public void unregister(Object object) {
    unsubscribeAll(finder.findAllSubscribers(object));
}
```

```java
public void post(Object event) {
    List<EventHandler> dispatching = new ArrayList<>();

    Set<Class<?>> dispatchTypes = flattenHierarchyCache.get(event.getClass());
    lock.readLock().lock();
    try {
        for (Class<?> eventType : dispatchTypes) {
            Set<EventHandler> wrappers = handlersByType.get(eventType);

            if (wrappers != null && !wrappers.isEmpty()) {
                dispatching.addAll(wrappers);
            }
        }
    } finally {
        lock.readLock().unlock();
    }

    Collections.sort(dispatching);

    for (EventHandler handler : dispatching) {
        dispatch(event, handler);
    }
}
private void dispatch(Object event, EventHandler handler) {
    try {
        handler.handleEvent(event);
    } catch (InvocationTargetException e) {
        LOGGER.error( message: "Could not dispatch event: " + event + " to handler " + handler, e);
    }
}
```

Lines 71 and 85 of the file worldedit-core/src/main/java/com/sk89q/worldedit/util/eventbus/EventHandler.java

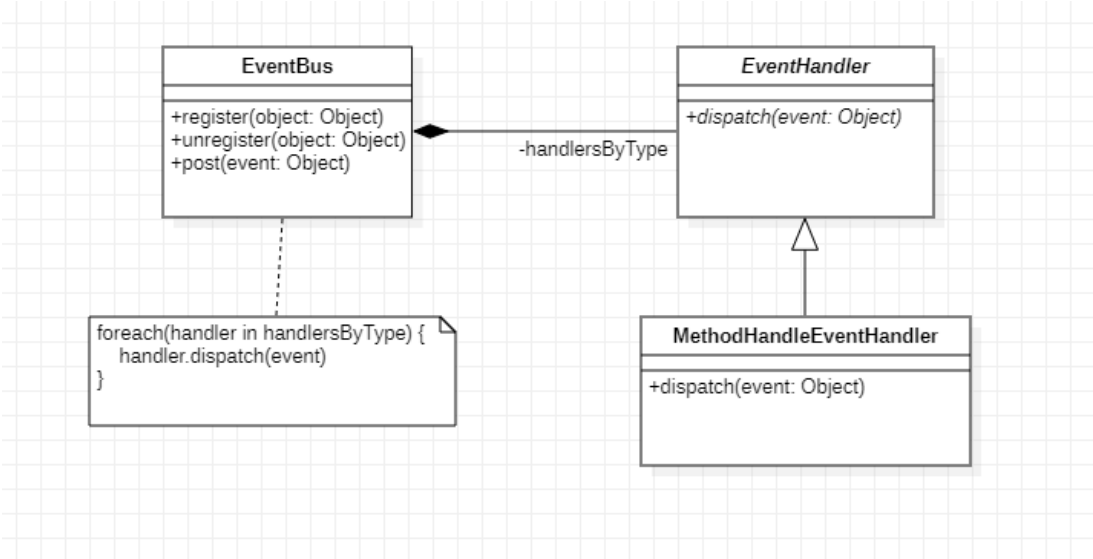
---

```java
public final void handleEvent(Object event) throws InvocationTargetException {
    try {
        dispatch(event);
    } catch (Throwable t) {
        throw new InvocationTargetException(t);
    }
}

public abstract void dispatch(Object event) throws Exception;
```

Line 49 of the file worldedit-core/src/main/java/com/sk89q/worldedit/util/eventbus/MethodHandleEventHandler.java

---

```java
@Override
public void dispatch(Object event) throws Exception {
    try {
        this.methodHandle.invokeExact(event);
    } catch (Exception | Error e) {
        throw e;
    } catch (Throwable t) {
        // If it's not an Exception or Error, throw it wrapped.
        throw new Exception(t);
    }
}
```



Explanation:

This implementation demonstrates the Observer pattern by allowing EventBus to act as the **Subject** that notifies multiple observers (EventHandler instances) about specific events.
The EventHandler class, as an **abstract observer**, defines how events should be processed, while the MethodHandleEventHandler provides a **concrete implementation** that uses dynamic invocation to handle events.
This structure supports flexibility and modularity, allowing various observers to respond to events in their own ways without modifying the EventBus.

Template Method [identification: 62909; review: 65969]

Lines 94-105 of the file worldedit-core/src/main/java/com/sk89q/worldedit/command/argument/AbstractDirectionConverter.java

---

```
@Override  wizjany +1 *
public ConversionResult<D> convert(String argument, InjectedValueAccess context) {
    Player player = context.injectedValue(Key.of(Player.class, OptionalArg.class))
        .orElse( other: null);
    try {
        return SuccessfulConversion.fromSingle(convertDirection(argument,
                player, includeDiagonals));
    } catch (Exception e) {
        return FailedConversion.from(e);
    }
}

protected abstract D convertDirection(String argument, @Nullable Player player,
                                      boolean includeDiagonals)
        throws UnknownDirectionException;
```

Lines 45-52 of the file worldedit-core/src/main/java/com/sk89q/worldedit/command/argument/DirectionConverter.java

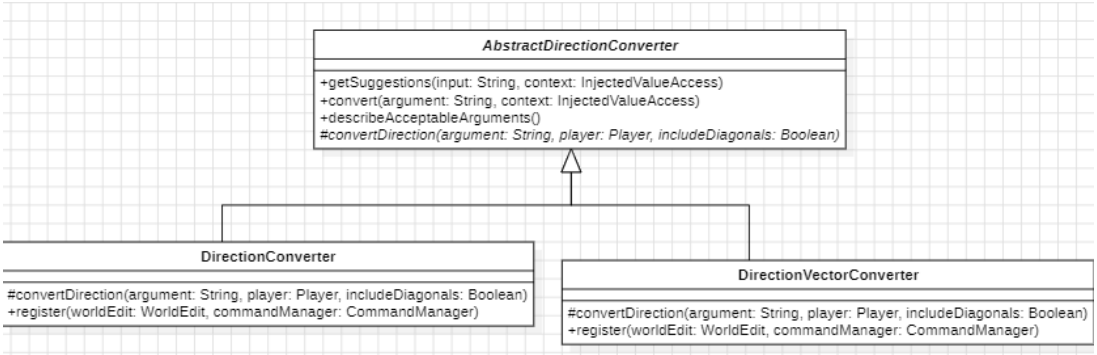
---

```
@Override  1 usage    wizjany *
protected Direction convertDirection(String argument, @Nullable Player player,
                                     boolean includeDiagonals)
        throws UnknownDirectionException {
    final BlockVector3 vec = includeDiagonals
            ? getWorldEdit().getDiagonalDirection(player, argument)
            : getWorldEdit().getDirection(player, argument);
    return Optional.ofNullable(Direction.findClosest
                    (vec.toVector3(), Direction.Flag.ALL))
            .orElseThrow(() -> new UnknownDirectionException(argument));
}
```

Lines 45-52 of the file worldedit-core/src/main/java/com/sk89q/worldedit/command/argument/DirectionVectorConverter.java

---

```java
@Override  1 usage    wizjany *
protected BlockVector3 convertDirection(String argument, @Nullable Player player,
                                        boolean includeDiagonals)
        throws UnknownDirectionException {
    return includeDiagonals
            ? getWorldEdit().getDiagonalDirection(player, argument)
            : getWorldEdit().getDirection(player, argument);
}
```



Explanation:

The abstract class AbstractDirectionConverter defines the template structure of the conversion process, outlining the general steps required to convert a direction argument. The DirectionVectorConverter and DirectionConverter classes override convertDirection to provide concrete behavior specific to their type of direction conversion.

Iterator Pattern [identification: 62909; review: 62111]

Lines 102-127 and 132-157 of the file worldedit-core/src/main/java/com/sk89q/worldedit/util/collection/DoubleArrayList.java

---

```java
public class ForwardEntryIterator<T extends Map.Entry<A, B>> 1 usage  sk89q +1
        implements Iterator<Map.Entry<A, B>> {

    private final Iterator<A> keyIterator;  3 usages
    private final Iterator<B> valueIterator;  2 usages

    public ForwardEntryIterator(Iterator<A> keyIterator, Iterator<B> valueIterator) {
        this.keyIterator = keyIterator;
        this.valueIterator = valueIterator;
    }

    @Override  sk89q
    public boolean hasNext() { return keyIterator.hasNext(); }

    @Override  sk89q
    public Map.Entry<A, B> next() throws NoSuchElementException {
        return new Entry<A, B>(keyIterator.next(), valueIterator.next());
    }

    @Override  sk89q
    public void remove() { throw new UnsupportedOperationException(); }
}
```

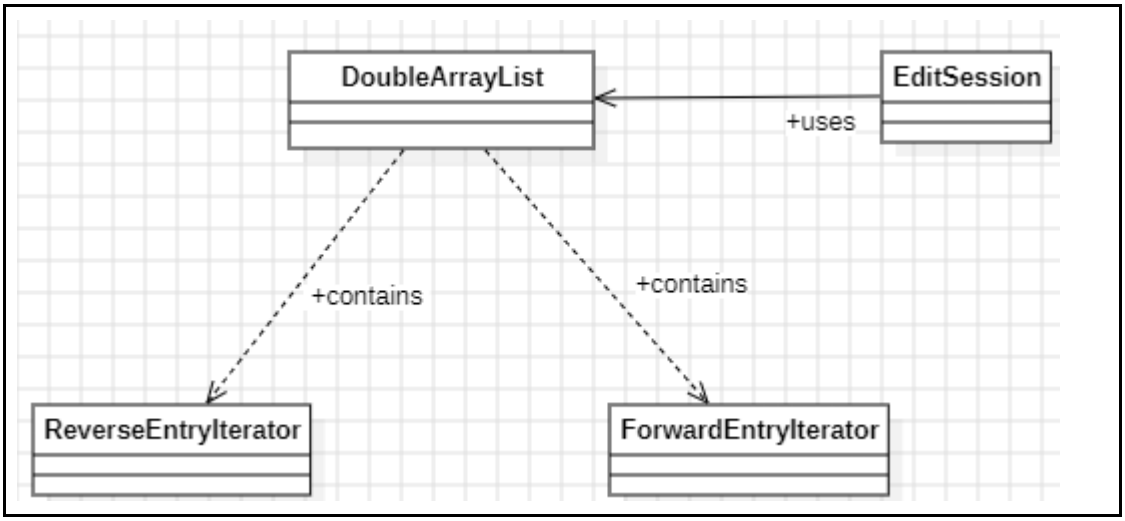
```java
public class ReverseEntryIterator<T extends Map.Entry<A, B>> 1 usage  sk89q +1
        implements Iterator<Map.Entry<A, B>> {

    private final ListIterator<A> keyIterator;  3 usages
    private final ListIterator<B> valueIterator;  2 usages

    public ReverseEntryIterator(ListIterator<A> keyIterator, ListIterator<B> valueIterator) {
        this.keyIterator = keyIterator;
        this.valueIterator = valueIterator;
    }

    @Override  sk89q
    public boolean hasNext() { return keyIterator.hasPrevious(); }

    @Override  sk89q
    public Map.Entry<A, B> next() throws NoSuchElementException {
        return new Entry<A, B>(keyIterator.previous(), valueIterator.previous());
    }

    @Override  sk89q
    public void remove() { throw new UnsupportedOperationException(); }
}
```

Lines 2436-2481 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java

---

```java
public int deformRegion(final Region region, final Vector3 zero, final Vector3 unit, final Expression expression,
                        final int timeout) throws ExpressionException, MaxChangedBlocksException {
    final Variable x = expression.getSlots().getVariable( name: "x")
        .orElseThrow(IllegalStateException::new);
    final Variable y = expression.getSlots().getVariable( name: "y")
        .orElseThrow(IllegalStateException::new);
    final Variable z = expression.getSlots().getVariable( name: "z")
        .orElseThrow(IllegalStateException::new);

    final WorldEditExpressionEnvironment environment = new WorldEditExpressionEnvironment( extent: this, unit, zero);
    expression.setEnvironment(environment);

    final DoubleArrayList<BlockVector3, BaseBlock> queue = new DoubleArrayList<>( isReversed: false);

    for (BlockVector3 targetBlockPosition : region) {
        final Vector3 targetPosition = targetBlockPosition.toVector3();
        environment.setCurrentBlock(targetPosition);

        // offset, scale
        final Vector3 scaled = targetPosition.subtract(zero).divide(unit);

        // transform
        expression.evaluate(new double[]{ scaled.x(), scaled.y(), scaled.z() }, timeout);

        final BlockVector3 sourcePosition = environment.toWorld(x.value(), y.value(), z.value());
```

```java
        // read block from world
        final BaseBlock material = world.getFullBlock(sourcePosition);

        // queue operation
        queue.put(targetBlockPosition, material);
    }

    int affected = 0;
    for (Map.Entry<BlockVector3, BaseBlock> entry : queue) {
        BlockVector3 position = entry.getKey();
        BaseBlock material = entry.getValue();

        // set at new position
        if (setBlock(position, material)) {
            ++affected;
        }
    }

    return affected;
}
```



Explanation:

The DoubleArrayList class relates to the Iterator Pattern in several ways, particularly in how it allows iteration over its elements through the ForwardEntryIterator and ReverseEntryIterator classes. The deformRegion method effectively utilizes this pattern to manage and apply changes to a region.

Abstract Factory [identification: 62909; review: 65330]

Lines 53-58 of the file worldedit-core/src/main/java/com/sk89q/worldedit/internal/registry/AbstractFactory.java

---

```
protected AbstractFactory(WorldEdit worldEdit, InputParser<E> defaultParser) {
    checkNotNull(worldEdit);
    checkNotNull(defaultParser);
    this.worldEdit = worldEdit;
    this.parsers.add(defaultParser);
}
```

Lines 46-57 of the file worldedit-core/src/main/java/com/sk89q/worldedit/extension/factory/PatternFactory.java

---

```java
public PatternFactory(WorldEdit worldEdit) {  1 usage  wizjany +1
    super(worldEdit, new SingleBlockPatternParser(worldEdit));

    // split and parse each sub-pattern
    register(new RandomPatternParser(worldEdit));

    // individual patterns
    register(new ClipboardPatternParser(worldEdit));
    register(new TypeOrStateApplyingPatternParser(worldEdit));
    register(new RandomStatePatternParser(worldEdit));
    register(new BlockCategoryPatternParser(worldEdit));
}
```

Lines 65-82 of file worldedit-core/src/main/java/com/sk89q/worldedit/extension/factory/MaskFactory.java

---

```java
public MaskFactory(WorldEdit worldEdit) {  1 usage  Mat
    super(worldEdit, new BlocksMaskParser(worldEdit));

    register(new ExistingMaskParser(worldEdit));
    register(new AirMaskParser(worldEdit));
    register(new ExposedMaskParser(worldEdit));
    register(new SolidMaskParser(worldEdit));
    register(new LazyRegionMaskParser(worldEdit));
    register(new RegionMaskParser(worldEdit));
    register(new OffsetMaskParser(worldEdit));
    register(new NoiseMaskParser(worldEdit));
    register(new BlockStateMaskParser(worldEdit));
    register(new NegateMaskParser(worldEdit));
    register(new ExpressionMaskParser(worldEdit));

    register(new BlockCategoryMaskParser(worldEdit));
    register(new BiomeMaskParser(worldEdit));
}
```

Lines 34-36 of file worldedit-core/src/main/java/com/sk89q/worldedit/extension/factory/ItemFactory.java

---

```
  public ItemFactory(WorldEdit worldEdit) {  1 usage  sk89q
      super(worldEdit, new DefaultItemParser(worldEdit));
  }
```

Lines 47-49 of file worldedit-core/src/main/java/com/sk89q/worldedit/extension/factory/BlockFactory.java

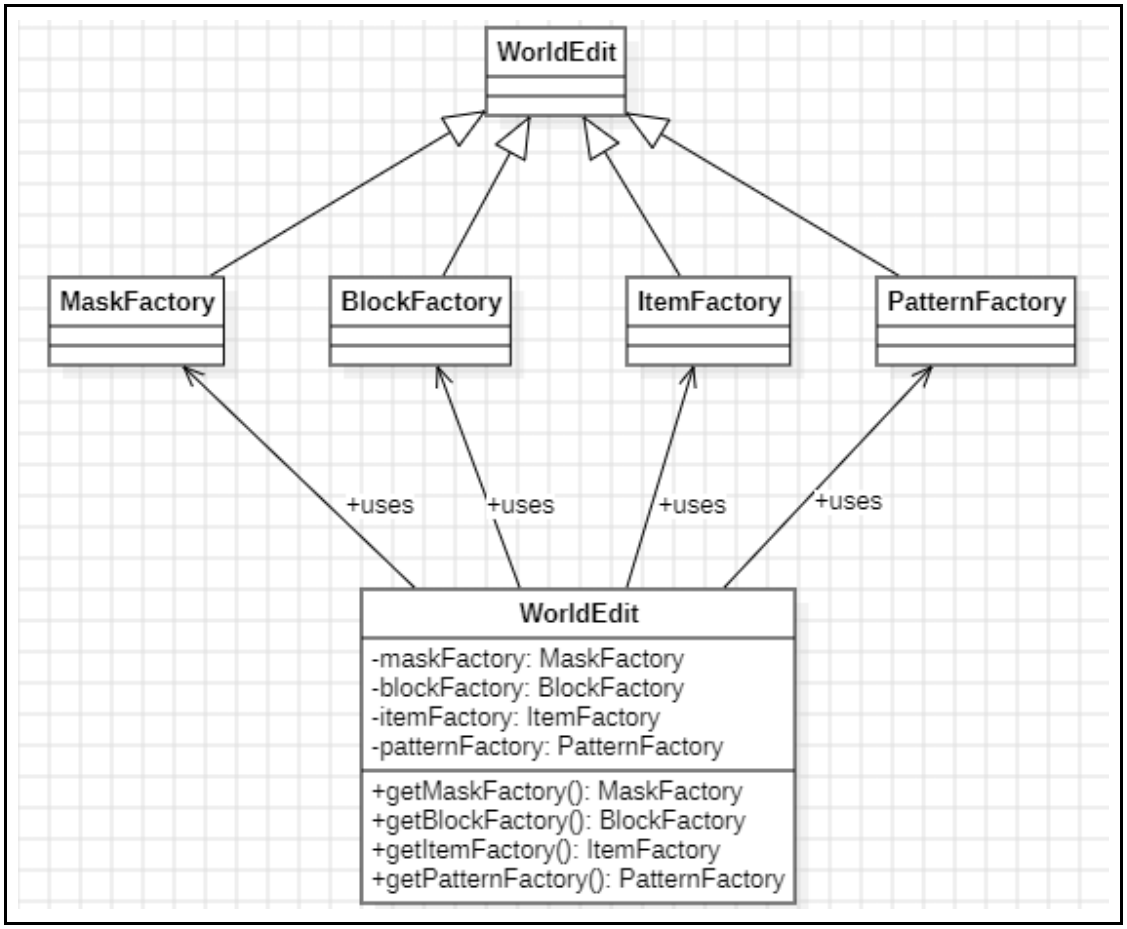
---

```
public BlockFactory(WorldEdit worldEdit) {  1 usage  sk89q
    super(worldEdit, new DefaultBlockParser(worldEdit));
}
```

Lines 193-231 of file worldedit-core/src/main/java/com/sk89q/worldedit/WorldEdit.java

---

```
/** Get the block factory from which new {@link BlockStateHolder}s can be ...*/
public BlockFactory getBlockFactory() { return blockFactory; }

/** Get the item factory from which new {@link BaseItem}s can be ...*/
public ItemFactory getItemFactory() { return itemFactory; }

/** Get the mask factory from which new {@link Mask}s ...*/
public MaskFactory getMaskFactory() {  7 usages  sk89q
    return maskFactory;
}

/** Get the pattern factory from which new {@link Pattern}s ...*/
public PatternFactory getPatternFactory() {  6 usages  sk89q
    return patternFactory;
}
```



Explanation:

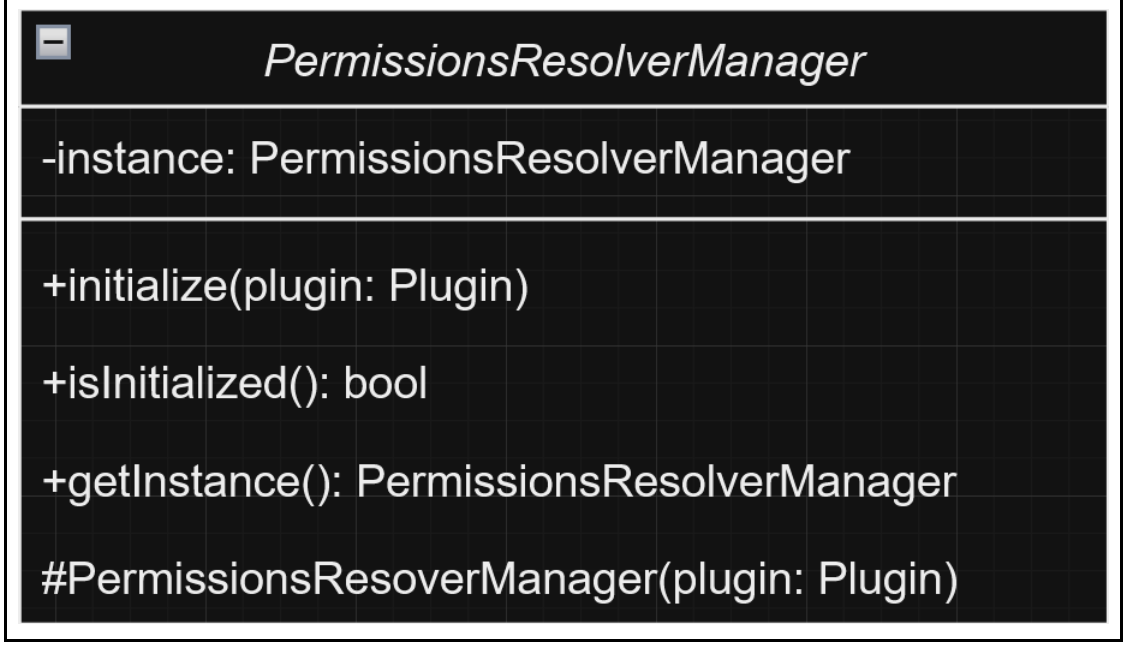
Given that the structure allows for the creation of various types of related objects through multiple parser methods (as seen in AbstractFactory), this design fits the Abstract Factory Pattern.

Singleton [identification: 65330; review: 62111]

Line 63-80 and 99-105 of the file worldedit-bukkit/src/main/java/com/sk89q/wepif/PermissionsResolverManager.java
---

```java
private static PermissionsResolverManager instance;  3 usages

public static void initialize(Plugin plugin) {  1 usage  zml2008
    if (!isInitialized()) {
        instance = new PermissionsResolverManager(plugin);
    }
}

public static boolean isInitialized() { return instance != null; }

public static PermissionsResolverManager getInstance() {  zml2008 +1
    if (!isInitialized()) {
        throw new WEPIFRuntimeException("WEPIF has not yet been initialized!");
    }
    return instance;
}
```

```java
protected PermissionsResolverManager(Plugin plugin) {  1 usage  zml2008
    this.server = plugin.getServer();
    (new ServerListener()).register(plugin); // Register the events

    loadConfig(new File( pathname: "wepif.yml"));
    findResolver();
}
```



Explanation:

This implementation demonstrates the Singleton pattern by ensuring that only one instance of PermissionsResolverManager exists throughout the application. The PermissionsResolverManager class contains a private static variable instance that holds the single instance of the class. The initialize method checks if the instance is already created using the isInitialized method, and if not, it creates a new instance. The getInstance method provides access to the single instance, throwing an exception if it has not been initialized. The protected constructor PermissionsResolverManager ensures that the class cannot be instantiated from outside. This prevents the creation of multiple instances and ensures that the same instance is used across the application. This design pattern is useful for managing shared resources or configurations in a centralized manner.

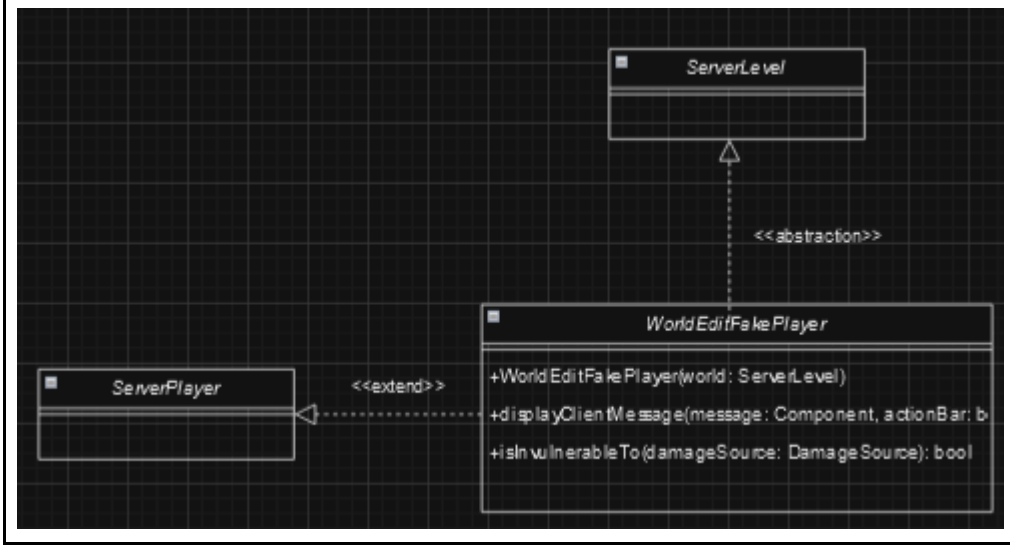
Adapter [identification: 65330; review: 65969]

Lines 40-42 and 56-64 of the file worldedit-fabric/src/main/java/com/sk89q/worldedit/fabric/WorldEditFakePlayer.java

---

```
public WorldEditFakePlayer(ServerLevel world) {  1 usage  Octavia Togami
    super(world.getServer(), world, FAKE_WORLDEDIT_PROFILE, FAKE_CLIENT_INFO);
}
```

```
@Override  Octavia Togami
public void displayClientMessage(Component message, boolean actionBar) {
    super.displayClientMessage(message, actionBar);
}

@Override  Matthew Miller
public boolean isInvulnerableTo(DamageSource damageSource) {
    return true;
}
```



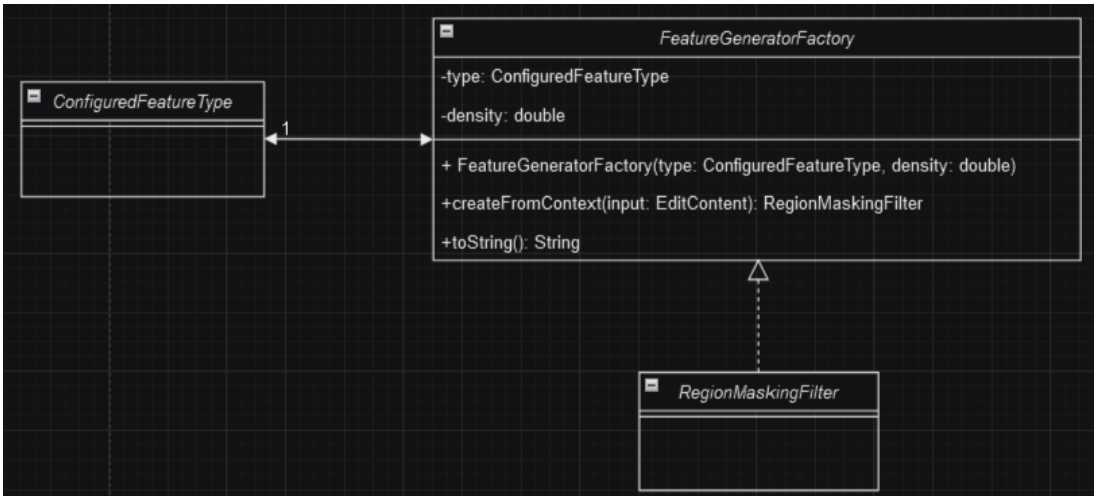
Explanation:

This implementation demonstrates the **Adapter pattern** by allowing WorldEditFakePlayer to act as an adapter between the ServerPlayer class and the WorldEdit functionality. The WorldEditFakePlayer class extends ServerPlayer and overrides specific methods to adapt the behavior to fit the needs of WorldEdit.
For example, the displayClientMessage method is overridden to display messages to the client, and the isInvulnerableTo method is overridden to make the player invulnerable to certain damage sources. This allows the WorldEditFakePlayer to integrate seamlessly with the existing ServerPlayer class while providing additional functionality required by WorldEdit.
By using the Adapter pattern, the WorldEditFakePlayer can leverage the existing ServerPlayer functionality and extend it without modifying the original class. This promotes code reuse and flexibility, making it easier to maintain and extend the application.

Factory [identification: 65330; review: 66081]

Lines 31-52 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/command/factory/FeatureGeneratorFactory.java
---

```java
public final class FeatureGeneratorFactory implements Contextual<RegionMaskingFilter> {
    private final ConfiguredFeatureType type;  3 usages
    private final double density;  2 usages

    public FeatureGeneratorFactory(ConfiguredFeatureType type, double density) {  1 usage
        this.type = type;
        this.density = density;
    }

    @Override  Maddy Miller
    public RegionMaskingFilter createFromContext(EditContext input) {
        return new RegionMaskingFilter(
            new NoiseFilter(new RandomNoise(), this.density),
            new FeatureGenerator((EditSession) input.getDestination(), this.type)
        );
    }

    @Override  Maddy Miller
    public String toString() { return "feature of type " + type; }
}
```



Explanation:

This implementation demonstrates the **Factory pattern** by providing a way to create objects without specifying the exact class of object that will be created. The FeatureGeneratorFactory class encapsulates the creation of RegionMaskingFilter objects, which are configured with a NoiseFilter and a FeatureGenerator.
The createFromContext method takes an EditContext as input and uses it to create a new RegionMaskingFilter instance. This method abstracts the instantiation logic, allowing the client code to create RegionMaskingFilter objects without knowing the details of their construction.
By using the Factory pattern, the FeatureGeneratorFactory promotes loose coupling and enhances flexibility. It allows the creation logic to be centralized

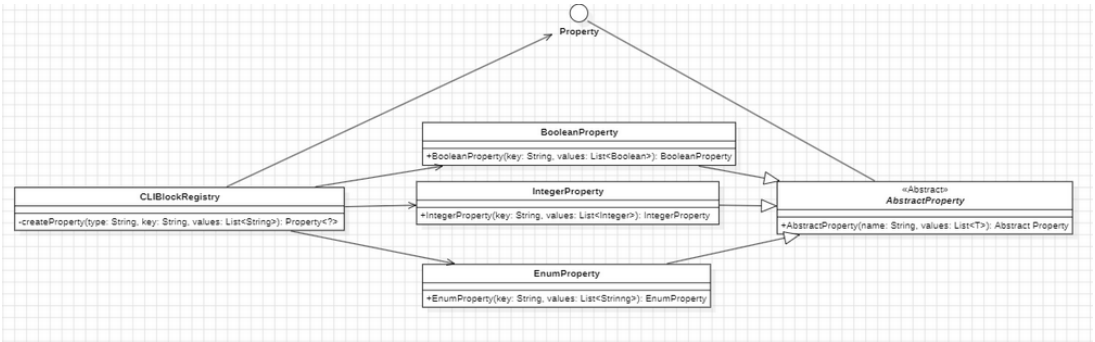
in one place, making it easier to manage and extend. This design pattern is particularly useful when the creation process is complex or when the specific type of object to be created is determined at runtime.

Factory Method [identification: 65969; review: 60593]

Lines 40-50 of the file worldedit-cli/src/main/java/com/sk89q/worldedit/cli/CLIBlockRegistry.java

---

```
private Property<?> createProperty(String type, String key, List<String> values) {  1 usage  Maddy Miller
    return switch (type) {
        case "int" -> new IntegerProperty(key, values.stream().map(Integer::parseInt).toList());
        case "bool" -> new BooleanProperty(key, values.stream().map(Boolean::parseBoolean).toList());
        case "enum" -> new EnumProperty(key, values);
        case "direction" ->
            new DirectionalProperty(key, values.stream().
                    map(String::toUpperCase).map(Direction::valueOf).toList());
        default -> throw new RuntimeException("Failed to create property");
    };
}
```



Explanation:

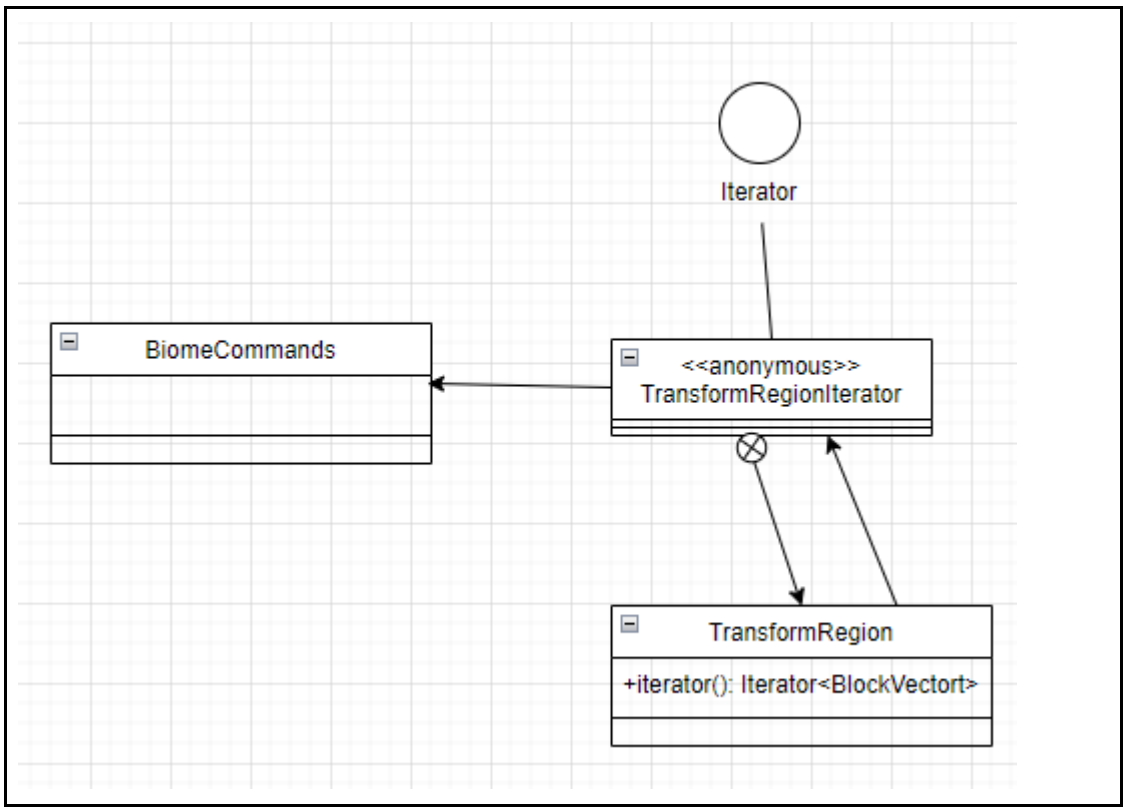
The Factory Method pattern is used here to create a Property object based on the argument type.
The correct product implementations are selected in the switch statement.

Iterator [identification: 65969; review: 62111]

Line 165-190 of the file worldedit-core/src/main/java/com/sk89q/worldedit/regions/TransformRegion.java

---

```
public Iterator<BlockVector3> iterator() {
    final Iterator<BlockVector3> it = region.iterator();

    return new Iterator<BlockVector3>() {  sk89q +1
        @Override  sk89q
        public boolean hasNext() { return it.hasNext(); }

        @Override  sk89q +1
        public BlockVector3 next() {
            BlockVector3 next = it.next();
            if (next != null) {
                return transform.apply(next.toVector3()).toBlockPoint();
            } else {
                return null;
            }
        }

        @Override  sk89q
        public void remove() { it.remove(); }
    };
}
```



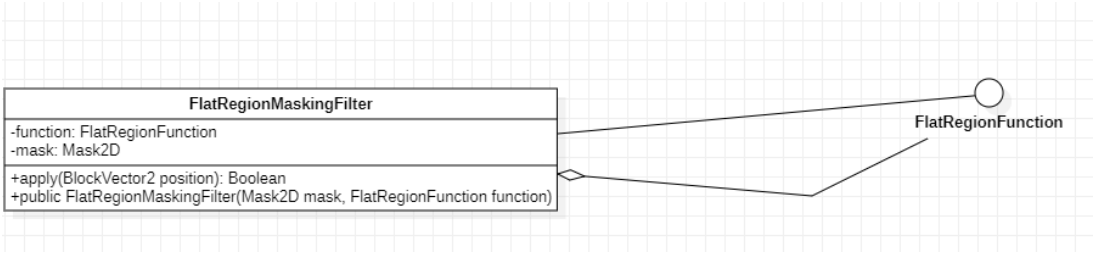
Explanation:

The Iterator pattern is used here to iterate over the elements in the region while transforming them.

Adapter [identification: 65969; review: 66081]

Line 33-57 of the file worldedit-core/src/main/java/com/sk89q/worldedit/function/FlatRegionMaskingFilter.java
---

```java
public class FlatRegionMaskingFilter implements FlatRegionFunction {  no usages

    private final FlatRegionFunction function;  2 usages
    private final Mask2D mask;  2 usages

    /**
     * Create a new masking filter.
     *
     * @param mask the mask
     * @param function the delegate function to call
     */
    public FlatRegionMaskingFilter(Mask2D mask, FlatRegionFunction function) {
        checkNotNull(function);
        checkNotNull(mask);

        this.mask = mask;
        this.function = function;
    }

    @Override  Kenzie Togami +1
    public boolean apply(BlockVector2 position) throws WorldEditException {
        return mask.test(position) && function.apply(position);
    }
}
```



Explanation:

This class is a decorator because implements the same interface as the attribute `function` and adds the functionality of the `mask` to the `function`.

Facade [identification: 66081; review: 62909]

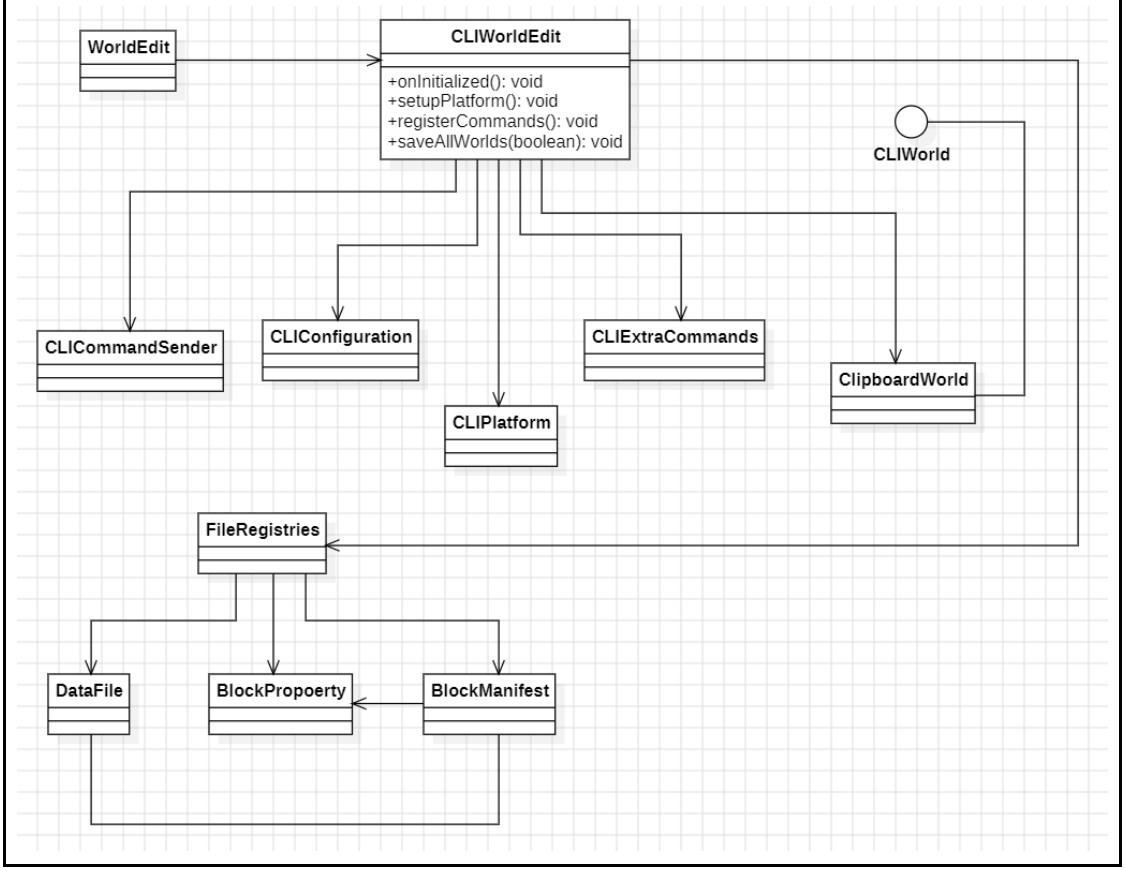
Line 92-116, 177-191 and 253-257 of the file worldedit-cli/src/main/java/com/sk89q/worldedit/cli/CLIWorldEdit.java

---

```
private void setupPlatform() {  1 usage  Matthew Miller +1
    WorldEdit.getInstance().getPlatformManager().register(platform);

    registerCommands();

    config = new CLIConfiguration( app: this);

    // There's no other platforms, so fire this immediately
    WorldEdit.getInstance().getEventBus().post(new PlatformsRegisteredEvent());

    this.fileRegistries = new FileRegistries( app: this);
    this.fileRegistries.loadDataFiles();
}

private void registerCommands() {  1 usage  Octavia Togami
    PlatformCommandManager pcm = WorldEdit.getInstance().getPlatformManager()
        .getPlatformCommandManager();
    pcm.registerSubCommands(
        name: "cli",
        ImmutableList.of(),
        desc: "CLI-specific commands",
        CLIExtraCommandsRegistration.builder(),
        new CLIExtraCommands()
    );
}
```

```
public void onInitialized() {  1 usage  Matthew Miller
    // Setup working directory
    workingDir = Paths.get( first: "worldedit");
    if (!Files.exists(workingDir)) {
        try {
            Files.createDirectory(workingDir);
        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }
    }

    this.commandSender = new CLICommandSender( app: this, LOGGER);
    this.platform = new CLIPlatform( app: this);
    LOGGER.info("WorldEdit CLI (version " + getInternalVersion() + ") is loaded");
}
```

```
public void saveAllWorlds(boolean force) {  2 usages  Matthew Mill
    platform.getWorlds().stream()
            .filter(world -> world instanceof CLIWorld)
            .forEach(world -> ((CLIWorld) world).save(force));
}
```



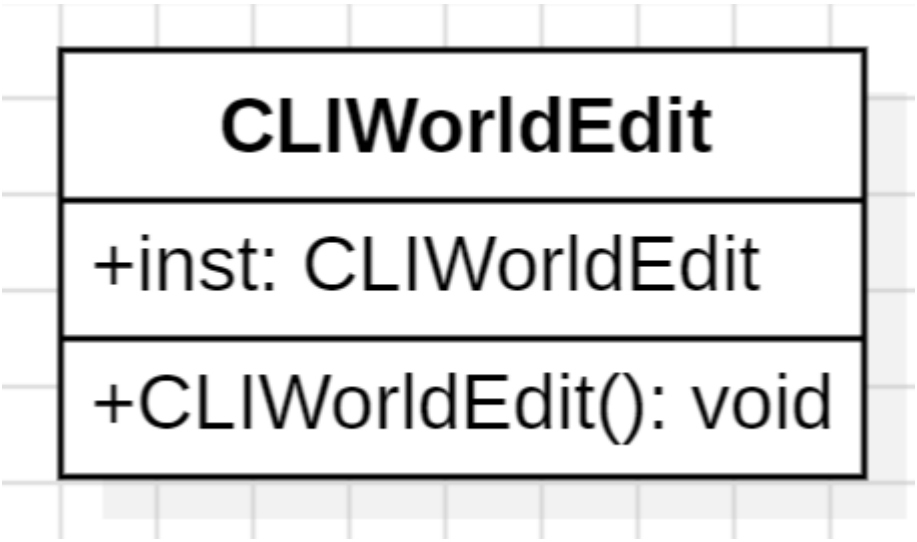
Explanation:

The facade pattern is used here to simplify interactions with the WorldEdit CLI subsystem and hide his complexity. it simplifies operations like initialization and setup, command registration, event handling and subsystem management.

Singleton [identification: 66081; review: 60593]

Line 77 and 88-90 of the file
worldedit-cli/src/main/java/com/sk89q/worldedit/cli/CLIWorldEdit.java
---
```
public static CLIWorldEdit inst;
```

```
public CLIWorldEdit() {
    inst = this;
}
```



Explanation:

The singleton pattern is used in the CLIWorldEdit in a non-optimal way, it has a public constructor with a unique instance field as a public static global variable. This ensures that only one instance of the class is created as it does not allow multiple instances to be created, however raises some problems. We cannot control who accesses the global variable, it is initialized automatically on class load and for not being private and final it can be modified for any other class. A better way would be to use a private constructor, a private static final global variable for the instance field and create a method to access it, so we can control when the instance is created and who can access it.

Proxy [identification: 66081; review: 65330]

Line 44-60 of the file worldedit-core/src/main/java/com/sk89q/worldedit/extension/platform/PlayerProxy.java

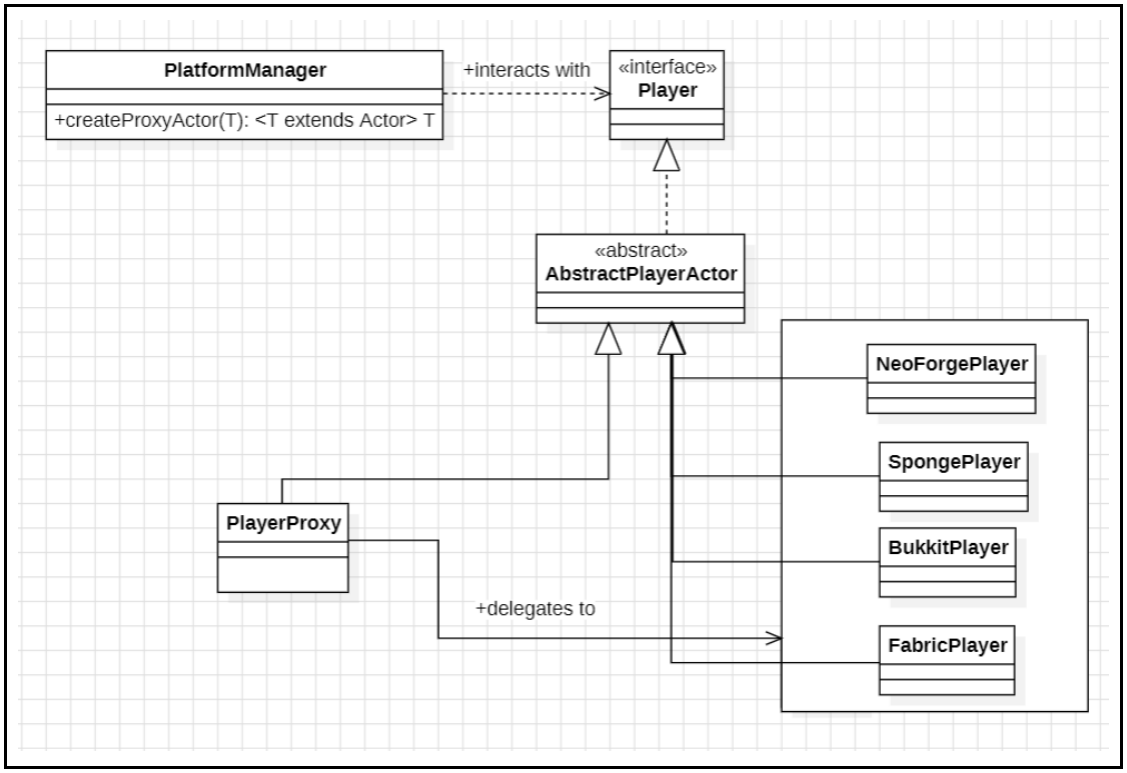
---

```
class PlayerProxy extends AbstractPlayerActor {  1 usage   sk89q +5

    private final Player basePlayer;  23 usages
    private final Actor permActor;  3 usages
    private final Actor cuiActor;  2 usages
    private final World world;  2 usages

    PlayerProxy(Player basePlayer, Actor permActor, Actor cuiActor, World world) {
        checkNotNull(basePlayer);
        checkNotNull(permActor);
        checkNotNull(cuiActor);
        checkNotNull(world);
        this.basePlayer = basePlayer;
        this.permActor = permActor;
        this.cuiActor = cuiActor;
        this.world = world;
    }
```

Line 258-276 of the file worldedit-core/src/main/java/com/sk89q/worldedit/extension/platform/PlatformManager.java

---

```
public <T extends Actor> T createProxyActor(T base) {
    checkNotNull(base);

    if (base instanceof Player player) {
        Player permActor = queryCapability(Capability.PERMISSIONS).matchPlayer(player);
        if (permActor == null) {
            permActor = player;
        }

        Player cuiActor = queryCapability(Capability.WORLDEDIT_CUI).matchPlayer(player);
        if (cuiActor == null) {
            cuiActor = player;
        }

        return (T) new PlayerProxy(player, permActor, cuiActor, getWorldForEditing(player.getWorld()
    } else {
        return base;
    }
}
```



Explanation:

The Proxy Pattern is demonstrated here by the PlayerProxy class that represents a real Actor and is used to control access to it. The PlatformManager class creates a proxy for the Player class, which is used to interact with the Player object. The proxy class PlayerProxy is used to check permissions and capabilities of the Player object before allowing access to it. This pattern is useful to add additional functionality to the real object, like checking permissions, logging, caching, etc. without modifying the real object.

Template [identification: 60593; review: 65969]

Line 47-100 of the file worldedit-core/src/main/java/com/sk89q/worldedit/regions/AbstractRegion.java

---

```
@Override  4 overrides  Kenzie Togami
public Vector3 getCenter() {
    return getMinimumPoint().add(getMaximumPoint()).toVector3().divide( n: 2);
}

/** Get the iterator. ...*/
@Override  6 overrides  Kenzie Togami +1
public Iterator<BlockVector3> iterator() { return new RegionIterator(this); }

@Override  TomyLobo +1
public World getWorld() { return world; }

@Override  TomyLobo +1
public void setWorld(World world) { this.world = world; }

@Override  6 overrides  TomyLobo +1
public void shift(BlockVector3 change) throws RegionOperationException {...}

@Override  5 overrides  aumgn
public AbstractRegion clone() {...}

@Override  4 usages  3 overrides  Kenzie Togami +2
public List<BlockVector2> polygonize(int maxPoints) {...}
```

Line 284-365 of the file worldedit-core/src/main/java/com/sk89q/worldedit/regions/CuboidRegion.java

---

```
@Override  TomyLobo +1
public void shift(BlockVector3 change) throws RegionOperationException {...}

@Override  Kenzie Togami +2
public Set<BlockVector2> getChunks() {...}

@Override  2 usages  Kenzie Togami +2
public Set<BlockVector3> getChunkCubes() {...}

@Override  Kenzie Togami
public boolean contains(BlockVector3 position) {...}

@Override  sk89q +3
public Iterator<BlockVector3> iterator() {...}
```

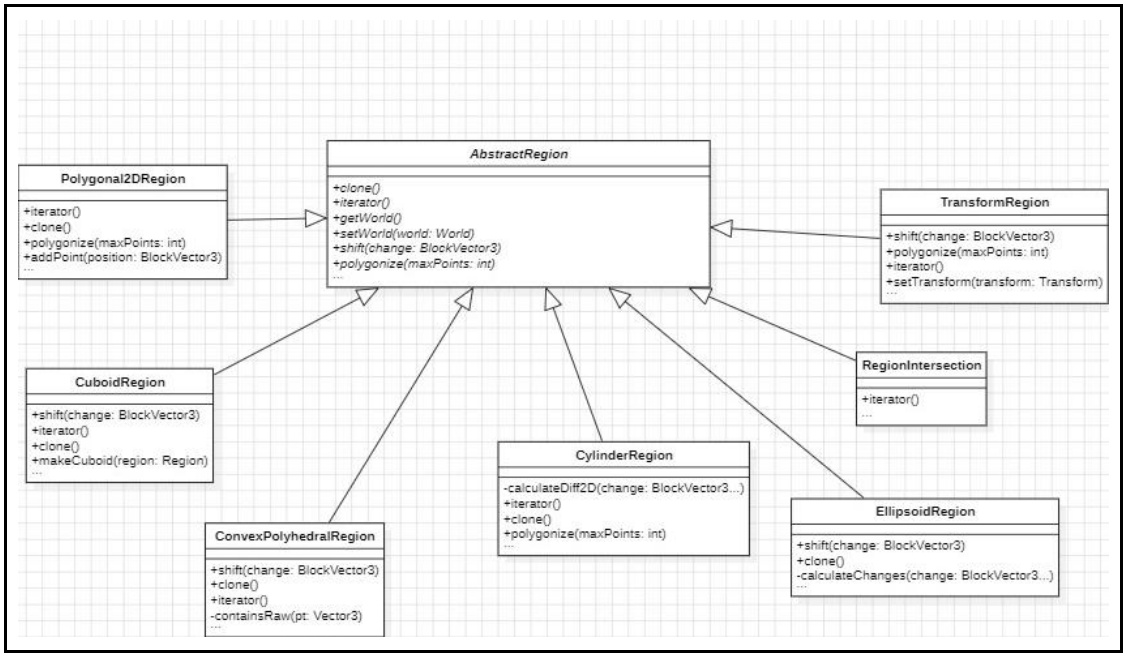
Line 248-267 and 328-330 of the file

worldedit-core/src/main/java/com/sk89q/worldedit/regions/ConvexPolyhedralRegion.java

---

```java
@Override  TomyLobo +1
public void shift(BlockVector3 change) throws RegionOperationException {
    Vector3 vec = change.toVector3();
    shiftCollection(vertices, change);
    shiftCollection(vertexBacklog, change);

    for (int i = 0; i < triangles.size(); ++i) {
        final Triangle triangle = triangles.get(i);

        final Vector3 v0 = vec.add(triangle.getVertex( index: 0));
        final Vector3 v1 = vec.add(triangle.getVertex( index: 1));
        final Vector3 v2 = vec.add(triangle.getVertex( index: 2));

        triangles.set(i, new Triangle(v0, v1, v2));
    }

    minimumPoint = change.add(minimumPoint);
    maximumPoint = change.add(maximumPoint);
    centerAccum = change.multiply(vertices.size()).add(centerAccum);
    lastTriangle = null;
}
```

```java
@Override  TomyLobo
public AbstractRegion clone() {
    return new ConvexPolyhedralRegion(this);
}
```



Explanation:

The AbstractRegion class provides a base implementation for Region objects, establishing common behaviors and methods like getCenter, iterator, getVolume, and getChunks. The other classes extend AbstractRegion, implementing these specific methods.

Adapter [identification: 60593; review: 66081]

Line 208-225 of the file worldedit-fabric/src/main/java/com/sk89q/worldedit/fabric/FabricWorld.java

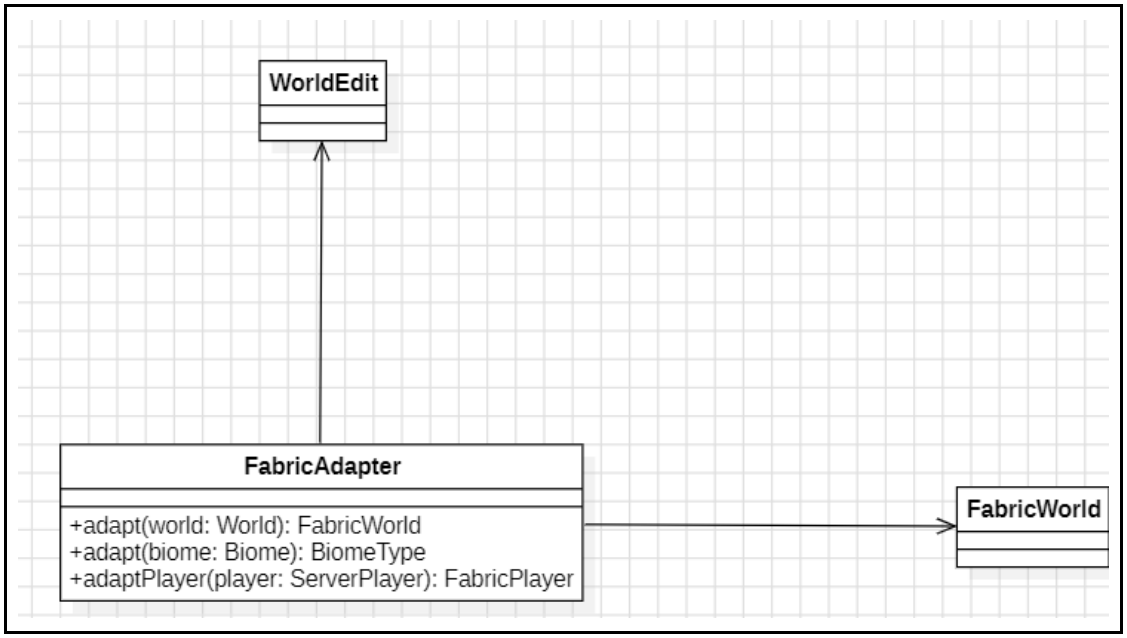
---

```
@Override  2 usages  Matthew Miller +1
public int getBlockLightLevel(BlockVector3 position) {
    checkNotNull(position);
    return getWorld().getMaxLocalRawBrightness(FabricAdapter.toBlockPos(position));
}

@Override  Matthew Miller +1
public boolean clearContainerBlockContents(BlockVector3 position) {
    checkNotNull(position);

    BlockEntity tile = getWorld().getBlockEntity(FabricAdapter.toBlockPos(position));
    if ((tile instanceof Clearable)) {
        ((Clearable) tile).clearContent();
        return true;
    }
    return false;
}
```

Line74-103 of the file worldedit-fabric/src/main/java/com/sk89q/worldedit/fabric/FabricAdapter.java

---

```
public static World adapt(net.minecraft.world.level.Level world) { return new FabricWorld(world); }

/** Create a Fabric world from a WorldEdit world. ...*/
public static net.minecraft.world.level.Level adapt(World world) {  Octavia Togami +1
    checkNotNull(world);
    if (world instanceof FabricWorld) {
        return ((FabricWorld) world).getWorld();
    } else {
        // TODO introduce a better cross-platform world API to match more easily
        throw new UnsupportedOperationException("Cannot adapt from a " + world.getClass());
    }
}

public static Biome adapt(BiomeType biomeType) {  Octavia Togami +1
    return FabricWorldEdit.getRegistry(Registries.BIOME)
        .get(ResourceLocation.parse(biomeType.id()));
}

public static BiomeType adapt(Biome biome) {...}
```



Explanation:

An adapter to Minecraft worlds for WorldEdit. The FabricAdapter class provides methods to adapt various objects from one API to another, such as World, Biome, Vector3, BlockState, and more. These methods convert objects between WorldEdit representations (e.g., World, BlockState, Vector3) and Minecraft's Fabric API representations

Command [identification: 60593; review: 62909]

Line 28-39 of the file worldedit-core/src/main/java/com/sk89q/worldedit/command/tool/Tool.java----------

```
public interface Tool {  19 implementations  sk89q <the.sk89q> +2

    /**
     * Checks to see if the player can still be using this tool (considering
     * permissions and such).
     *
     * @param actor the actor
     * @return true if use is permitted
     */
    boolean canUse(Actor actor);  14 implementations  Matthew Miller

}
```

Line 33-68 of the file worldedit-core/src/main/java/com/sk89q/worldedit/command/tool/BlockTool.java

---

```
public interface BlockTool extends Tool {  12 implementations  Matthew Miller +4 *

    /** Perform the primary action of this tool. ...*/
    @Deprecated  Octavia Togami *
    default boolean actPrimary(Platform server, LocalConfiguration config, Player player,
                               LocalSession session, Location clicked) {
        return actPrimary(server, config, player, session, clicked,  face: null);
    }

    /** Perform the primary action of this tool. ...*/
    @NonAbstractForCompatibility(  11 overrides  Matthew Miller +1 *
        delegateName = "actPrimary",
        delegateParams = { Platform.class, LocalConfiguration.class, Player.class
                , LocalSession.class, Location.class }
    )
    default boolean actPrimary(Platform server, LocalConfiguration config, Player player,
                               LocalSession session, Location clicked, @Nullable Direction face) {
        DeprecationUtil.checkDelegatingOverride(getClass());
        return actPrimary(server, config, player, session, clicked);
    }
}
```

Line 36-73 of the file worldedit-core/src/main/java/com/sk89q/worldedit/command/tool/DoubleActionBlockTool.java

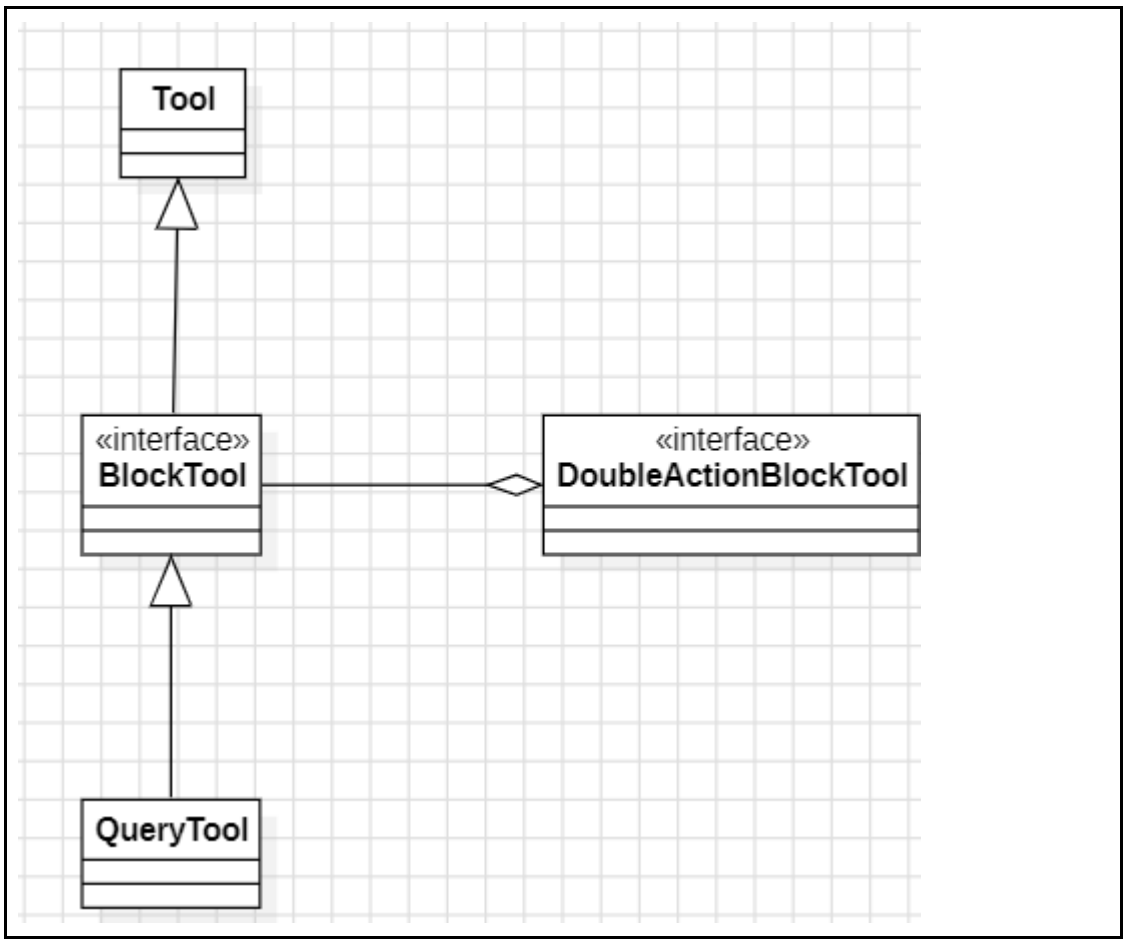
---

```java
public interface DoubleActionBlockTool extends BlockTool {  6 usages  3 implementations  Matthew Miller +4 *

    /** Perform the secondary action of this block tool. ...*/
    @Deprecated  Octavia Togami *
    default boolean actSecondary(Platform server, LocalConfiguration config, Player player,
                                 LocalSession session, Location clicked) {
        return actSecondary(server, config, player, session, clicked,  face: null);
    }

    /** Perform the secondary action of this block tool. ...*/
    @NonAbstractForCompatibility(  3 usages  3 overrides  Matthew Miller +1 *
        delegateName = "actSecondary",
        delegateParams = { Platform.class, LocalConfiguration.class, Player.class,
                LocalSession.class, Location.class }
    )
    default boolean actSecondary(Platform server, LocalConfiguration config, Player player,
                                 LocalSession session, Location clicked, @Nullable Direction face) {
        DeprecationUtil.checkDelegatingOverride(getClass());
        return actSecondary(server, config, player, session, clicked);
    }

}
```

Line 44-83 of file worldedit-core/src/main/java/com/sk89q/worldedit/command/tool/QueryTool.java

---

```java
public class QueryTool implements BlockTool {  2 usages  wizjany +5 *

    @Override  sk89q <the.sk89q> +1
    public boolean canUse(Actor player) { return player.hasPermission("worldedit.tool.info"); }

    @Override  wizjany +4 *
    public boolean actPrimary(Platform server, LocalConfiguration config, Player player,
                              LocalSession session, Location clicked, @Nullable Direction face) {...}

}
```


Tool
«interface»
BlockTool
«interface»
DoubleActionBlockTool
QueryTool

Explanation:

Each Tool action method ('actPrimary' and 'actSecondary') encapsulates a command that can be executed independently. The Tool hierarchy is designed to perform specific commands/actions (such as acting on a block). Each 'actPrimary' and 'actSecondary' call represents a command being executed on a target location.
Invoker: ToolCommand class in package com.sk89q.worldedit.command.
Receiver: Platform, Player, and World, which handle the requested operations and provide contextual information.

Codebase Metrics Assessment

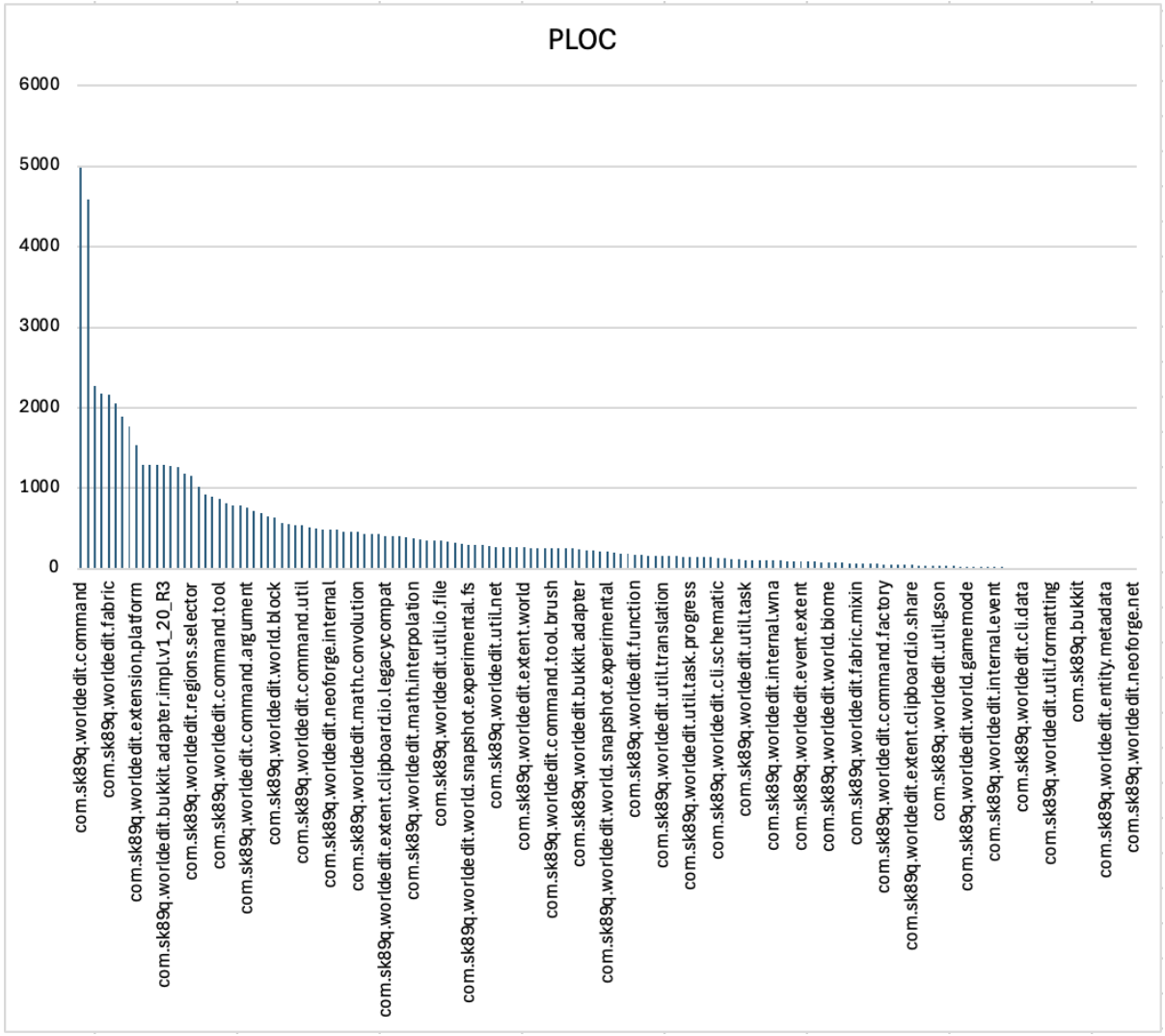
[Identification: 60593; review: 62111]

Lines of Code (LOC)

Definition: LOC measures the total number of code lines in a project or package. Blank lines and comments are included, reflecting the full volume of the source code.

Importance: This metric provides an overview of the project's size, helping estimate the required effort and future maintenance needs.

Potential Issues: Projects with high LOC may indicate a complex system that could be challenging to maintain. High LOC may signal large classes or methods, which makes understanding and testing harder.



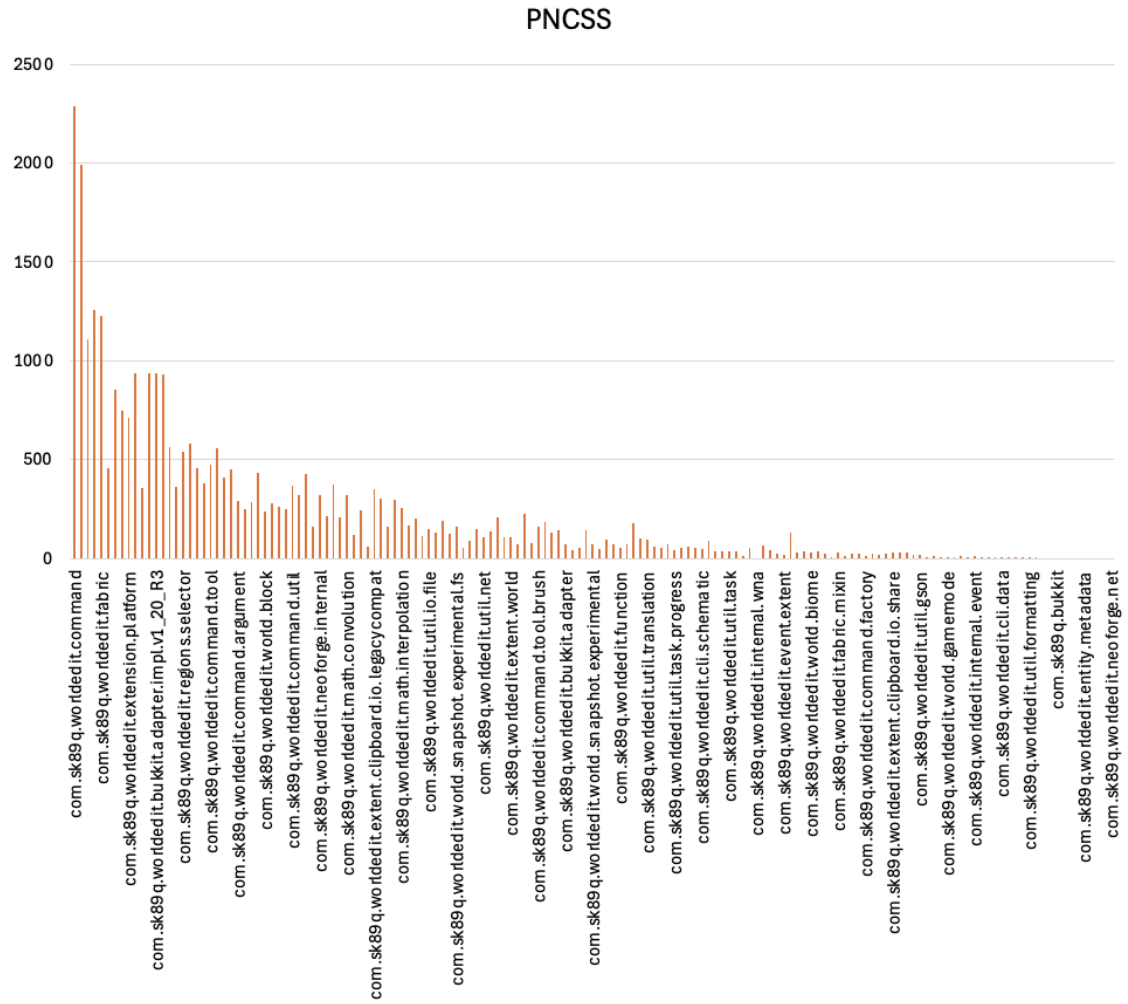
Project Level
Calculated Metric: 73,383 LOC
Package Level (com.sk89q.worldedit)
Note that these values do not include the contents of sub-packages within this package.
Calculated Metric: 4,592 LOC
Class Level (EditSession)
Calculated Metric: 2,440 LOC

Non-Commenting Source Statements (NCSS) (Project Level)

Definition: NCSS counts only executable lines of code, excluding comments and blank statements. This metric closely tracks the points where executable code appears, offering a more precise view of logical complexity.

Importance: NCSS can indicate the effort required to understand critical parts of the code and helps identify where potential simplification can reduce complexity.

Potential Issues: High NCSS in specific classes may indicate long methods or overloaded classes. The metric may also indicate code smells, such as methodsor classes with multiple responsibilities.



Project Level

Calculated Metric: 41,753 NCSS

Package Level (com.sk89q.worldedit)

Note that these values do not include the contents of sub-packages within this package.

Calculated Metric: 1,993 NCSS

Class Level (EditSession)

Calculated Metric: 1,266 NCSS

Number of Concrete Classes (NOCC) (Project Level)

Definition: Counts all concrete (non-abstract) classes in the project or package. This metric helps measure the system's extensibility and modularity.

Importance: A high number of concrete classes can signal a project with low code reuse, highlighting the need for improvements in abstraction and reuse of common functionalities.

Potential Issues: A high number of concrete classes can complicate the project structure, making it harder to understand and increasing the likelihood of duplicated logic. It indicates a potential increase in the need for unit and integration tests.
Project Level
Calculated Metric: 1,513 Concrete Classes
Package Level (com.sk89q.worldedit)
Note that these values do not include the contents of sub-packages within this package.
Calculated Metric: 18 Concrete Classes

Relationship with Code Smells (Project Level)

These metrics may reflect the presence of classic code smells, such as:

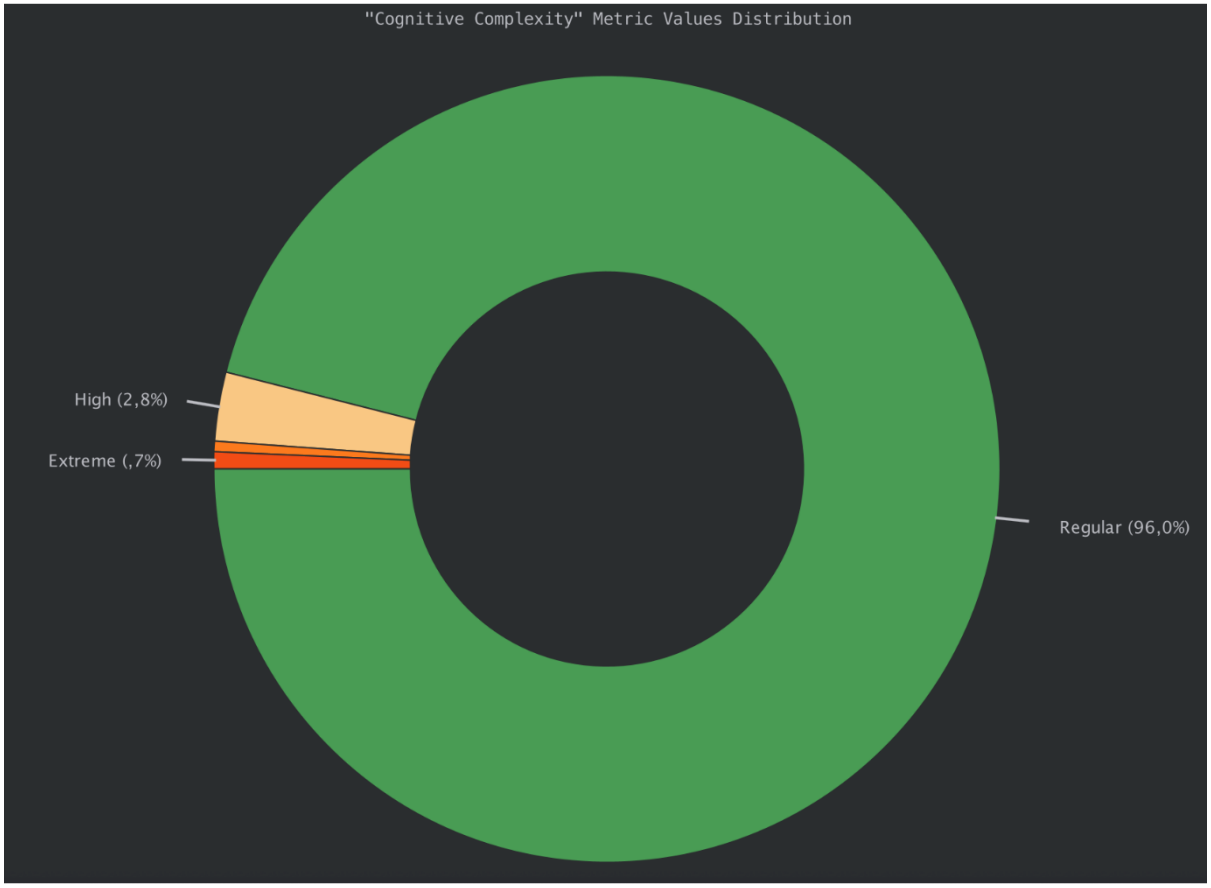
- **God Class/Method:** Classes and methods with many LOC or NCSS are typical indicators of responsibility overload.

- **Long Method:** Extensive methods indicate low modularity and are difficult to test individually. **Large Class:** Many concrete classes or classes with high LOC suggest a lack of abstraction and poor separation of responsibilities.

- **Duplicated Code:** Identifying and refactoring duplicate blocks of code reduce the LOC and NCSS.

- **Lazy Class:** Classes with low LOC and NCSS may be lazy classes.

Cognitive Complexity



Definition: Measures how understandable a piece of code is by evaluating control flow structures such as loops, conditionals, and nesting levels. It focuses on the mental effort required to understand the code, rather than the number of possible paths.

Importance: High Cognitive Complexity increases the difficulty of reading, understanding, and maintaining code, leading to a higher likelihood of bugs and issues during development. Lower complexity helps keep code readable, making it easier to debug, refactor, and test.

Potential Issues: Code with high Cognitive Complexity can be challenging to maintain, as developers may struggle to understand the logic. High complexity increases the risk of errors and the time required to make updates or fix issues.

Code Smells:

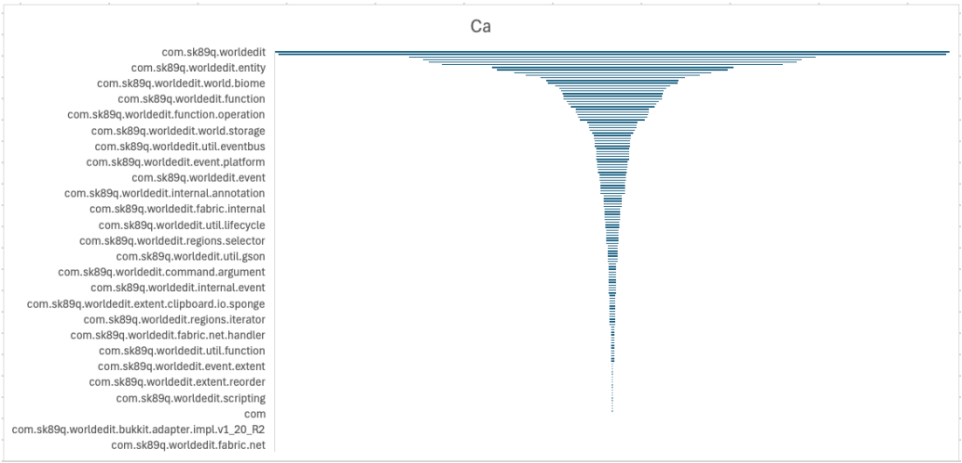
- Long Method: Methods that are overly long tend to be more complex, often indicating that the function is handling too much logic and should be broken down.
- Deeply Nested Conditionals: Extensive nesting (e.g., if statements within loops or other conditionals) increases Cognitive Complexity and makes the code harder to follow.
- Duplicated Code: Duplicate blocks of code lead to redundancy, making maintenance more difficult, especially when complex logic is repeated across different areas.

Metric values:
Class EditSession (com.sk89q.worldedit.world) - Cognitive Complexity: 381
Class SelectionCommands (com.sk89q.worldedit.command) - Cognitive Complexity: 202

Dependency Metrics

Afferent and Efferent Coupling

Definition:
Afferent Coupling (Ca): The number of other modules that depend on a specific module, indicating how central or critical the module is to the rest of the system.



Efferent Coupling (Ce): The number of modules that a specific module depends on, indicating how much the module relies on external components or other modules.

Importance: Managing dependencies is crucial for maintainability and stability. High afferent coupling can create bottlenecks where a change in one module

affects many others. High efferent coupling suggests that a module may be too reliant on external components, increasing the risk of breaking changes.

Potential Issues: High afferent coupling (Ca) means the module is critical, making changes risky because many other modules depend on it. High efferent coupling (Ce) indicates that the module is fragile, as it depends heavily on other parts of the codebase.

Code Smells:
- God Class: Modules with very high afferent coupling often contain too much functionality, leading to a God Class that centralizes too many responsibilities. **-**
- Cyclic Dependencies: Cycles occur when modules depend on each other, either directly or indirectly, creating a tightly coupled structure that makes refactoring difficult and can lead to unexpected bugs.
- Feature Envy: Modules with high efferent coupling often have logic that would be better suited elsewhere, leading to excessive reliance on other modules' data and functions.

Metric Values:
Package worldedit - Afferent Coupling: 320 (highly depended upon by many modules)
Package worldedit - Efferent Coupling: 107 (relies heavily on multiple external services)
This combination can indicate a God Class or a Cyclic Dependency issue. Such a module both depends on and is depended upon by many other modules, making it hard to maintain.

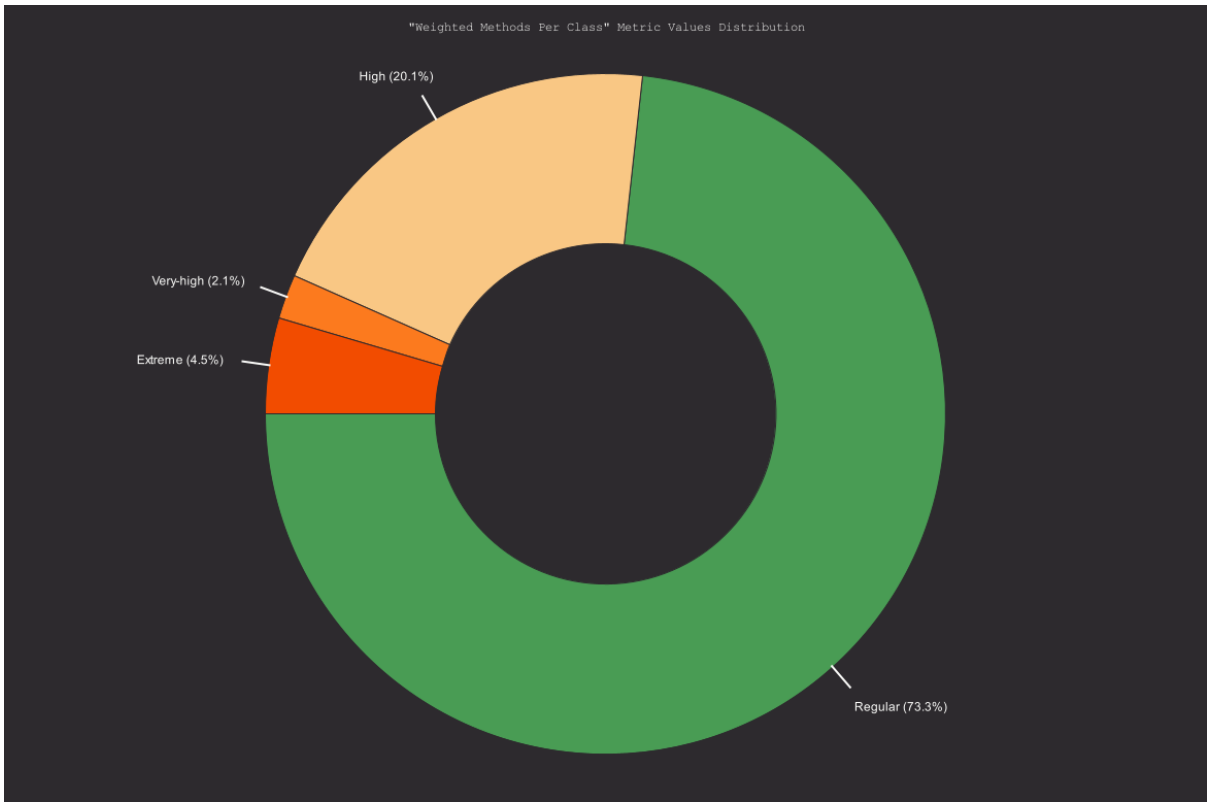
[identification: 62111; review: 66081]

Chidamber-Kemerer metrics set (Class level metrics)

Weighted methods per class:

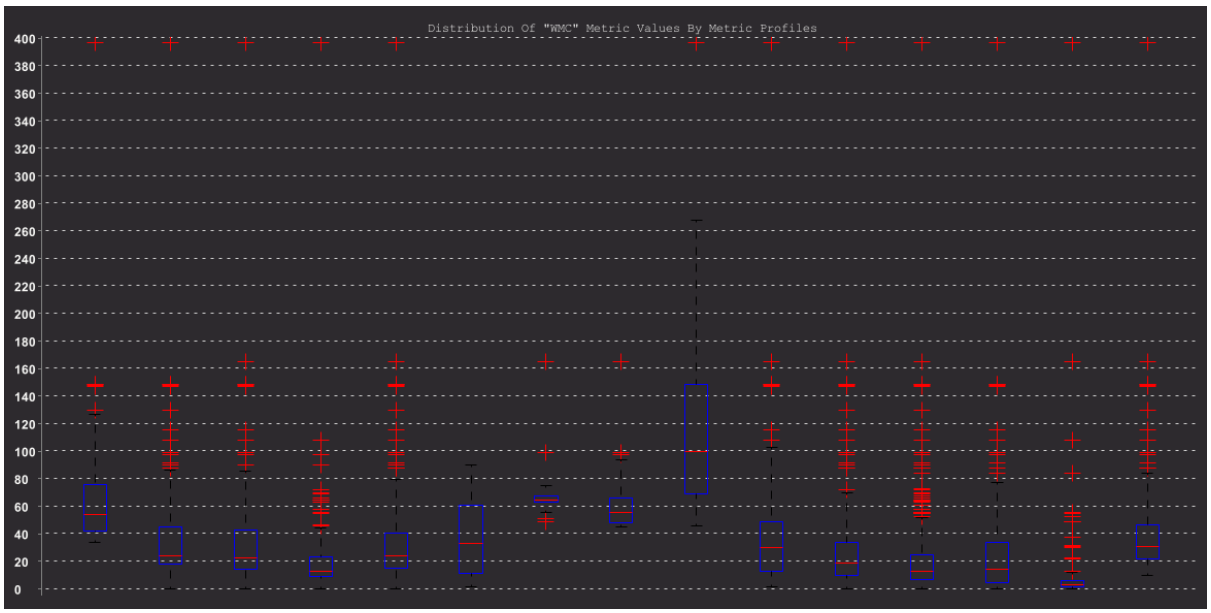
A weighted sum of methods implemented within a class. It is parameterized by a way to compute the weight of each method.
It provides insight into how complex or "heavy" a class is, which can influence its readability, maintainability, and potential code quality issues.

Charts:



○ High ○ Very-high ○ Extreme

Class	Range	Value
EditSession	[45..∞)	397
LocalSession	[45..∞)	165
PaperweightAdapter	[45..∞)	149
PaperweightAdapter	[45..∞)	149
PaperweightAdapter	[45..∞)	148
PaperweightAdapter	[45..∞)	147
YAMLNode	[45..∞)	130
AbstractPlayerActor	[45..∞)	116
WorldEdit	[45..∞)	108
CompilingVisitor	[45..∞)	99
SelectionCommands	[45..∞)	98
FabricWorld	[45..∞)	92
DefaultBlockParser	[45..∞)	90
NeoForgeWorld	[45..∞)	88
BukkitWorld	[45..∞)	88
BlockVector3	[45..∞)	84
Vector3	[45..∞)	73
CommandContext	[45..∞)	72
CommandsManager	[45..∞)	70
GeneralCommands	[45..∞)	69
NeoForgeWorldEdit	[45..∞)	68
Polygonal2DRegion	[45..∞)	66
BlockTransformExtent	[45..∞)	65



Profile Name	Metric Profile
Brain Class	CC ≥ 3 ∧ LOC ≥ 30 ∧ MND ≥ 3 ∧ NOAV ≥ 3 ∧ TCC < 0.5 ∧ WMC ≥ 34
Brain Method	CC ≥ 3 ∧ LOC ≥ 30 ∧ MND ≥ 3 ∧ NOAV ≥ 3
Complex Method	CC ≥ 8
Data Class	NOAM ≥ 4 ∧ NOPA ≥ 3 ∧ WMC < 15 ∧ WOC < 0.34
Deeply Nested Conditions	CND ≥ 3
Dispersed Coupling	CDISP ≥ 0.66 ∧ CINT ≥ 8 ∧ MND ≥ 2
Feature Envy	ATFD ≥ 5 ∧ FDP ≥ 5 ∧ LAA < 0.33
God Class (type 1)	ATFD ≥ 6 ∧ TCC < 0.33 ∧ WMC ≥ 47
God Class (type 2)	ATFD < 4 ∧ NOA ≥ 30 ∧ WMC ≥ 44
God Class (type 3)	ATFD ≥ 4 ∧ CBO ≥ 11 ∧ WMC ≥ 44
God Class (type 4)	CBO ≥ 1 ∧ RFC ≥ 144 ∧ TCC < 0.37 ∧ WMC ≥ 46
High Coupling	CBO ≥ 20
Intensive Coupling	CDISP < 0.5 ∧ CINT ≥ 8 ∧ MND ≥ 2
Long Method	LOC ≥ 16
Long Parameters List	NOPM ≥ 4
Too Many Fields	NOA ≥ 15
Too Many Methods	NOM ≥ 10

Potential trouble spots:

A notable **4.5% of classes have "Extreme" WMC values**, indicating they are highly complex and potentially overloaded with responsibilities. These classes are at risk of becoming **God Classes** or having multiple responsibilities, which can lead to maintenance issues, difficulty in testing, and reduced code readability.
The most obvious case is the EditSession class, located in the com.sk89q.worldedit package, with a WMC of 397.
The boxplot chart outliers also suggest potential violations of the **Single Responsibility Principle**, as classes may be overloaded with responsibilities, contributing to their high WMC scores.

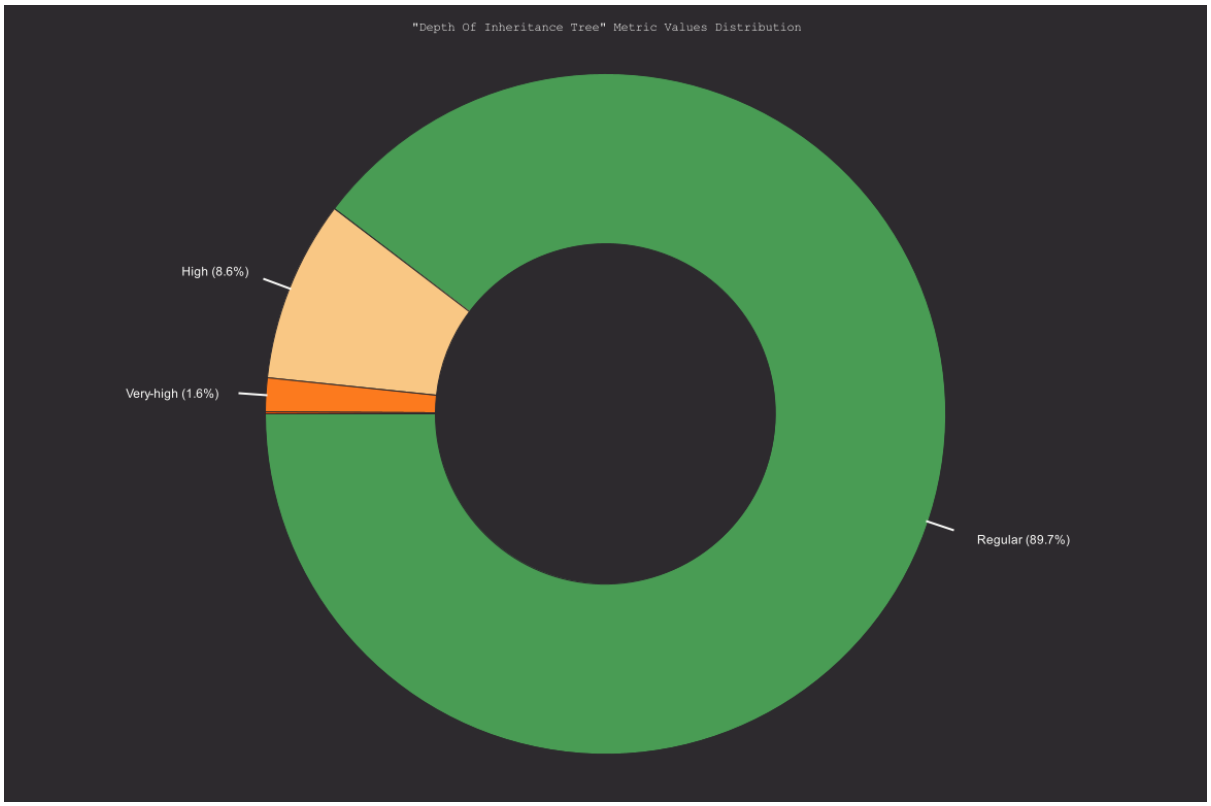
Relation with code smell:

A WMC of 397 indicates that the EditSession class has a very high number of methods, contributing to its complexity and lack of cohesion. This high WMC score is typical of a **God Class** because it suggests that EditSession is doing too much on its own, handling numerous responsibilities that should ideally be distributed across multiple, smaller classes.

Depth of Inheritance Tree:

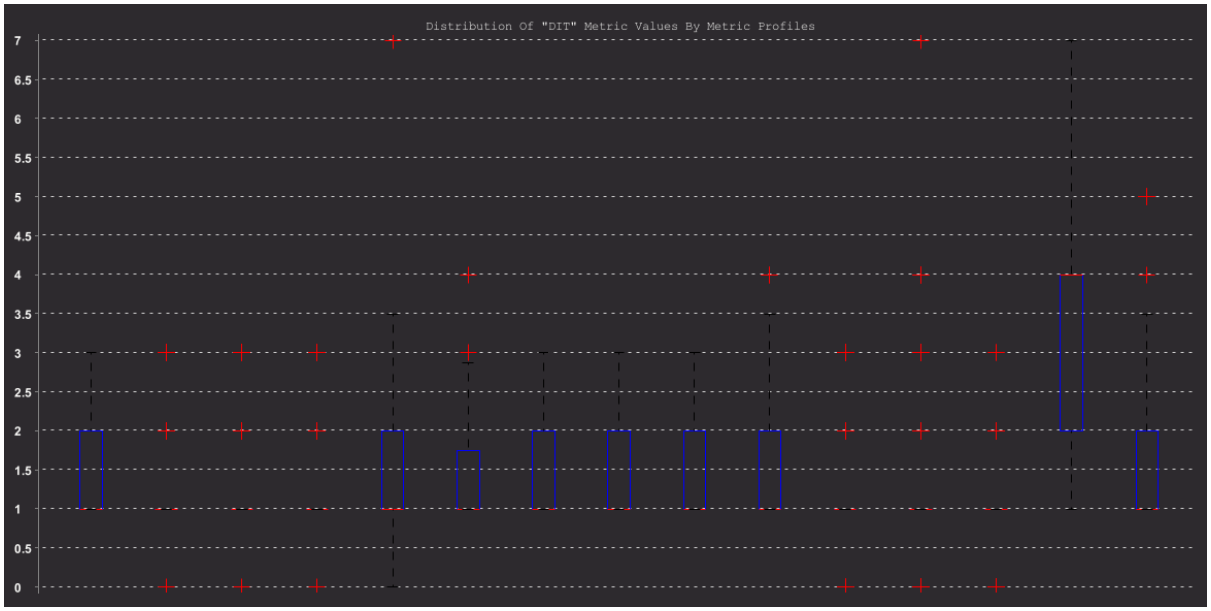
The maximum length of a path from a class to a root class in the inheritance structure of a system. DIT measures how many super-classes can affect a class.
This metric is used to assess the complexity of a class's inheritance structure and can provide insight into potential design issues or benefits.

Charts:



High | Very-high | Extreme

Class	Range	Value
NavigableEditorPane	[7..∞)	7



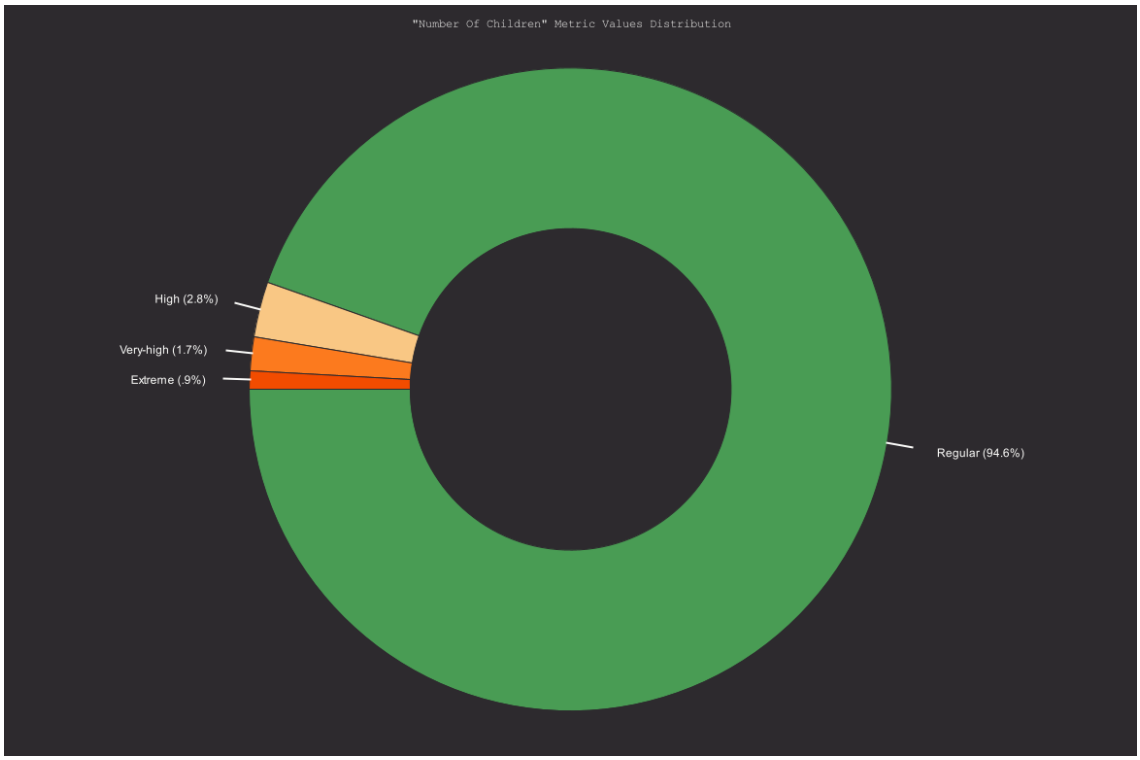
Potential trouble spots:

High DIT can indicate over-reliance on inheritance, which can lead to rigidity in the code and difficulty in maintaining or extending it.
The only extreme case is the NavigableEditorPane private class located inside the InfoEntryPoint in the com.sk89q.worldedit.internal.util package, with a DIT of 7.
In the boxplot chart, the **Long Method** profile stands out with a significant number of high outliers. This suggests that these classes are not only lengthy in terms of lines of code or complexity within individual methods, but they may also be embedded in deep inheritance chains.

Number of Children:

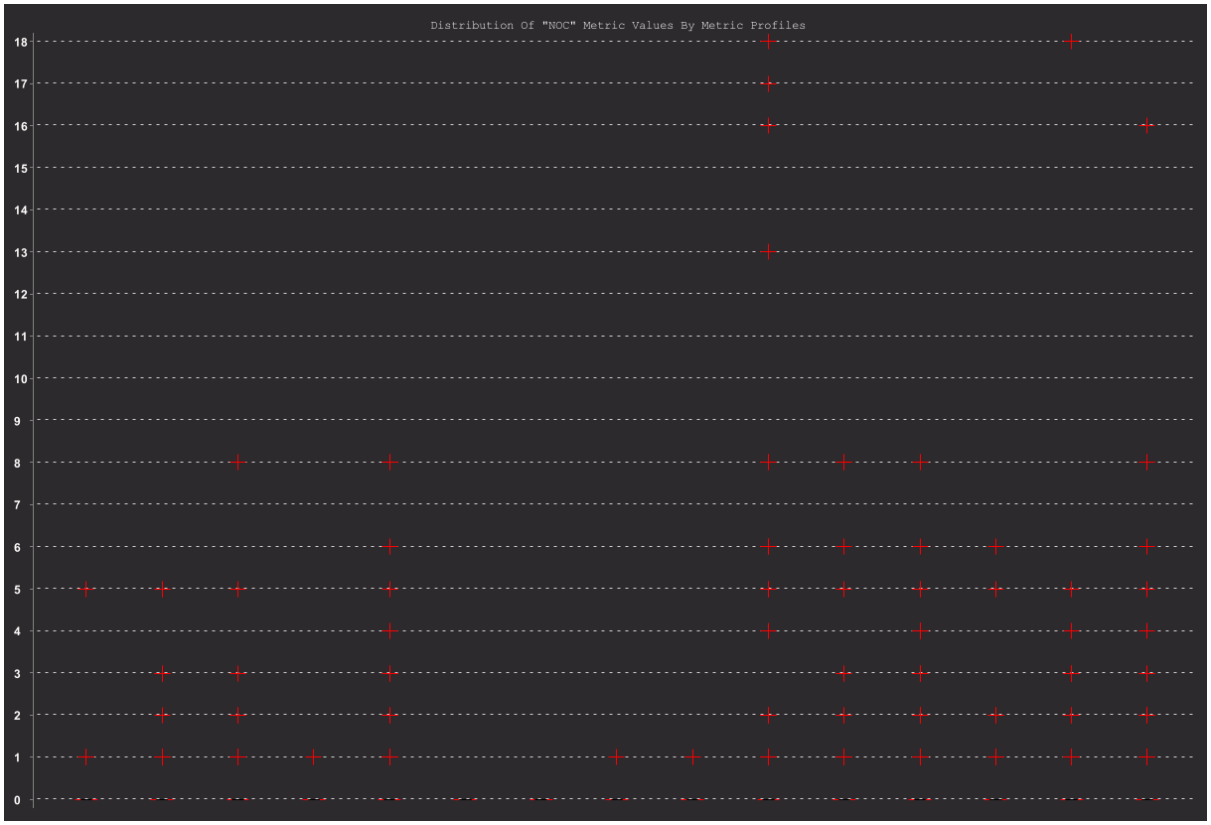
Calculates the total number of direct subclasses of given class. This metric measures how many sub-classes are going to inherit the methods of the parent class.
The size of NOC approximately indicates the level of reuse in an application. If NOC grows it means reuse increases.
On the other hand, as NOC increases, the amount of testing will also increase because more children in a class indicate more responsibility. So, NOC represents the effort required to test the class and reuse.

Charts:



O High O Very-high O Extreme

Class	Range	Value
WorldEditException	[7..∞)	18
InputParser	[7..∞)	17
AbstractDelegateExtent	[7..∞)	16
Tag	[7..∞)	13
AbstractMask	[7..∞)	12
Event	[7..∞)	9
AbstractRegion	[7..∞)	8
PaginationBox	[7..∞)	8
AbstractExtentMask	[7..∞)	8
AbstractMask2D	[7..∞)	7



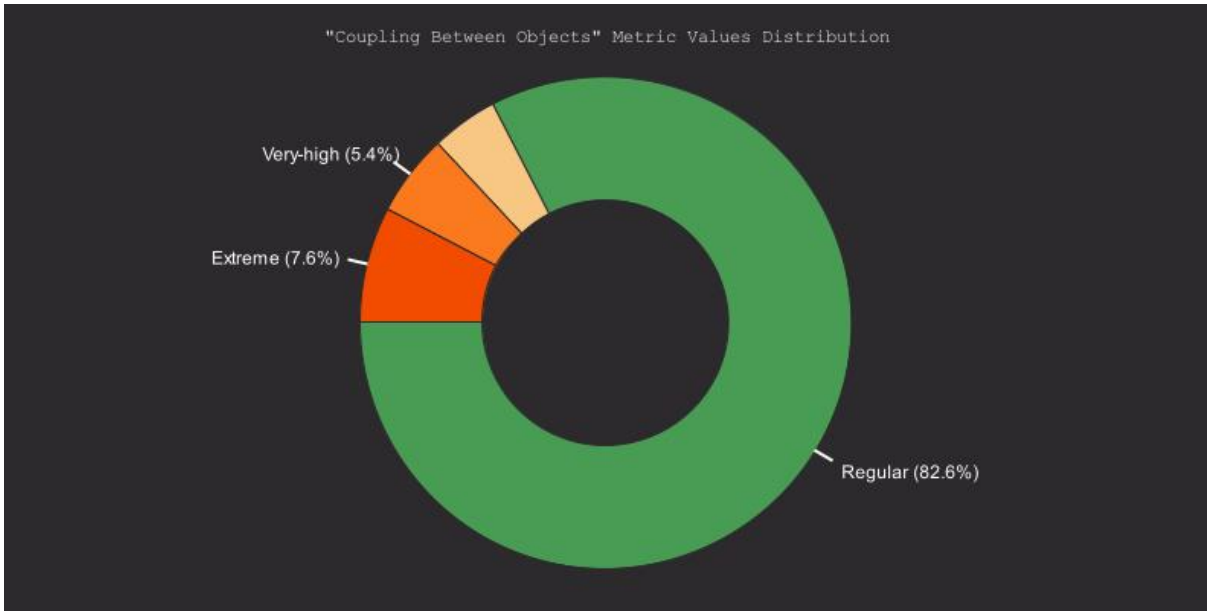
Potential trouble spots:

Extremely high NOC values may indicate over-reliance on a single base class, which could make maintenance more challenging. The tables shows some extreme values, being the worst one the WorldEditException abstract class located in the com.sk89q.worldedit package, with a NOC of 18.
In the boxplot chart, the **High Coupling** profile, having the most highest outliers, aligns with the idea that high coupling can result in complex interdependencies, potentially leading to a sprawling inheritance structure where classes depend on numerous other classes. This can make the system harder to maintain and increase the risk of rigidity, as changes in one class may propagate across many child classes.

Coupling between object classes:

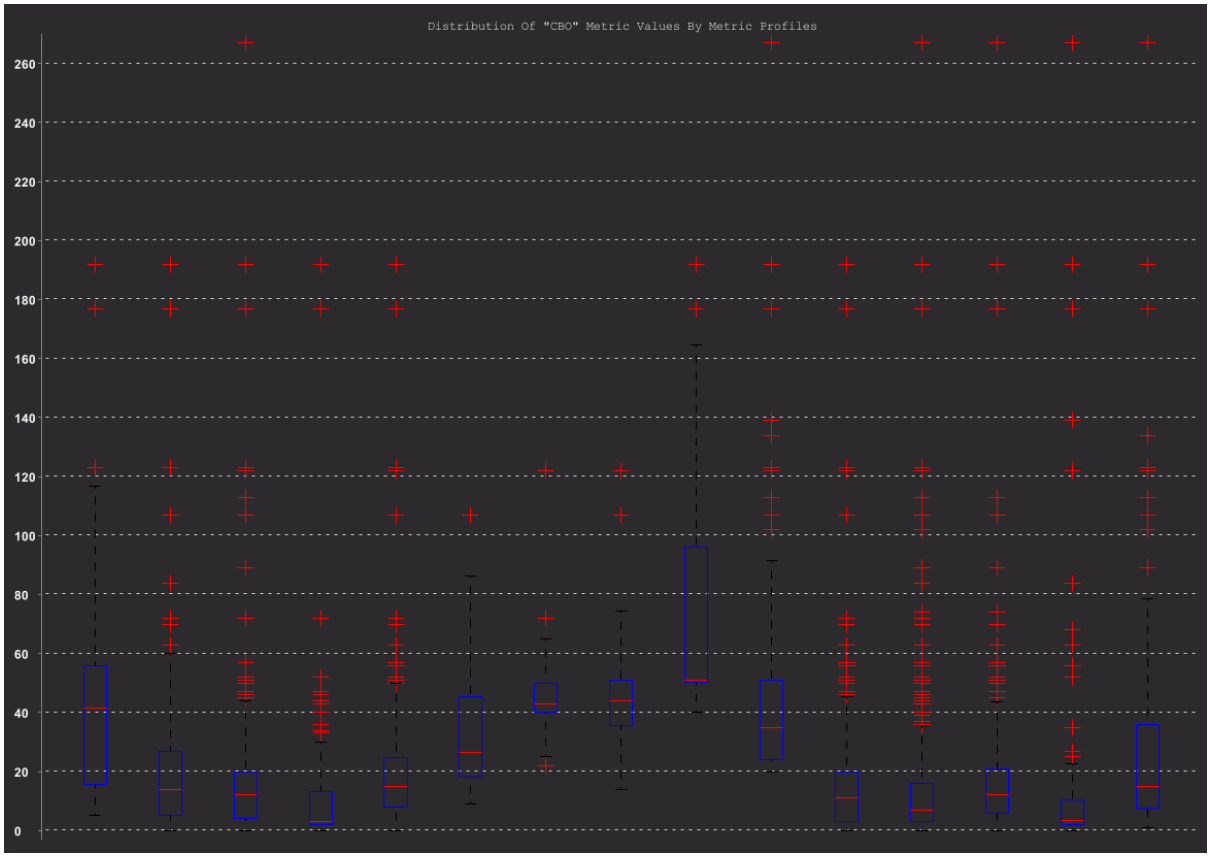
Is the number of other classes that a class is coupled to. It measures the degree of dependency between classes.
Specifically, it counts the number of distinct classes a class depends on or uses in some way (e.g., through method calls, class instances, or inheritance).
The CBO metric helps assess the maintainability and complexity of a class by highlighting its dependencies. Generally, keeping coupling low is recommended to improve modularity, flexibility, and ease of maintenance.

Charts:



High Very-high Extreme

Class	Range	Value
BlockVector3	[23..∞)	267
EditSession	[23..∞)	192
WorldEdit	[23..∞)	177
WorldEditException	[23..∞)	139
BlockType	[23..∞)	134
BlockState	[23..∞)	123
LocalSession	[23..∞)	122
Vector3	[23..∞)	113
PlatformCommandManager	[23..∞)	107
BaseBlock	[23..∞)	102
Location	[23..∞)	89
LocalConfiguration	[23..∞)	84
BlockVector2	[23..∞)	74
PlatformManager	[23..∞)	72
BrushCommands	[23..∞)	70
BiomeType	[23..∞)	68
BlockTypes	[23..∞)	68
WorldEditPlugin	[23..∞)	63
ItemType	[23..∞)	59
BukkitAdapter	[23..∞)	57
NeoForgeDataFixer	[23..∞)	57



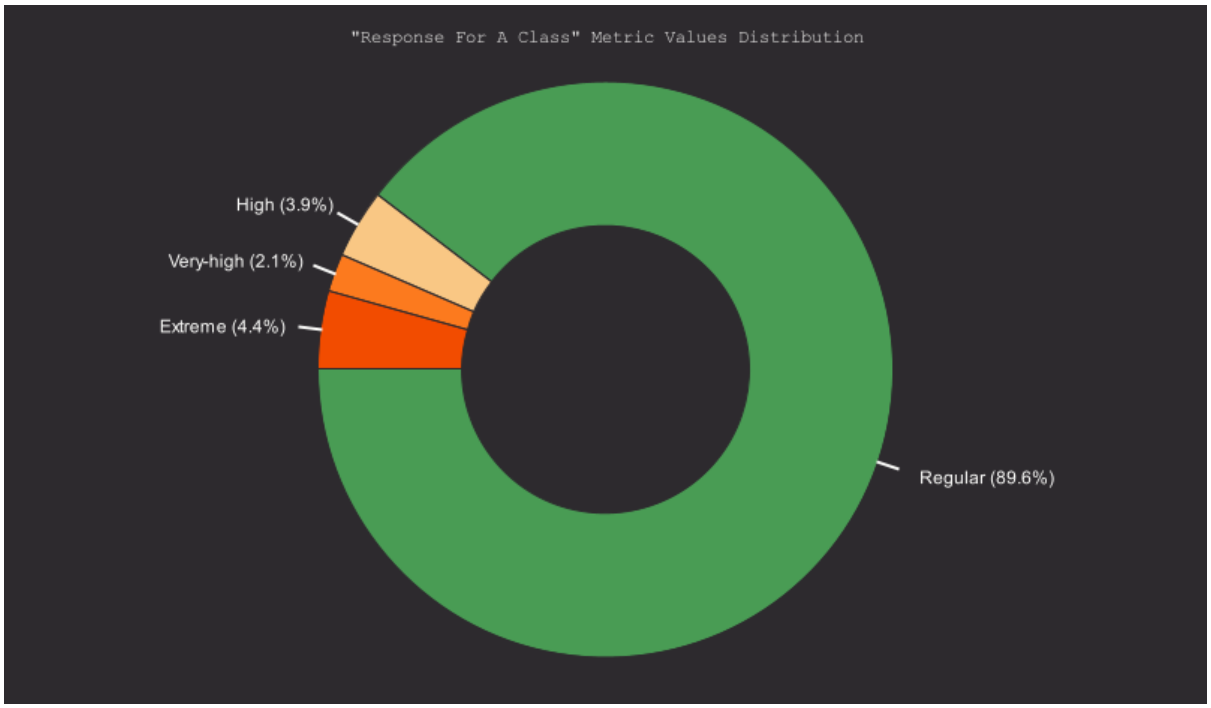
Potential trouble spots:

High CBO indicates that a class depends on many others, which makes it more complex and harder to maintain. Such high dependency reduces modularity and flexibility, as changes in one class can easily ripple across others, leading to unintended side effects. Additionally, it hinders reusability, as highly coupled classes are difficult to adapt or move to new contexts without also including their dependencies.
The most extreme case being the BlockVector3 record in the com.sk89q.worldedit.math package, with a CBO of 267.
The boxplot chart shows that **Complex Method** and **Long Method** profiles have the most outliers, indicating high coupling. For Complex Methods, high CBO suggests interactions with many classes, increasing complexity and potential maintenance issues. For Long Methods, high coupling with external classes complicates the structure and makes refactoring harder.

Response for a Class:

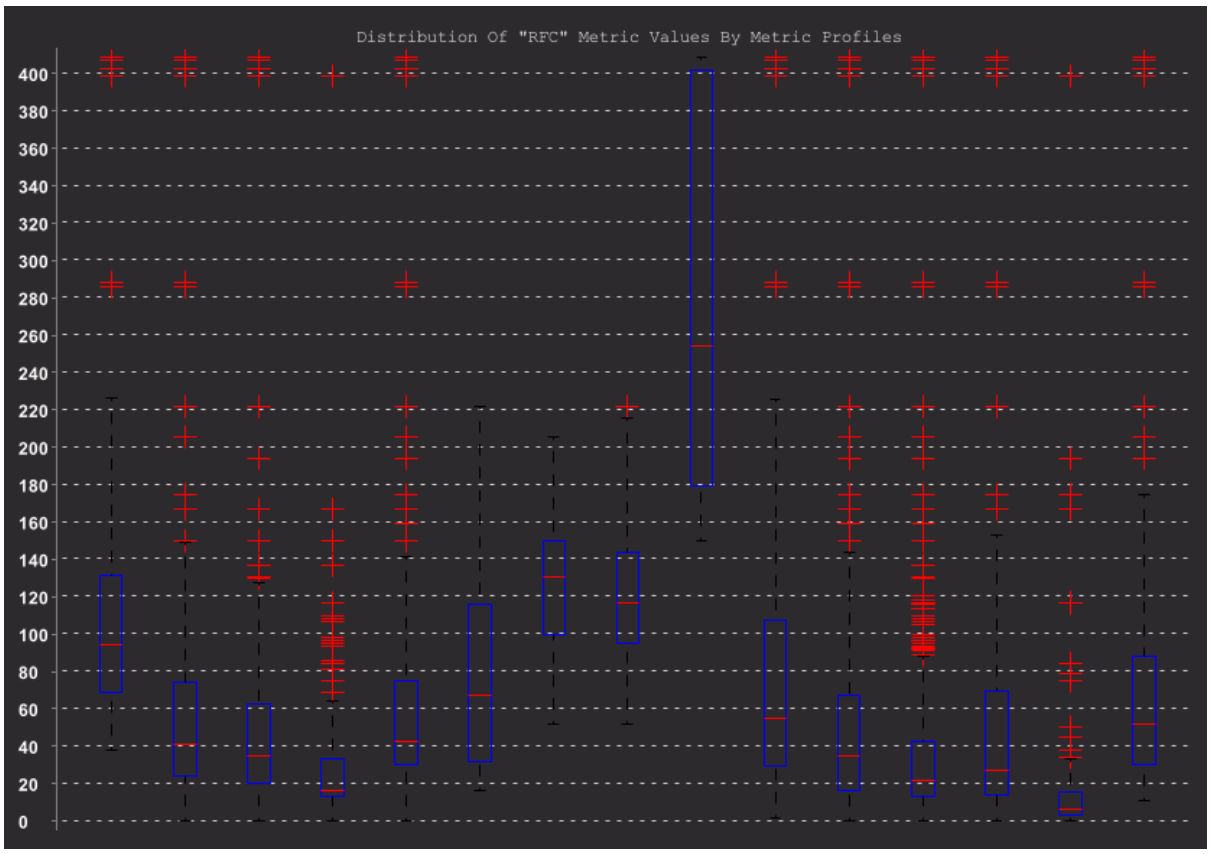
Is the total number of methods that can potentially be executed in response to a message received by an object of a class. This number is the sum of the methods of the class, and all distinct methods are invoked directly within the class methods.
Additionally, inherited methods are counted, but overridden methods are not, because only one method of a particular signature will always be available to an object of a given class.

Charts:



High Very-high Extreme

Class	Range	Value
PaperweightAdapter	[80..∞)	409
PaperweightAdapter	[80..∞)	407
PaperweightAdapter	[80..∞)	403
PaperweightAdapter	[80..∞)	403
EditSession	[80..∞)	399
FabricWorld	[80..∞)	288
NeoForgeWorld	[80..∞)	286
PlatformCommandManager	[80..∞)	222
SpongeWorld	[80..∞)	206
LocalSession	[80..∞)	194
WorldEditPlugin	[80..∞)	175
WorldEdit	[80..∞)	167
BukkitWorld	[80..∞)	159
SpongeWorldEdit	[80..∞)	150
NeoForgeWorldEdit	[80..∞)	137
SelectionCommands	[80..∞)	137
FabricWorldEdit	[80..∞)	131
BukkitAdapter	[80..∞)	130



Potential trouble spots:

High RFC values can lead to difficult maintenance and testing, as these classes are more interconnected and harder to isolate.
The worst case seem to be the PaperweightAdapter class, in the com.sk89q.worldedit.bukkit.adapter.impl.v1_20_R4 package, with a RFC of 409.
The boxplot chart shows that the **Long Method** profile has the most outliers, with RFC values significantly exceeding the typical range. This indicates that classes associated with this profile tend to have a large number of method calls or complex interactions within methods, aligning with the characteristics of a long method code smell. Such high RFC values suggest that these methods are likely doing too much or interacting extensively with other components, which can lead to maintenance difficulties and reduced modularity.

Relation with code smell:

The **high Response for Class (RFC) value of 194** in LocalSession suggests it's doing too much and relies heavily on methods from other classes, especially EditSession. This reliance reflects the **Feature Envy** smell, as LocalSession frequently accesses external methods, indicating it's overextended and lacks cohesion.

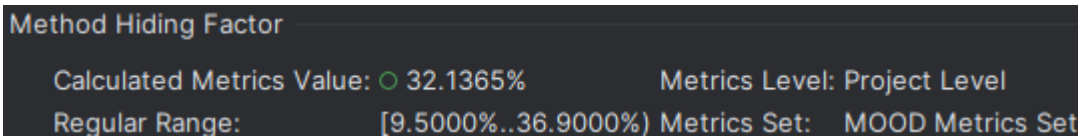
[identification: 62909; review: 60593]

MOOD (Metrics for Object-Oriented Design) metrics set (Project level metrics)

Method Hiding Factor

MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system. The invisibility of a method is the percentage of the total classes from which this method is not visible.

Chart



Potential trouble spots

Low MHF could signal that too many methods are publicly accessible, potentially exposing the internal state or implementation unnecessarily. High MHF could signal that too few methods are publicly accessible, potentially making classes underutilized or redundant. An MHF value of around 32% lies within the recommended range of 9.5% to 36.9%, which is generally considered an optimal interval according to studies. This suggests a balanced approach in terms of method visibility and encapsulation within the project.

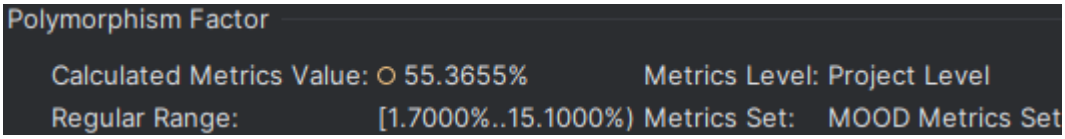
Relation with code smells

A low MHF often signals that classes might be exposing too much of their internal logic. This can lead to **Inappropriate Intimacy**, where classes become too intertwined and rely on each other's inner workings, making future changes difficult and prone to errors. It can also lead to **Feature Envy** since classes with many public methods might tempt other classes to interact with their internal state or logic excessively. A high MHF could indicate **Refused Bequest**. This happens when a subclass doesn't fully utilize the functionality of its superclass, suggesting an inappropriate use of inheritance.

Polymorphism Factor

PF is defined as the quotient between the actual number of different possible polymorphic situations, and the maximum number of possible distinct polymorphic situations for given class.

Chart



Potential trouble spots

Low PF might indicate an underutilization of polymorphism. This could suggest missed opportunities for flexibility and extensibility, where inheritance and method overriding could be used to avoid duplication or improve design. High PF might indicate excessive use of polymorphism, where methods are overridden too frequently. This can lead to an overly complex hierarchy that is difficult to maintain and debug. A PF of around 55% is well above the recommended range of 1.7% to 15.1%. It means that more than half of the methods across classes are overridden in subclasses. This level of overriding can make the code very complex to understand and maintain.

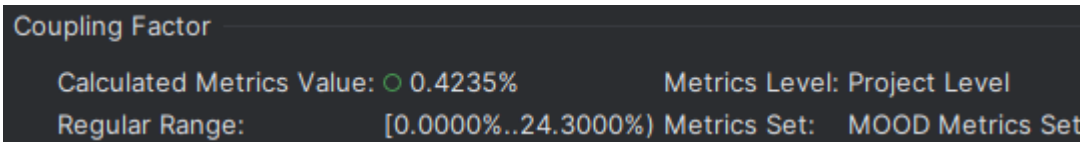
Relation with code smells

A high PF might lead to **Inappropriate Intimacy**, where subclasses know too much about their superclass's internal workings and must override methods just to adapt to the superclass's implementation. With so many overridden methods, it can also result in **Shotgun Surgery**, where a single change cascades through many parts of the codebase.

Coupling Factor

CF measures the coupling between classes excluding coupling due to inheritance. It is the ratio between the number of actually coupled pairs of classes in a scope (e.g. package) and the possible number of coupled pairs of classes.

Chart



Potential trouble spots

High CF indicates excessive dependency among classes, which can reduce the software's quality by making it harder to understand, modify, and test. A CF of around 0.4% indicates that each class is interacting with only a very small proportion of the other classes in the system. This is typically seen in well-encapsulated and modular designs where the classes are self-contained and don't depend heavily on each other.

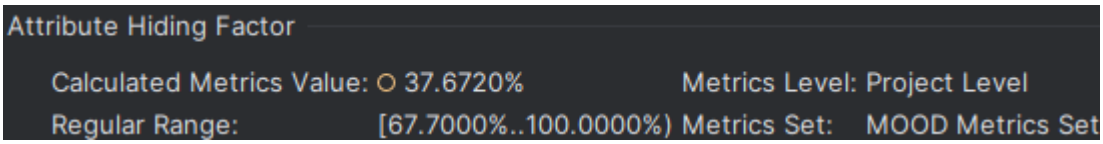
Relation with code smells

A high CF can cause classes to become overly dependent on each other's internal details, leading to **Inappropriate Intimacy**. A high CF can also lead to **Feature Envy**, as classes excessively rely on each other's internals.

Attribute Hiding Factor

AHF is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration.

Chart



Potential trouble spots

Low AHF suggests that too many attributes are directly accessible, which can lead to undesirable consequences such as increased complexity, reduced encapsulation, and a higher risk of creating bugs or unintended behavior due to uncontrolled access to the class's internals. An AHF of around 38% indicates that the project is applying relatively low information hiding, which could lead to several potential issues in terms of software design and maintainability.

Relation with code smells

A low AHF exposes related attributes without encapsulating them properly within their own class. This may lead to **Data Clumps**, where the same set of attributes is passed around or manipulated across multiple places in the code. A low AHF also leads to the excessive creation of getter and setter methods because more attributes are exposed, which may lead to **Data Classes**.

[identification: 65330; review: 62909]

Lorenz-Kidd metrics set (Class level metrics)

NOA: Number of Attributes

Description of the metric

NOA measures the total count of attributes (or properties) in a class within object-oriented programming. This metric is used to assess the complexity of a class, as a higher number of attributes often indicates increased complexity, potential for higher memory usage, and potential challenges in maintenance and testing. It provides insight into the data structure and modularization within a codebase, helping to identify classes that may benefit from refactoring or simplification.

Graphs

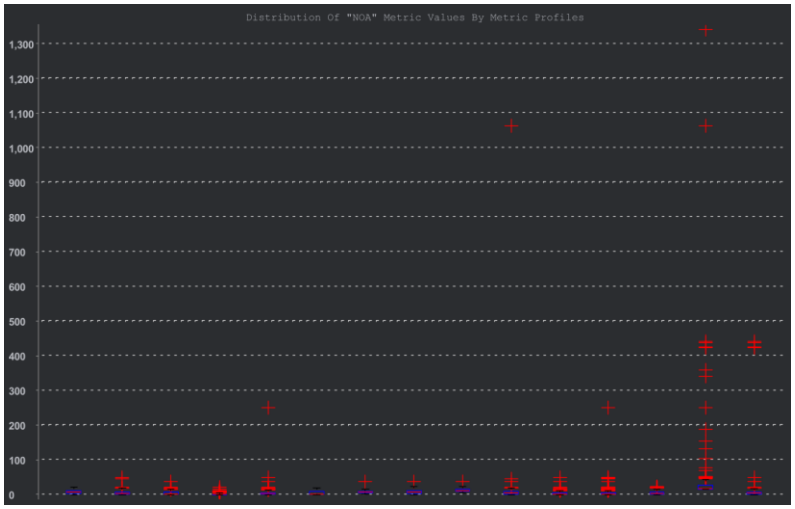


fig1

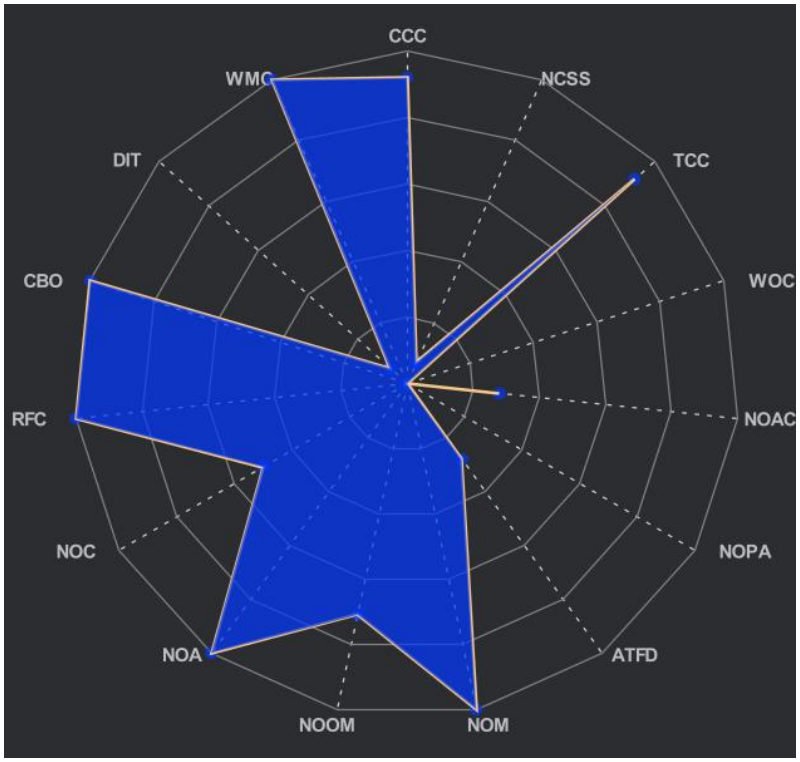

CCC
NCSS
TCC
WOC
NOAC
NOPA
ATFD
NOM
NOOM
NOA
NOC
RFC
CBO
DIT
WMC

fig2

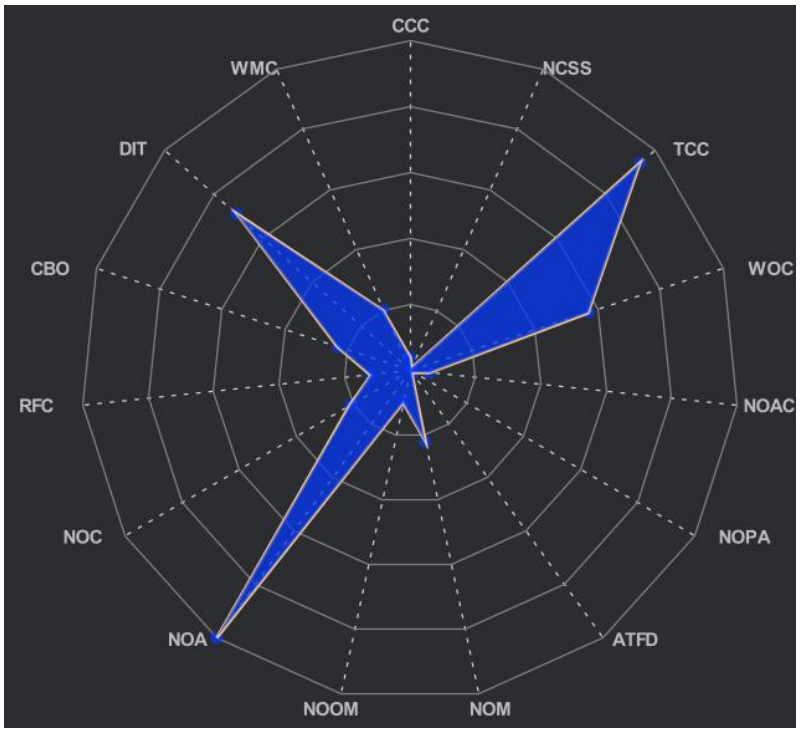

CCC
NCSS
TCC
WOC
NOAC
NOPA
ATFD
NOM
NOOM
NOA
NOC
RFC
CBO
DIT
WMC

fig3

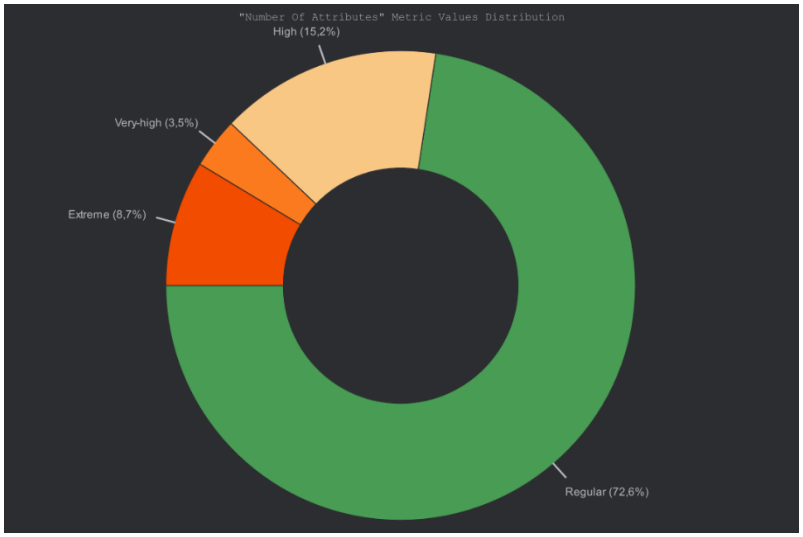


fig4

O High O Very-high O Extreme

Class	Range	Value
ItemTypes	[14..∞)	1341
BlockTypes	[14..∞)	1064
PaperweightFakePlayer	[14..∞)	441
PaperweightFakePlayer	[14..∞)	438
PaperweightFakePlayer	[14..∞)	426
PaperweightFakePlayer	[14..∞)	423
WorldEditFakePlayer	[14..∞)	359
WorldEditFakePlayer	[14..∞)	341
NavigableEditorPane	[14..∞)	251
BlockCategories	[14..∞)	189
ItemCategories	[14..∞)	154
EntityTypes	[14..∞)	131
BiomeTypes	[14..∞)	103
RhinoContext	[14..∞)	78
BiomeCategories	[14..∞)	70
FabricConfiguration	[14..∞)	51
NeoForgeConfiguration	[14..∞)	51
BukkitConfiguration	[14..∞)	51
SpongeConfiguration	[14..∞)	51
PropertiesConfiguration	[14..∞)	49
ConfigurateConfiguration	[14..∞)	49
CLIConfiguration	[14..∞)	49
YAMLConfiguration	[14..∞)	48
LocalConfiguration	[14..∞)	46
LocalSession	[14..∞)	37

Fig5

Potential trouble spots

In figure 1 we can observe that the last two (TooManyFields and TooManyMethods) have the highest number and largest spread of outliers, thus we can conclude that this metric is unreliable for identifying those code smells, as it can indicate that there is and inconsistency in measurement and a lack of specific which can lead to false positives.

Relation with code smell

In figure 1 we can see that BrainClass and FeatureEnvy are at the bottom of the chart. This can indicate that NOA is a good metric for identifying those code smells.
In figures 2 and 3 we can see that NOA has a value of 100% which indicates that NOA is a good metric to identify GodClass type 4 and TooManyFields code smells.
In figure 4 and 5 we can see that there is a decent percentage of extreme values, most notably in figure 4 we can see the classes "ItemTypes" and "BlockTypes" have a disproportional value of Numbers of attributes, which could indicatethat they have an unusual amount of code smells that can be deduced from NOA, which makes sense as a class with a highnumber of attributes can indicate that it has too many responsibilities and is doing too much(God Class type 4 and BrainClass) or that it is too reliant on other classes (Feature Envy) or that it has too manyfields/attributes (TooManyFields).

NOO: Number of Operations

Description of the metric

NOO represents the total count of methods or functions defined in a class. This metric evaluates the behavioral complexity of a class, where a higher NOO indicates more functionality within a single class. A high NOO may suggest that a class is handling too many responsibilities, potentially violating principles like single-responsibility and leading to issues in maintainability, readability, and testing.

Graph

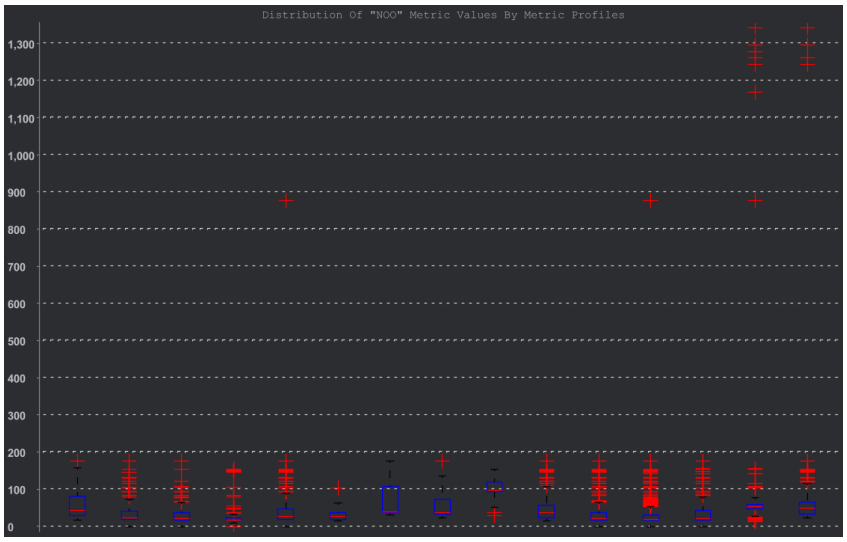


fig6

Potential trouble spots

In figure 6 we can see that most code smells have a lot of outliers, this can indicate that this metric is unreliable for identifying those code smells, except `God Class type 1` and `Feature Envy` which have a low number of outliers along with a low dispersion of values. The reasons for the unreliability is inconsistent detection and a lack of correlation with code smells which leads to false alarms and possibly even missed detections.

Relation with code smell

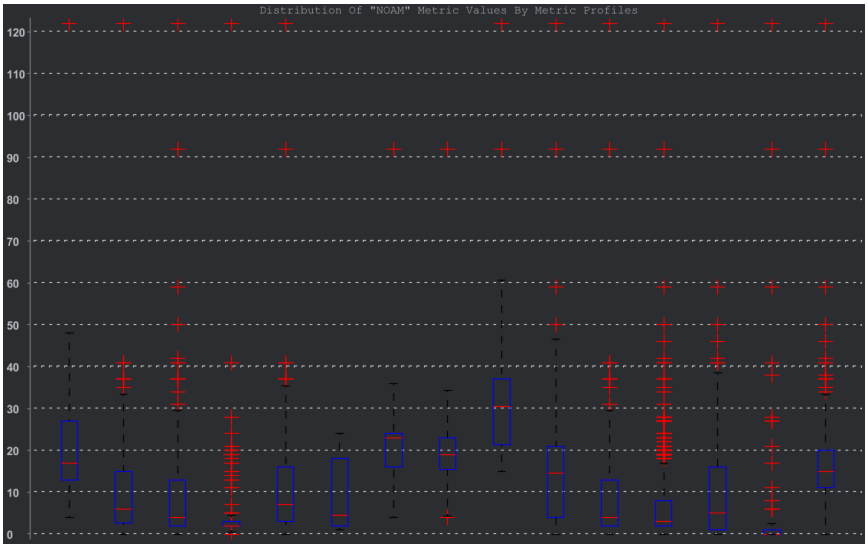
In figure 6 we can see that `God Class type 1` and `Feature Envy` are at the bottom of the chart this can indicate that NOO is a good metric for identifying those code smells, which makes sense as a class with a high number of operations can indicate that it has too many responsibilities and is doing too much (God Class type 1) or that it is too reliant on other classes (Feature Envy).

NOAM: Number of Added Methods

Description of the metric

NOAM counts the number of getter and setter methods in a class. This metric reflects how much of a class's data is exposed or modifiable, which can impact encapsulation. A high NOAM suggests that a class has many publicly accessible fields, possibly indicating weaker data hiding and encapsulation, which may lead to tighter coupling and increased dependencies in the codebase.

Graph



Potential trouble spots

In figure 7 we can see that most code smells have a lot of outliers, this can indicate that this metric is unreliable for identifying most code smells, however we can see some exceptions, namely: `Brain Clas, Feature Envy and God Class type 1 and 3 which have a low number of outliers. The reasons for the unreliability are an inconsistent detection and a high sensitivity to irrelevant code aspects, so due to the high variability of the metric, it can lead to false alarms and possibly even missed detections

Relation with code smell

In figure 7 we can see that BrainClass, FeatureEnvy, GodClass type 1 and GodClass type 3 are at the bottom of the chart and have little to no outlier this can indicate that NOAM is a good metric for identifying those code smells, which makes sense as a class with a high number of added methods can indicate that it has too many responsibilities and is doing too much(God Class type 1 and 3) or that it is too reliant on other classes (Feature Envy) or that it has too many methods(BrainClass).

[identification: 65969; review: 65330]

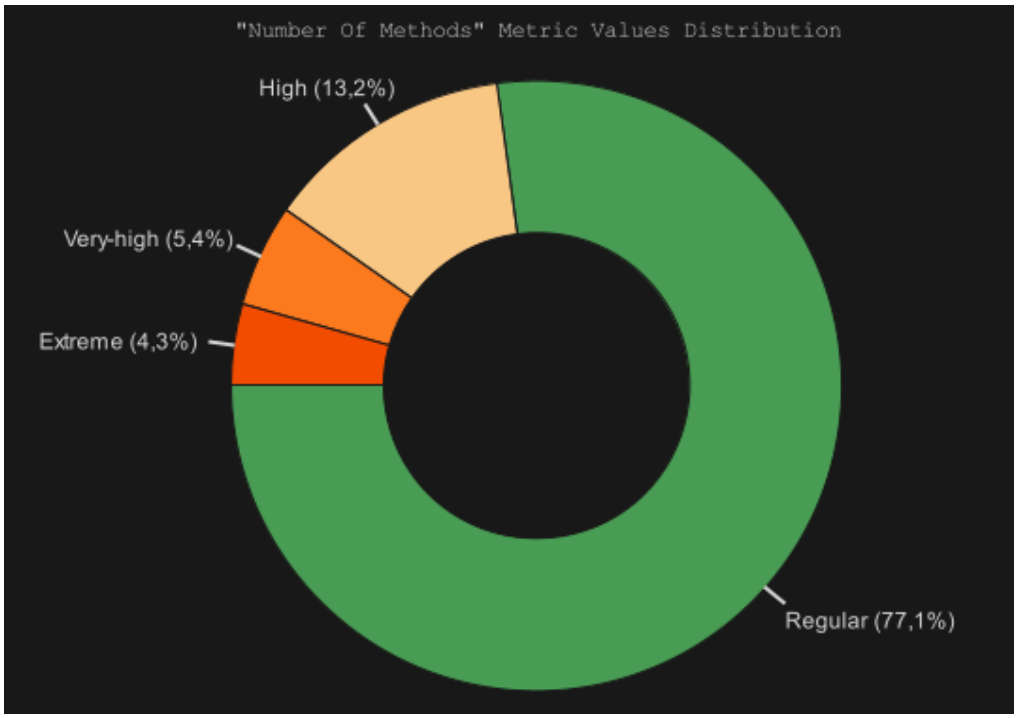
Li-Henry metrics set

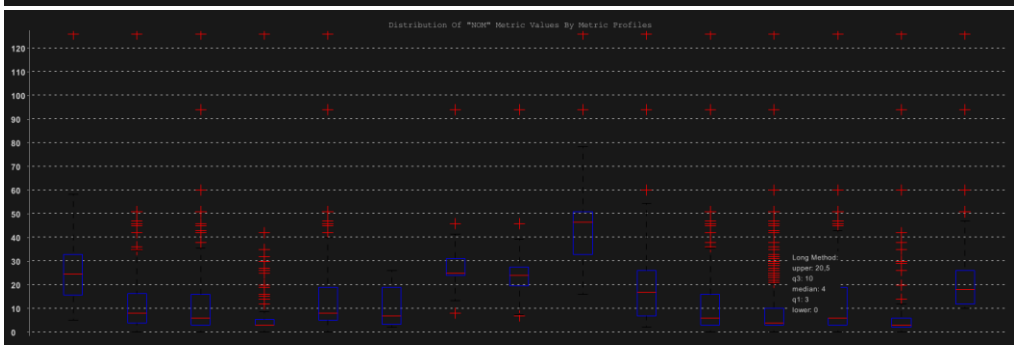
Number of Methods

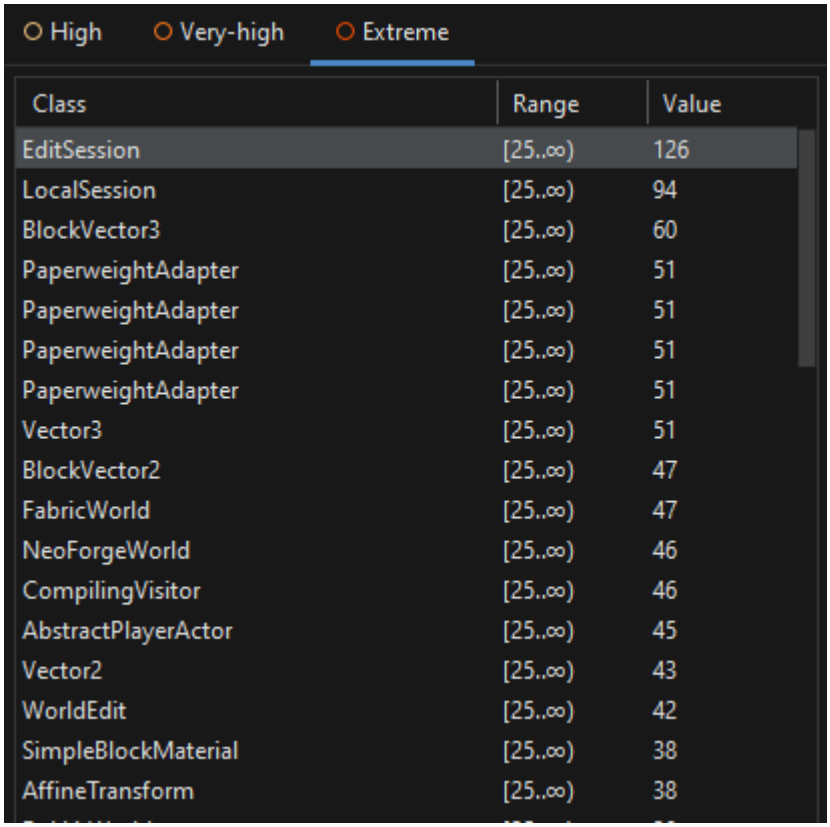
Description of the metric

The number of methods in a class is a simple metric that provides insight into the size and complexity of a class.

Graph







Potencial trouble spots

As we can see in the box plot (third image) the metric profile Long Method (last four) is the one that has the highest number of outliers, this makes perfect sense because the metric Number of Methods counts the number of methods in a class, and has nothing to do with the number of lines of code in each method.

Code smells

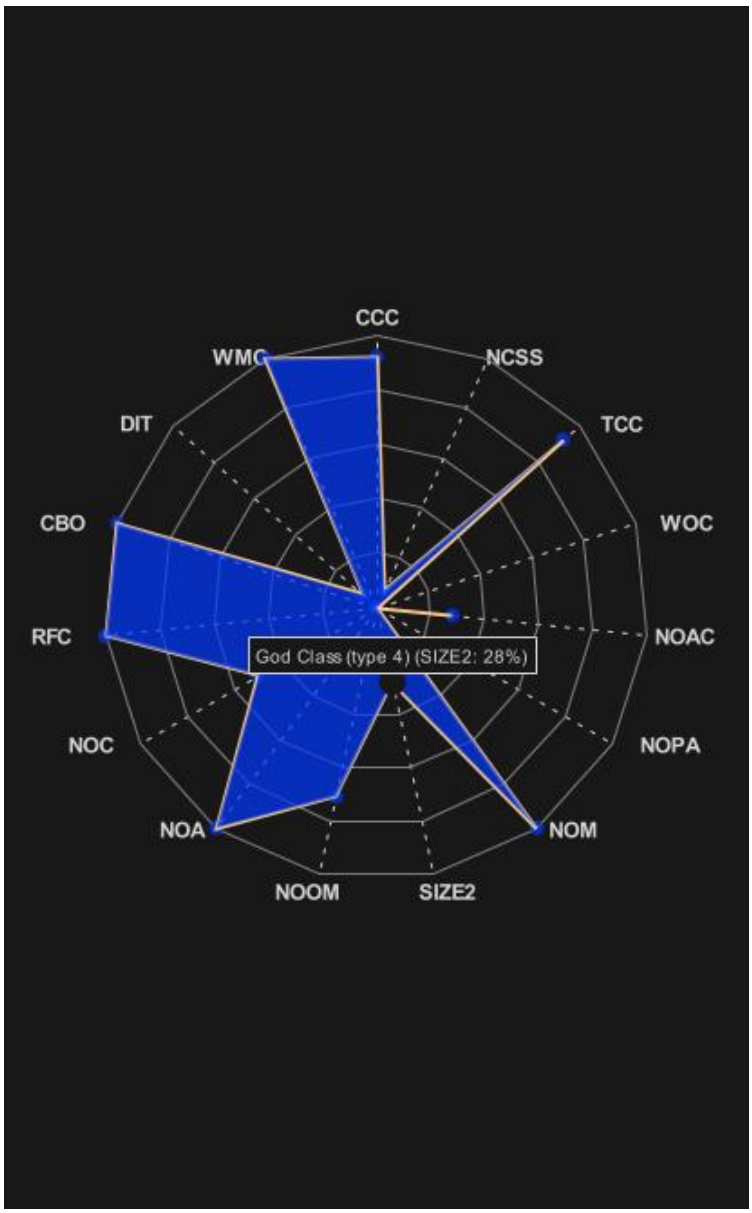
4,3% of classes have Extreme NOM values, indicating they have a high number of methods.
The worst three are EditSession with a value of 126, LocalSession with a value of 94 and BlockVector3 with 60.
This indicates that these classes may be too complex, having most likely the code smell of God Class and too many methods.

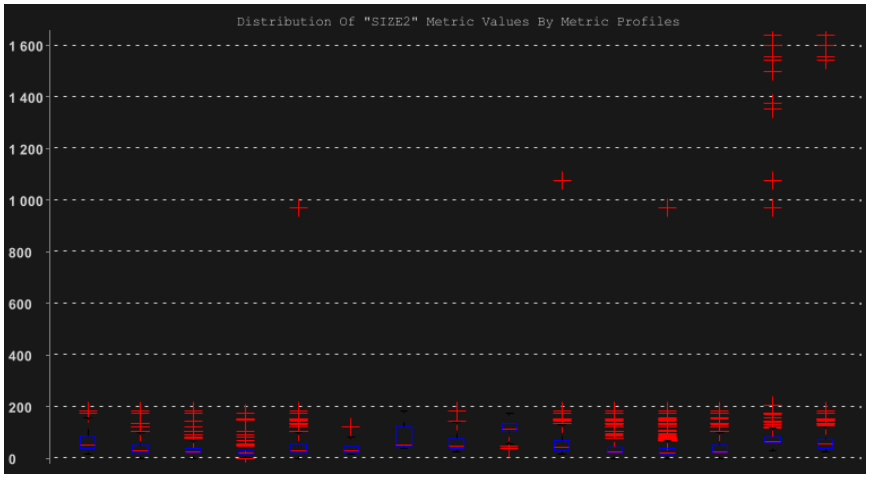
Number of Attributes and Methods

Description

Measures the class size by counting attributes and methods.

Charts





Potencial trouble spots

As we can see in the graph the too many fields and too many methods (the last to in the graph) are the metrics profile that has a higher number of outliers that are the most fare away of the box, meaning that the metric SIZE2 is not reliable for this metrics profiles.
This is probably because the SIZE2 metric is a combination of the number of attributes and methods, if the metric has a high value manly because of the number of attributes and the number of methods will be low does not make sense basing this value on metrics profile too many methods (vice versa for the too many fields)

Code smells

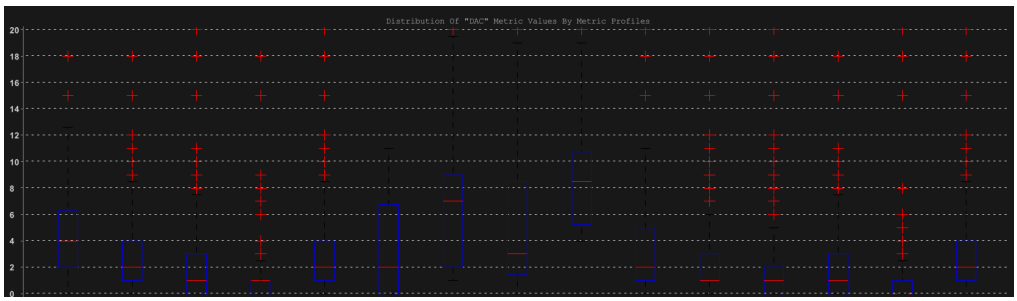
This metric indicates that the classes with high SIZE2 values may have the code smell of God Class.
As we can see in the spider graph this metric holds a value of 28% percent that compared with the other metrics is the highest value.

Data Abstraction Coupling

Description

Measures how much a class depends on other abstract classes. High DAC values suggest high dependencies, which might hinder flexibility.

Charts



Potencial trouble spots

As we can see in the graph only the metric profiles Brain Class,Feature Envy and God Class (type 1,3,4) are the only ones that have almost non outliers. And the rest of the metric profiles have a high number of outliers, this is because this metrics profiles rely mostly on the field, parameters and method counting and method content analysis in which the DAC does not measure.

Code smells

The classes with high DAC values may have the code smell of Brain Class, Feature Envy and God Class.
Feature Envy: A high DAC value means that the class is accessing many other classes. If it frequently accesses specific classes, it could be a sign that it "envies" certain features or behaviors of those classes.
Brain Class: A high DAC value means that the class interacts with many other classes, potentially processing or coordinating data across them.
God Class: High DAC suggests that this class has dependencies on many other classes, which often happens when it tries to "do everything."

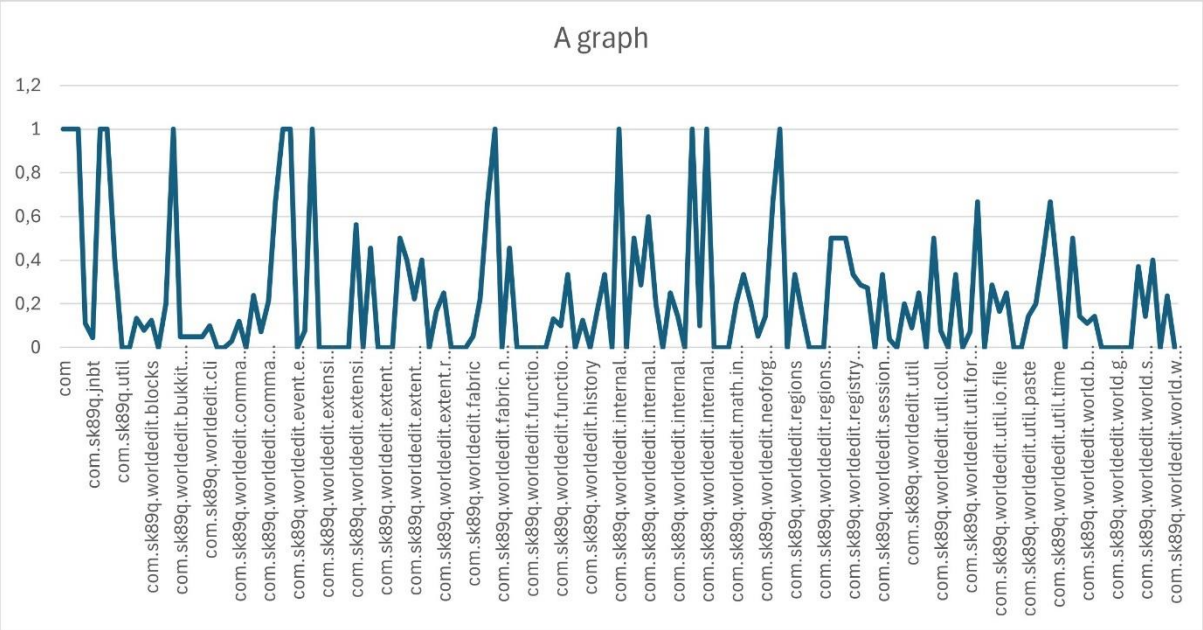
[Identification: 66081; review: 65969]

Robert C. Martin metrics set (Package level metrics)

Abstractness

Abstractness (A) is a metric that quantifies the ratio of the number of abstract classes (Na) to the total number of classes (Nc) in a package.
It varies between 0 and 1, being 0 a package completely concrete and 1 a package completely abstract.
It is calculated using the formula: A = Na / Nc

Graph:



Potential trouble spots:

- **High Abstractness**: A high value of Abstractness (A) indicates that the package is highly abstract, with many abstract classes or interfaces. While this can be beneficial for flexibility and extensibility, it can also lead to over-engineering and unnecessary complexity if not managed properly. The packages with the highest Abstractness values are
the com.sk89q.worldedit.extension, com.sk89q.worldedit.bukkit.adapter.impl, com.sk89q.worldedit.internal.wna, com.sk89q.worldedit.internal.registry, etc...

- **Low Abstractness**: A low value of Abstractness (A) suggests that the package is concrete, with few abstract classes. This can make the package rigid and less adaptable to change, as it may lack the necessary abstractions to support future modifications. The packages with the lowest Abstractness values are the com.sk89q.util, com.sk89q.worldedit.math, com.sk89q.worldedit.util.asset.holder, com.sk89q.worldedit.util.concurrency, etc...

Relation with code smells:

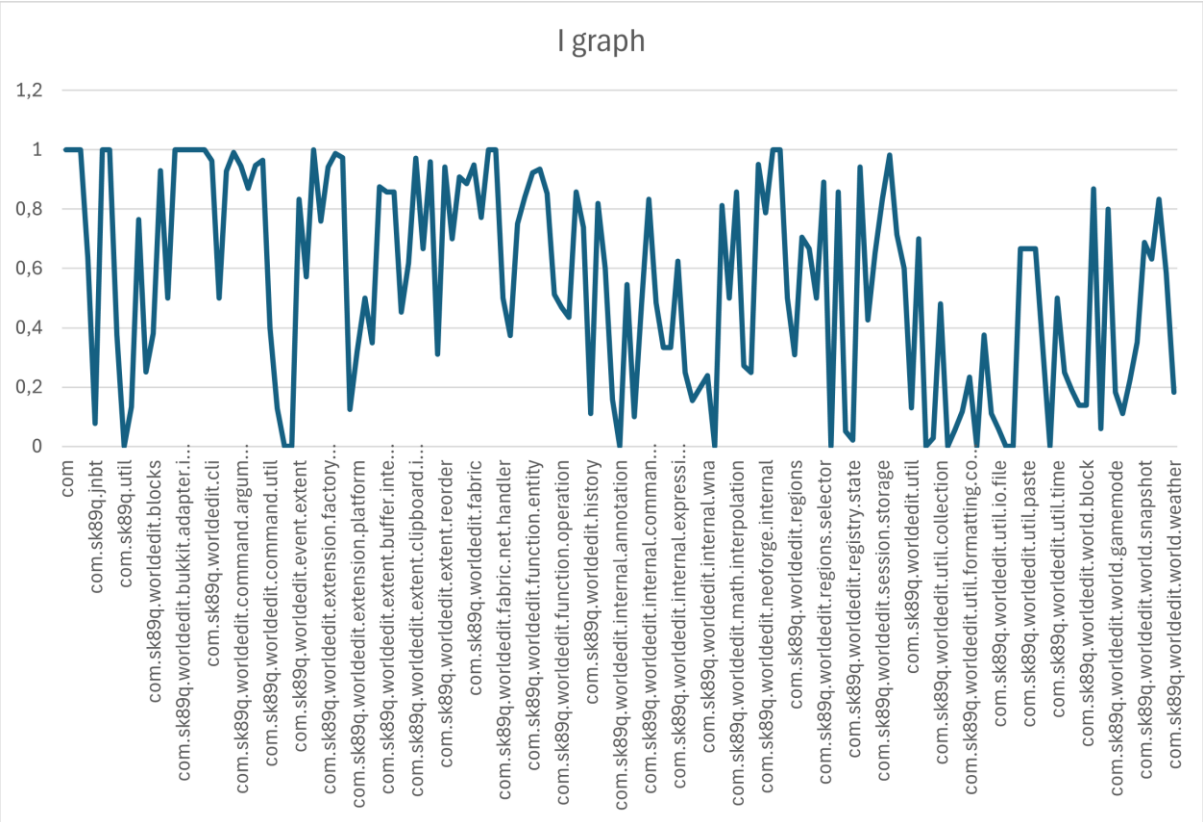
- **High Abstractness**: A high Abstractness value can sometimes indicate the presence of **Speculative Generality**, where classes are overly abstract and provide unnecessary abstractions that are not currently needed, they are "just in case" abstractions. This can lead to unnecessary complexity and make the code harder to understand and maintain. It can also lead to **Zones of Uselessness**, that's when a highly abstract package is not widely used or implemented, meaning it provides abstraction with little practical application.
- **Low Abstractness**: A low Abstractness value may suggest **Rigidity**, where the package lacks the necessary abstractions to support future changes. This can lead to code that is difficult to extend or modify without significant refactoring. Also, may lead to **Duplicate Code**, as concrete classes may repeat similar logic.

Instability:

Instability (I) is a metric that quantifies the ratio of efferent couplings (Ce) to the sum of efferent couplings (Ce) and afferent couplings (Ca) in a package.
Efferent Coupling (Ce) is the number of classes outside the package that depend on classes within the package, as each class counts only once. Afferent Coupling (Ca) is the opposite, the number of classes inside the package that depend on classes outside the package, and as in Ce, each class counts only once.
Instability varies between 0 and 1, with 0 indicating a stable package that is not dependent on others and 1 indicating an unstable package that is highly dependent on others.

It is calculated using the formula: I = Ce / (Ce + Ca)

Graph:



Potential trouble spots:

- **High Instability**: A high value of Instability (I) indicates that the package is unstable, with many dependencies on classes outside the package. This can make the package more prone to changes in external classes and can lead to a higher risk of bugs and errors. The packages more unstable are com.sk89q.worldedit.extension, com.sk89q.worldedit.bukkit.adapter.impl, com.sk89q.worldedit.fabric.mixin, etc...
- **Low Instability**: A low value of Instability (I) suggests that the package is stable, with few dependencies on classes outside the package. This can make the package more resilient to changes in external classes and can reduce the risk of bugs and errors, however, it can also lead to a lack of flexibility and adaptability. The most stable packages are com.sk89q.util, com.sk89q.worldedit.math, com.sk89q.worldedit.internal.annotation, etc...

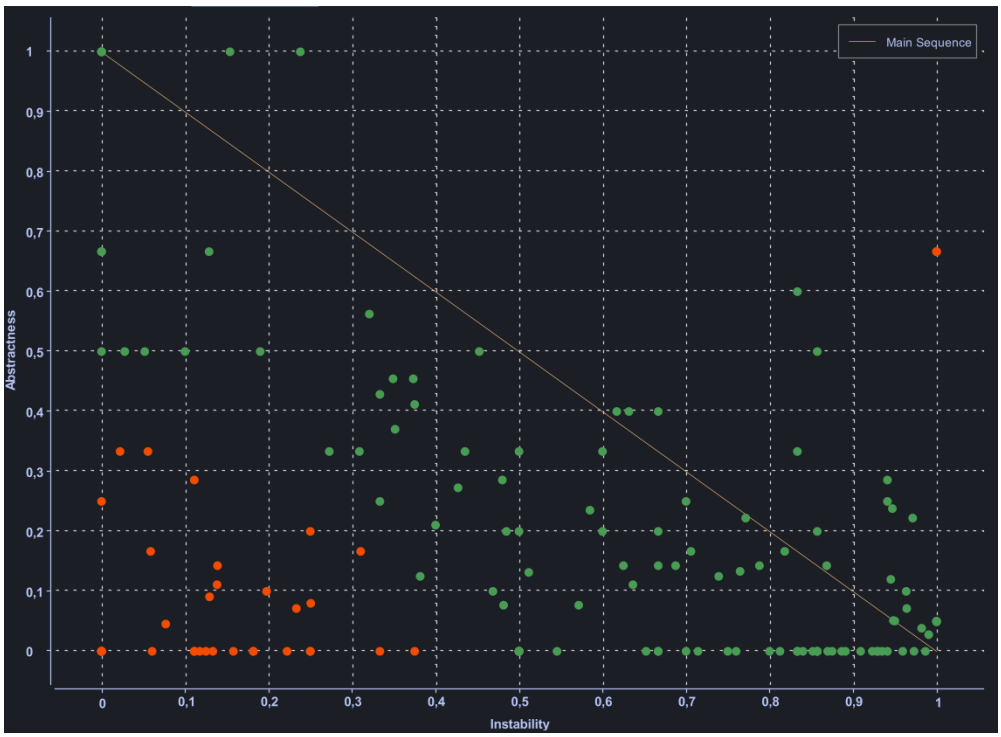
Relation with code smells:

- **High Instability**: A high Instability value can indicate **Rigidity**, where the package is highly dependent on external classes and is more likely to break when external classes change. This can make the package difficult to maintain and extend. It can also lead to **Shotgun Surgery**, where changes in external classes require many modifications across the package.
- **Low Instability**: A low Instability value can suggest **Fragility**, where the package is not well integrated with external classes and may not function correctly when external classes change. This can lead to bugs and errors that are difficult to diagnose and fix.

Normalized Distance from Main Sequence

Normalized Distance from Main Sequence or just Distance (D) is a metric that quantifies the distance of a package from the main sequence, which is a line that goes from the point (0,1) to the point (1,0) in the Abstractness-Instability graph. The main sequence represents the ideal balance between Abstractness and Instability, with packages close to it being well-balanced in terms of these two metrics. Basically, a highly unstable package should be highly abstract to avoid being too dependent on other packages, while a highly stable package should be highly concrete to avoid unnecessary complexity.
It is calculated using the formula: D = |A + I - 1|

Chart:



Analysing the chart, we can see that the majority of the packages are close to the main sequence, but are more unstable than abstract. Inclusive, we have a big part of the packages in the bottom-left quadrant, which means that the main problems related to rigidity in the project are there.

Potential trouble spots:

- **High Distance**: A high value of Distance (D) indicates that the package is far from the main sequence, which means that it is either highly abstract and unstable (top-right corner) or highly concrete and stable (bottom-left) corner). Both of these extremes can lead to many issues such as over-engineering, under-engineering, or unnecessary complexity. The packages with the highest Distance values are the com.sk89q.util, com.sk89q.worldedit.extension, com.sk89q.worldedit.math, com.sk89q.worldedit.bukkit.adapter.impl, com.sk89q.worldedit.fabric.net, etc...

Relation with code smells:

- **High Distance**: A high Distance value should indicate some code smells related to excessive volatility or rigidity in a package's design.
    - **highly concrete and stable packages**: may indicate **Rigidity**, where the package is not flexible enough to support future changes. This creates a **Zone of Pain**, zones where is difficult to modify the classes because they have many incoming dependencies (Ca is high) but are not abstract, so modifications to these concrete classes ripple throughout the system.
    - **highly abstract and unstable packages**: may indicate **Fragility**, where the package is too dependent on external classes and is more likely to break when external classes change. This can lead to **Shotgun Surgery**, where changes in external classes require many modifications across the package. We may have **Speculative Generality** too, where classes are overly abstract and provide unnecessary abstractions that are not currently needed and create **Zones of Uselessness**, where the package provides abstraction with little practical application.

Code Smells

God Class [identification: 62111; review: 65969]

Lines 1721, 1889, 2249, 852, 1168, 1217 and 1535 of the file worldedit-core/src/main/java/com/sk89q/worldedit/ EditSession.java

---

```java
public int makeCylinder(BlockVector3 pos, Pattern block, double radius, int height, boolean
filled) throws MaxChangedBlocksException {
    // Cylinder generation code
}

public int makeSphere(BlockVector3 pos, Pattern block, double radius, boolean filled) throws
MaxChangedBlocksException {
    // Sphere generation code
}

public int makeForest(BlockVector3 basePosition, int size, double density, TreeGenerator.
TreeType treeType) throws MaxChangedBlocksException {
    // Forest generation code
}

public void undo(EditSession editSession) {
    // Undo functionality
}

public int setBlock(BlockVector3 position, BlockState block) {
    // Basic block manipulation
}

public int replaceBlocks(Region region, Mask mask, Pattern pattern) throws
MaxChangedBlocksException {
    // Block replacement functionality
}

public int moveRegion(Region region, BlockVector3 offset, int multiplier, boolean copyAir,
Pattern replacement) throws MaxChangedBlocksException {
    // Region movement code
}
```

Explanation:

The EditSession class is a "**God Class**" because it handles too many unrelated responsibilities, like shape creation, region management, environmental effects, and history tracking, making it complex and hard to maintain.
This class also have a **Weighted Method Count (WMC) of 397**, which indicates that the EditSession class has a very high number of methods, contributing to its **complexity and lack of cohesion**. In essence, a high WMC in EditSession reflects that it is overloaded with various functionalities, making it difficult to understand, maintain, and extend.

Proposal of a refactoring:

To address the "**God Class**" smell in EditSession, the responsibilities can be split across multiple smaller, more focused classes. Here's a proposed refactoring:

1. **GeometryEditor Class:**
    - **Responsibility:** Manage geometric operations like creating shapes (e.g., cylinders, spheres).
    - **Methods to Move:** makeCylinder, makeSphere, makePyramid, drawLine, drawSpline.
    - **Benefit:** This isolates shape creation in a single class, making it easier to maintain and test.
2. **RegionManager Class:**
    - **Responsibility:** Handle region-related operations, such as moving or replacing blocks within a region.
    - **Methods to Move:** moveRegion, replaceBlocks.
    - **Benefit:** With region-related functionality separated, EditSession no longer needs to manage regions directly.
3. **EnvironmentManager Class:**
    - **Responsibility:** Manage environment modifications, such as forest generation, snow simulation, and liquid draining.
    - **Methods to Move:** makeForest, simulateSnow, drainArea.
    - **Benefit:** Centralizing environment manipulation improves modularity and makes it easier to test these specific functionalities.
4. **HistoryManager Class:**
    - **Responsibility:** Manage history tracking and undo/redo functionality.
    - **Methods to Move:** undo, redo.
    - **Benefit:** Encapsulating undo/redo behavior reduces the responsibility of EditSession and makes history tracking reusable.
5. **EditSession Class (Refactored):**
    - After moving these methods, EditSession will focus on managing basic block manipulation and session management (e.g., managing the session lifecycle and setting block limits).
    - **Benefit:** The refactored EditSession is now focused on managing the core editing session, with other responsibilities delegated to more specific classes.

Feature Envy [identification: 62111; review: 65330]

Lines 254 and 280 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/LocalSession.java
---

```java
public EditSession undo(@Nullable BlockBag newBlockBag, Actor actor) {
    checkNotNull(actor);
    --historyPointer;
    if (historyPointer >= 0) {
        EditSession editSession = history.get(historyPointer);
        try (EditSession newEditSession =
                 WorldEdit.getInstance().newEditSessionBuilder()
                     .world(editSession.getWorld()).blockBag(newBlockBag).actor(actor)
                     .build()) {
            prepareEditingExtents(newEditSession, actor);
            editSession.undo(newEditSession);
        }
        return editSession;
    } else {
        historyPointer = 0;
        return null;
    }
}
```

```java
public EditSession redo(@Nullable BlockBag newBlockBag, Actor actor) {
    checkNotNull(actor);
    if (historyPointer < history.size()) {
        EditSession editSession = history.get(historyPointer);
        try (EditSession newEditSession =
                 WorldEdit.getInstance().newEditSessionBuilder()
                     .world(editSession.getWorld()).blockBag(newBlockBag).actor(actor)
                     .build()) {
            prepareEditingExtents(newEditSession, actor);
            editSession.redo(newEditSession);
        }
        ++historyPointer;
        return editSession;
    }

    return null;
}
```

Explanation:

Both undo and redo methods are performing operations that involve extensive interaction with EditSession, such as creating new instances, setting up EditSession for undo/redo, and invoking methods like undo and redo directly on EditSession. This makes these methods dependent on the internal details of EditSession, demonstrating a **Feature Envy** smell, as they rely on behaviors that might logically belong closer to EditSession.

However, since EditSession is already a **God Class**, moving this logic there would worsen the design. Instead, we can improve the design by creating, the already proposed, helper class **HistoryManager** focused in managing history tracking and undo/redo functionality.

The **high Response for Class (RFC) value of 194** in LocalSession suggests it's doing too much and relies heavily on methods from other classes, especially EditSession. This reliance reflects the **Feature Envy** smell, as LocalSession frequently accesses external methods, indicating it's overextended and lacks cohesion.

Proposal of a refactoring:

1. **Create, the already proposed, class** called HistoryManager, which will be responsible for handling undo and redo operations.
2. **Encapsulate the logic** for undo and redo in this helper class, removing the need for LocalSession to directly manipulate EditSession instances for these operations.
3. **Delegate** the undo and redo calls from LocalSession to HistoryManager, thereby reducing the dependency on EditSession.

Long Method [identification: 62111; review: 66081]

Line 157 of the file worldedit-core/src/main/java/com/sk89q/worldedit/session/SessionManager.java

---

```
public synchronized LocalSession get(SessionOwner owner) {
    checkNotNull(owner);

    LocalSession session = getIfPresent(owner);
    LocalConfiguration config = worldEdit.getConfiguration();
    SessionKey sessionKey = owner.getSessionKey();

    // No session exists yet -- create one
    if (session == null) {
        try {
            session = store.load(getKey(sessionKey));
            session.postLoad();
        } catch (IOException e) {
            LOGGER.warn("Failed to load saved session", e);
            session = new LocalSession();
        }
        Request.request().setSession(session);

        session.setConfiguration(config);
        session.setBlockChangeLimit(config.defaultChangeLimit);
        session.setTimeout(config.calculationTimeout);
        try {
            String sessionItem = session.isWandItemDefault() ? null : session.getWandItem();
            setDefaultWand(sessionItem, config.wandItem, session, new SelectionWand());
        } catch (InvalidToolBindException e) {
            if (warnedInvalidTool.add("selwand")) {
                LOGGER.warn("Invalid selection wand tool set in config. Tool will not be assigned: " + e.getItemType());
            }
        }
        try {
            String sessionItem = session.isNavWandItemDefault() ? null : session.getNavWandItem();
            setDefaultWand(sessionItem, config.navigationWand, session, new NavigationWand());
        } catch (InvalidToolBindException e) {
            if (warnedInvalidTool.add("navwand")) {
                LOGGER.warn("Invalid navigation wand tool set in config. Tool will not be assigned: " + e.getItemType());
            }
        }
        session.compareAndResetDirty();

        // Remember the session regardless of if it's currently active or not.
        // And have the SessionTracker FLUSH inactive sessions.
        sessions.put(getKey(owner), new SessionHolder(sessionKey, session));
    }

    if (shouldBoundLimit(owner, "worldedit.limit.unrestricted", session.getBlockChangeLimit(), config.maxChangeLimit)) {
        session.setBlockChangeLimit(config.maxChangeLimit);
    }
    if (shouldBoundLimit(owner, "worldedit.timeout.unrestricted", session.getTimeout(), config.maxCalculationTimeout)) {
        session.setTimeout(config.maxCalculationTimeout);
    }

    // Have the session use inventory if it's enabled and the owner
    // doesn't have an override
    session.setUseInventory(config.useInventory
            && !(config.useInventoryOverride
            && (owner.hasPermission("worldedit.inventory.unrestricted")
            || (config.useInventoryCreativeOverride && (!(owner instanceof Player) || ((Player) owner).getGameMode() == GameModes.CREATIVE)))));

    // Force non-locatable actors to use placeAtPos1
    if (!(owner instanceof Locatable)) {
        session.setPlacement(new Placement(PlacementType.POS1, BlockVector3.ZERO));
    }

    return session;
}
```

Explanation:

The get method performs multiple responsibilities, including:

- Checking and retrieving an existing session.
- Loading session configurations and handling exceptions.
- Configuring various session properties (inventory, permissions, tool settings).
- Adding the session to a collection.

This violates the **Single Responsibility Principle** because the get method is handling both session retrieval and session configuration. A long method like this also affects readability and maintainability.

Proposal of a refactoring:

Break down the get method into smaller, more focused private methods within SessionManager, such as:

- retrieveOrLoadSession(SessionOwner owner)
- configureSessionSettings(LocalSession session, SessionOwner owner)
- addSessionToCollection(LocalSession session, SessionOwner owner)

This would improve readability and make each method easier to understand and maintain.

Data Class [identification: 62909; review: 66081]

Line 39-116 of the file worldedit-core/src/main/java/com/sk89q/worldedit/blocks/BaseItem.java

---

```java
public class BaseItem implements NbtValued {  1 inheritor  new *

    private ItemType itemType;  4 usages
    @Nullable  5 usages
    private LazyReference<LinCompoundTag> nbtData;

    /** Construct the object. ...*/
    public BaseItem(ItemType itemType) {...}

    /** Construct the object. ...*/
    @Deprecated  new *
    public BaseItem(ItemType itemType, @Nullable CompoundTag nbtData) {...}

    /** Construct the object. ...*/
    public BaseItem(ItemType itemType, @Nullable LazyReference<LinCompoundTag> tag) {...}

    /** Get the type of item. ...*/
    public ItemType getType() { return this.itemType; }

    /** Set the type of the item. ...*/
    public void setType(ItemType itemType) { this.itemType = itemType; }

    @{...}
    public LazyReference<LinCompoundTag> getNbtReference() { return this.nbtData; }
```

```java
@Override  7 usages  new *
public void setNbtReference(@Nullable LazyReference<LinCompoundTag> nbtData) {
    this.nbtData = nbtData;
}

@Override  new *
public String toString() {...}
```

Explanation:

The BaseItem class' functionality is centered around managing data rather than performing significant operations or computations. The class primarily serves as a holder for data without meaningful behavior, making it a **data class**.

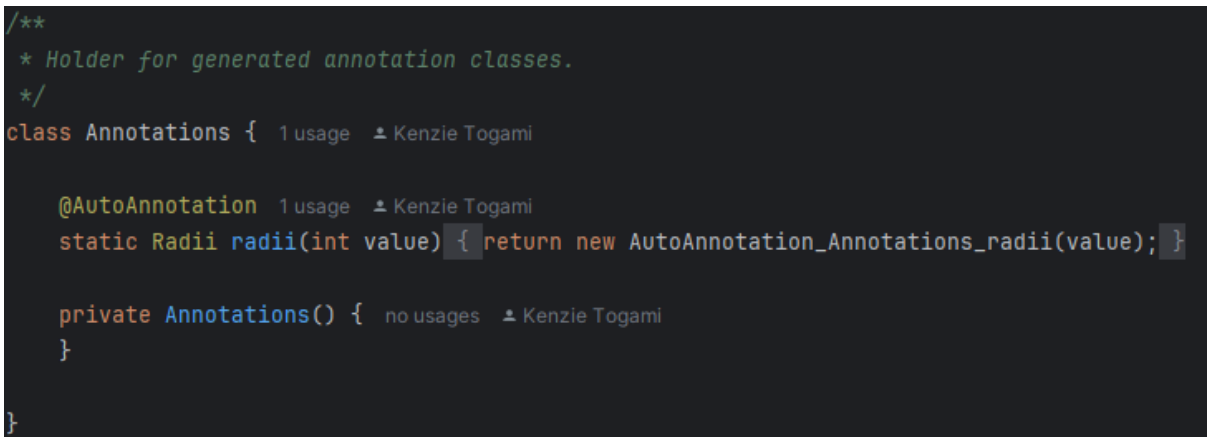
Refactoring Proposal:

To mitigate the data class smell, we can consider introducing more methods that operate on the BaseItem's state to enrich its functionality. For example, methods that manipulate item data, validate item properties, or interact with game mechanics to add significant value.

Lazy Class [identification: 62909; review: 62111]

Line 28-38 of the file worldedit-core/src/main/java/com/sk89q/worldedit/extension/platform/Annotations.java

---



Explanation:

The Annotations class is minimal in its functionality — it doesn’t manage state, hold data, or perform significant logic. It merely serves to generate an annotation with a single method. It doesn't provide much functionality beyond the single method since it only has a single usage, so it might be considered under-utilized.

Refactoring Proposal:

To mitigate the data class smell, we can consider refactoring to be simply an auxiliary method in the class where it is used, removing the need for a separate Annotations class altogether:

This:

```
private Radii radii(int value) {  1 usage   new *
    return new AutoAnnotation_Annotations_radii(value);
}

private void registerArgumentConverters() {  1 usage   Kenzie Togami +5 *
    DirectionVectorConverter.register(worldEdit, commandManager);
    DirectionConverter.register(worldEdit, commandManager);
    FactoryConverter.register(worldEdit, commandManager);
    for (int count = 2; count <= 3; count++) {
        commandManager.registerConverter(Key.of(double.class, radii(count)),
                CommaSeparatedValuesConverter.wrapAndLimit(ArgumentConverters.get(
                        TypeToken.of(double.class)
                ), count)
        );
    }
```

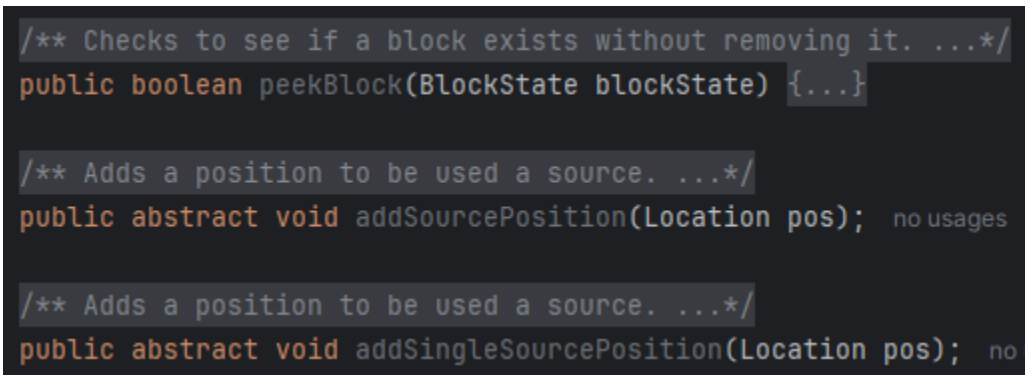
Instead of this:

```
private void registerArgumentConverters() {  1 usage   Kenzie Togami +5
    DirectionVectorConverter.register(worldEdit, commandManager);
    DirectionConverter.register(worldEdit, commandManager);
    FactoryConverter.register(worldEdit, commandManager);
    for (int count = 2; count <= 3; count++) {
        commandManager.registerConverter(Key.of(double.class, Annotations.radii(count)),
            CommaSeparatedValuesConverter.wrapAndLimit(ArgumentConverters.get(
                TypeToken.of(double.class)
            ), count)
        );
    }
```

Speculative Generality [identification: 62909; review: 60593]

Line 98-113 and 119-131 of the file worldedit-core/src/main/java/com/sk89q/worldedit/extent/inventory/BlockBag.java

---



Line 175-181 of the file worldedit-bukkit/src/main/java/com/sk89q/worldedit/bukkit/BukkitPlayerBlockBag.java

---

Explanation:

Speculative generality occurs when code includes abstractions or methods in anticipation of future needs, but without a clear, immediate use case. Since neither the base BlockBag class nor its subclass BukkitPlayerBlockBag currently make use of these methods, it suggests they were added with potential, but unfounded, future needs in mind. Currently they add extra complexity without delivering any direct benefit. Unused methods can create confusion for developers, who may assume they have a specific purpose or wonder if they’re needed for some functionality.

Refactoring Proposal:

Reducing speculative generality helps keep the codebase focused, cleaner, and easier to understand. In this case we can:

- Document as optional and make them protected: If these methods could potentially be useful in some contexts (e.g., by future subclasses), they could be documented as "optional" and changed from public to protected. This would limit their visibility to subclasses within the same package and indicate that they aren't core methods of BlockBag.
- Remove them entirely: Deleting these methods from both BlockBag and BukkitPlayerBlockBag where they are implemented, which reduces the class size and complexity, making it easier to read and maintain.

Large Class [identification: 65330; review: 65969]

Lines 41-140 of the file
\worldedit-bukkit\src\main\java\com\sk89q\wepif\
PermissionsResolverManager.java

```
public class PermissionsResolverManager implements PermissionsResolver {  12 usages  zml2008 +8

    private static final String CONFIG_HEADER = "#\r\n"  1 usage
        + "# WEPIF Configuration File\r\n"
        + "#\r\n"
        + "# This file handles permissions configuration for every plugin using WEPIF\r\n"
        + "#\r\n"
        + "# About editing this file:\r\n"
        + "# - DO NOT USE TABS. You MUST use spaces or Bukkit will complain. If\r\n"
        + "#   you use an editor like Notepad++ (recommended for Windows users), you\r\n"
        + "#   must configure it to \"replace tabs with spaces.\" In Notepad++, this can\r\n"
        + "#   be changed in Settings > Preferences > Language Menu.\r\n"
        + "# - Don't get rid of the indents. They are indented so some entries are\r\n"
        + "#   in categories (like \"enforce-single-session\" is in the \"protection\"\r\n"
        + "#   category.\r\n"
        + "# - If you want to check the format of this file before putting it\r\n"
        + "#   into WEPIF, paste it into https://yaml-online-parser.appspot.com/\r\n"
        + "#   and see if it gives \"ERROR:\".\r\n"
        + "# - Lines starting with # are comments and so they are ignored.\r\n"
        + "\r\n";
    private static final Logger LOGGER = LogManagerCompat.getLogger();  8 usages

    private static PermissionsResolverManager instance;  3 usages

    public static void initialize(Plugin plugin) {  1 usage  zml2008
        if (!isInitialized()) {
            instance = new PermissionsResolverManager(plugin);
        }
    }
```

Explanation

The `PermissionsResolverManager` class exhibits the "Large Class" code smell due to the following reasons:

1) High Number of Methods: The class contains numerous methods, including initialize, isInitialized, getInstance, findResolver, setPluginPermissionsResolver, and several overridden methods from the PermissionsResolver interface. This indicates that the class is handling multiple responsibilities.
2) Multiple Responsibilities: The class is responsible for managing permissions resolvers, loading configurations, and handling various permission-related operations. This violates the Single Responsibility Principle (SRP), making the class harder to maintain and understand.
3) Large Amount of Code: The class has a significant amount of code, including complex logic for loading configurations, finding resolvers, and handling exceptions. This increases the cognitive load on developers who need to understand or modify the class.
4) Complex Configuration Handling: The class includes detailed configuration handling logic, such as loading YAML files, setting properties, and managing enabled/disabled resolvers. This adds to the complexity and size of the class.

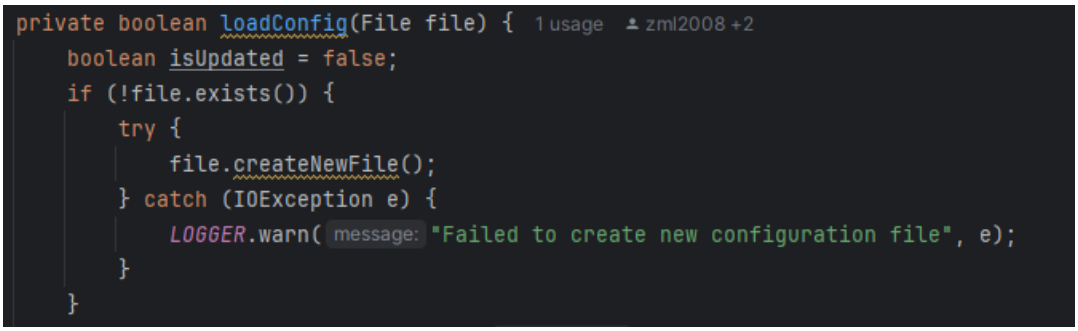
Proposal of a refactoring

Overall, the PermissionsResolverManager class is doing too much, it needs heavy refactoring to improve maintainability and readability. The class should be divided into other smaller classes that divide its responsibilities to make the implementation less convoluted. To address the "Large Class" code smell in the PermissionsResolverManager class, we can refactor it by dividing its responsibilities into smaller, more focused classes:

- Configuration Management: Extract the configuration-related methods into a separate class.
- Resolver Management: Extract the resolver-related methods into another class.
- Permissions Management: Extract the permissions-related methods into another class.

Long Method [identification: 65330; review: 62909]

Lines 188-271 of the file
\worldedit-bukkit\src\main\java\com\sk89q\wepif\
PermissionsResolverManager.java



Explanation

The loadConfig method is an example of the Long Method code smell because it:

- Contains multiple responsibilities, such as file creation, configuration loading, and property setting.
- Has a significant amount of code, making it harder to understand and maintain.
- Includes complex logic for handling configurations and updating properties, increasing cognitive load

Proposal of a refactoring

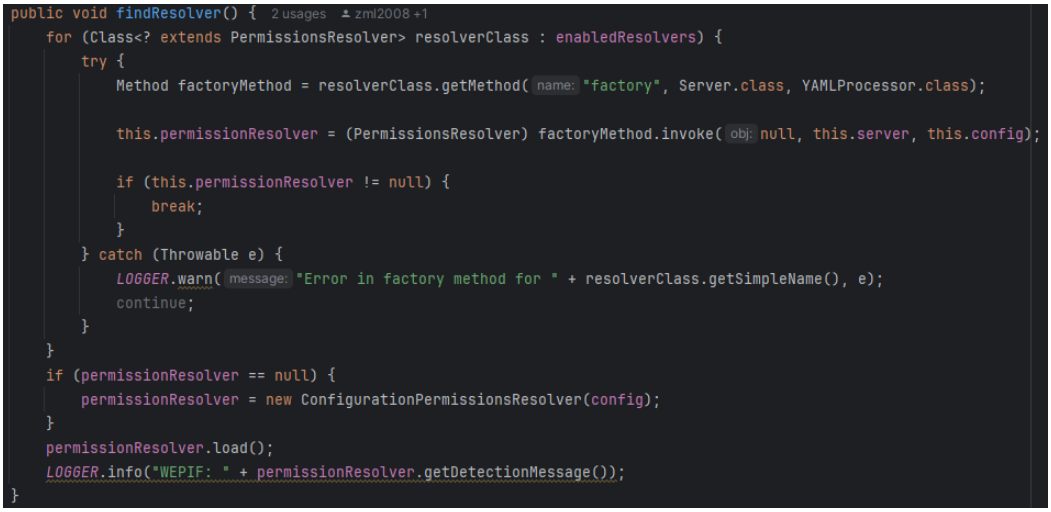
To address the Long Method code smell in the loadConfig method we should break it down to smaller methods:

- loadConfig: will use the new methods instead of the long method.
- createConfigFileIfNotExists: will create the configuration file if it does not exist.
- loadConfigFile: will load the configuration file using the YAMLProcessor.
- setDefaultProperties: will set default properties if they are missing in the configuration.
- updateResolvers: will update the resolvers based on the configuration.
- updateEnabledAndDisabledResolvers: will update the enabled and disabled resolvers based on the configuration.
- removeDeprecatedProperties: will remove deprecated properties from the configuration.
- generateDefaultPermissions: will generate default permissions if they are missing in the configuration.

Feature Envy [identification: 65330; review: 60593]

Lines 107-127 of the file \worldedit-bukkit\src\main\java\com\sk89q\wepif\PermissionsResolverManager.java

---



```
public void findResolver() {  2 usages   zml2008 +1
    for (Class<? extends PermissionsResolver> resolverClass : enabledResolvers) {
        try {
            Method factoryMethod = resolverClass.getMethod( name: "factory", Server.class, YAMLProcessor.class);

            this.permissionResolver = (PermissionsResolver) factoryMethod.invoke( obj: null, this.server, this.config);

            if (this.permissionResolver != null) {
                break;
            }
        } catch (Throwable e) {
            LOGGER.warn( message: "Error in factory method for " + resolverClass.getSimpleName(), e);
            continue;
        }
    }
    if (permissionResolver == null) {
        permissionResolver = new ConfigurationPermissionsResolver(config);
    }
    permissionResolver.load();
    LOGGER.info("WEPIF: " + permissionResolver.getDetectionMessage());
}
```

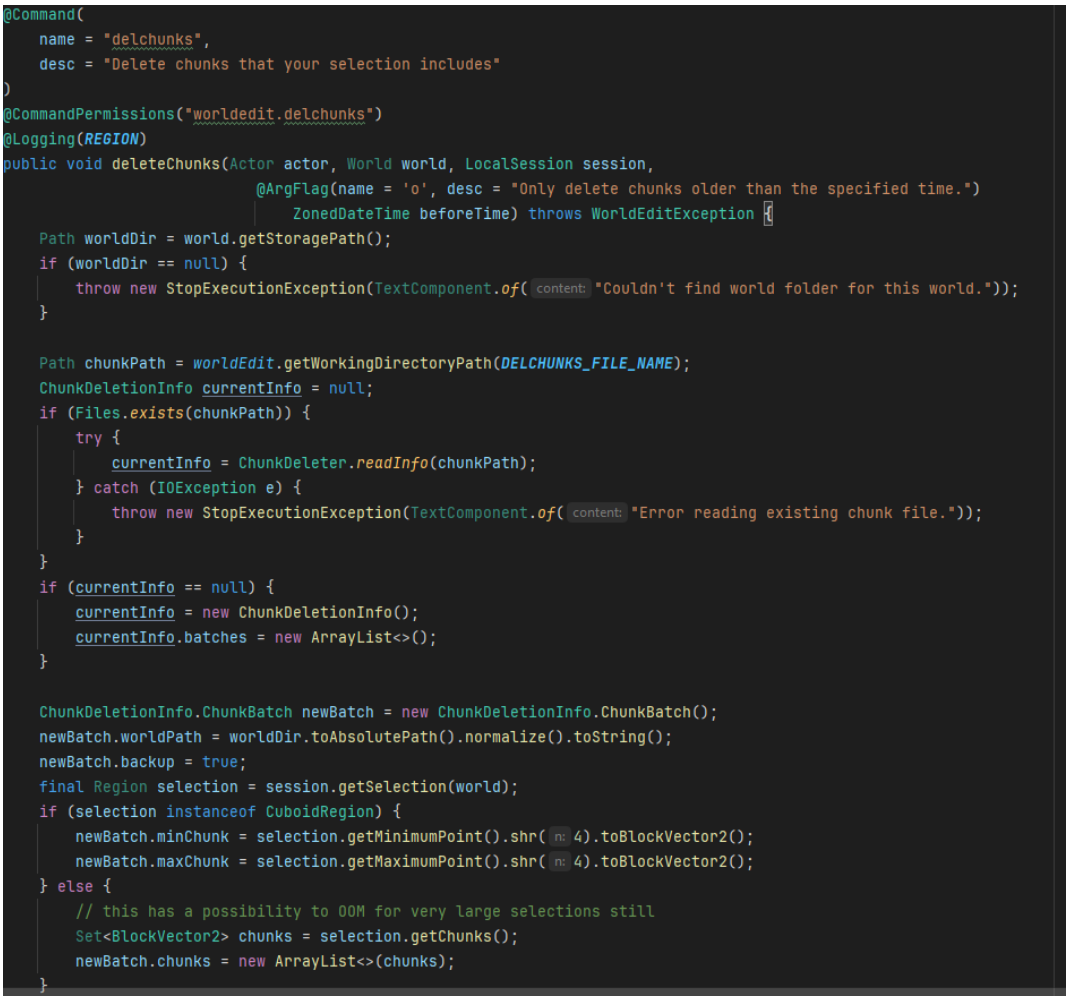
Explanation

The findResolver method exhibits the Feature Envy code smell because it seems more interested in other classes functionality rather than its own, this can be seen in the constant use of the PermissionsResolver class methods and factoryMethod, using basically none of functionalities of the class it resides in (PermissionsResolverManager).

Proposal of a refactoring

To address the Feature Envy code smell in the findResolver method, we should refactor it to delegate the responsibility of finding the resolver to a new class or method that is more closely related to PermissionsResolver.

Long Method [identification: 65969; review: 65330]
Lines 110-179 of the file worldedit-core/src/main/java/com/sk89q/worldedit/Command/ChunkCommands.Java



```
@Command(
    name = "delchunks",
    desc = "Delete chunks that your selection includes"
)
@CommandPermissions("worldedit.delchunks")
@Logging(REGION)
public void deleteChunks(Actor actor, World world, LocalSession session,
                         @ArgFlag(name = 'o', desc = "Only delete chunks older than the specified time.")
                             ZonedDateTime beforeTime) throws WorldEditException {
    Path worldDir = world.getStoragePath();
    if (worldDir == null) {
        throw new StopExecutionException(TextComponent.of( content: "Couldn't find world folder for this world."));
    }

    Path chunkPath = worldEdit.getWorkingDirectoryPath(DELCHUNKS_FILE_NAME);
    ChunkDeletionInfo currentInfo = null;
    if (Files.exists(chunkPath)) {
        try {
            currentInfo = ChunkDeleter.readInfo(chunkPath);
        } catch (IOException e) {
            throw new StopExecutionException(TextComponent.of( content: "Error reading existing chunk file."));
        }
    }
    if (currentInfo == null) {
        currentInfo = new ChunkDeletionInfo();
        currentInfo.batches = new ArrayList<>();
    }

    ChunkDeletionInfo.ChunkBatch newBatch = new ChunkDeletionInfo.ChunkBatch();
    newBatch.worldPath = worldDir.toAbsolutePath().normalize().toString();
    newBatch.backup = true;
    final Region selection = session.getSelection(world);
    if (selection instanceof CuboidRegion) {
        newBatch.minChunk = selection.getMinimumPoint().shr( n: 4).toBlockVector2();
        newBatch.maxChunk = selection.getMaximumPoint().shr( n: 4).toBlockVector2();
    } else {
        // this has a possibility to OOM for very large selections still
        Set<BlockVector2> chunks = selection.getChunks();
        newBatch.chunks = new ArrayList<>(chunks);
    }
```

```
if (beforeTime != null) {
    newBatch.deletionPredicates = new ArrayList<>();
    ChunkDeletionInfo.DeletionPredicate timePred = new ChunkDeletionInfo.DeletionPredicate();
    timePred.property = "modification";
    timePred.comparison = "<";
    timePred.value = String.valueOf((int) beforeTime.toOffsetDateTime().toEpochSecond());
    newBatch.deletionPredicates.add(timePred);
}
currentInfo.batches.add(newBatch);

try {
    ChunkDeleter.writeInfo(currentInfo, chunkPath);
} catch (IOException | JsonIOException e) {
    throw new StopExecutionException(TextComponent.of( content: "Failed to write chunk list: " + e.getMessage()));
}

actor.print(TextComponent.of(
    String.format("%d chunk(s) have been marked for deletion the next time the server starts.",
        newBatch.getChunkCount())
));
if (currentInfo.batches.size() > 1) {
    actor.printDebug(TextComponent.of(
        String.format("%d chunks total marked for deletion. (May have overlaps).",
            currentInfo.batches.stream().mapToInt(ChunkDeletionInfo.ChunkBatch::getChunkCount).sum())
    ));
}
actor.print(TextComponent.of( content: "You can mark more chunks for deletion, or to stop now, run: ", TextColor.LIGHT_PURPLE)
        .append(TextComponent.of( content: "/stop", TextColor.AQUA)
                .clickEvent(ClickEvent.of(ClickEvent.Action.SUGGEST_COMMAND,  value: "/stop"))));
```

Explanation

This function is a long method because it has a lot of lines of code, and it does a lot of things. It is responsible for deleting chunks that the user selects. It has a lot of responsibilities, such as reading and writing files, creating new objects, and printing messages to the user. This method could be refactored to be more readable and maintainable.

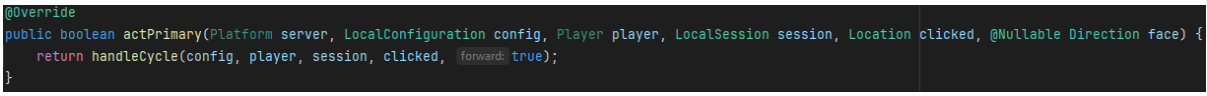
Proposal of a refactoring

- deleteChunks content could be broken down into functions:
    - readChunkInfo - to get the current chunk info, reading the file.
    - isBeforeTime - check if the beforeTime is not null. Doing all the logic that comes after that.
    - writeChunkInfo - to write the chunk info to the file.
    - printMessages - to print the messages to the user.

Long Parameter List [identification:65969; reviewer: 62909]

Lines 119-122 of the file worldedit-core/src/main/java/com/sk89q/worldedit/Command/tool/BlockDataCyler.Java

Code Snippet



Explanation

This method has a lot of parameter lists that can be encapsulated in an object. This increases the complexity of the code and makes it harder to maintain and understand.

Proposal of a refactoring

Create an object that encapsulates all the parameters and pass this object as a parameter to the method.

```
public class ActionContext {
    private final Platform server;
    private final LocalConfiguration config;
    private final Player player;
    private final LocalSession session;
    private final Location clicked;
    private final Direction face;

    public ActionContext(Platform server, LocalConfiguration config, Player player,
LocalSession session, Location clicked, @Nullable Direction face) {
        this.server = server;
        this.config = config;
        this.player = player;
        this.session = session;
        this.clicked = clicked;
        this.face = face;
    }

    public Platform getServer() {
        return server;
    }

    public LocalConfiguration getConfig() {
        return config;
    }

    public Player getPlayer() {
```

```
        return player;
    }

    public LocalSession getSession() {
        return session;
    }

    public Location getClicked() {
        return clicked;
    }

    public Direction getFace() {
        return face;
    }
}
```

And modify the method so it looks like this:

```
 @Override
   public boolean actPrimary(ActionContext actionContext) {
        return handleCycle(actionContext.getConfig(), actionContext.getPlayer(),
actionContext.getSession(), actionContext.getClicked(), true);
    }
```

Duplicate Code [identification:65969; reviewer: 66081]

Lines 110-179 of the file worldedit-core/src/main/java/com/sk89q/worldedit/Command/tool/BrushCommands.Java

Code Snippet

```
@Command(
    name = "cylinder",
    aliases = { "cyl", "c" },
    desc = "Choose the cylinder brush"
)
@CommandPermissions("worldedit.brush.cylinder")
public void cylinderBrush(Player player, LocalSession session,
                          @Arg(desc = "The pattern of blocks to set")
                              Pattern pattern,
                          @Arg(desc = "The radius of the cylinder", def = "2")
                              double radius,
                          @Arg(desc = "The height of the cylinder", def = "1")
                              int height,
                          @Switch(name = 'h', desc = "Create hollow cylinders instead")
                              boolean hollow) throws WorldEditException {
    worldEdit.checkMaxBrushRadius(radius);
    worldEdit.checkMaxBrushRadius(height);

    Brush brush = hollow ? new HollowCylinderBrush(height) : new
CylinderBrush(height);

    BrushTool tool = session.forceBrush(
```

```
        player.getItemInHand(HandSide.MAIN_HAND).getType(),
        brush,
        "worldedit.brush.cylinder"
    );
    tool.setFill(pattern);
    tool.setSize(radius);

    player.printInfo(TranslatableComponent.of("worldedit.brush.cylinder.equip",
TextComponent.of((int) radius), TextComponent.of(height)));
    ToolCommands.sendUnbindInstruction(player, UNBIND_COMMAND_COMPONENT);
}

@Command(
    name = "splatter",
    aliases = { "splat" },
    desc = "Choose the splatter brush"
)
@CommandPermissions("worldedit.brush.splatter")
public void splatterBrush(Player player, LocalSession session,
                    @Arg(desc = "The pattern of blocks to set")
                        Pattern pattern,
                    @Arg(desc = "The radius of the splatter", def = "2")
                        double radius,
                    @Arg(desc = "The decay of the splatter between 0 and 10", def =
"1")
                        int decay) throws WorldEditException {
    worldEdit.checkMaxBrushRadius(radius);

    if (decay < 0 || decay > 10) {
        player.printError(TranslatableComponent.of("worldedit.brush.splatter.decay-out-
of-range", TextComponent.of(decay)));
        return;
    }

    BrushTool tool = session.forceBrush(
        player.getItemInHand(HandSide.MAIN_HAND).getType(),
        new SplatterBrush(decay),
        "worldedit.brush.splatter"
    );
    tool.setFill(pattern);
    tool.setSize(radius);

    player.printInfo(TranslatableComponent.of("worldedit.brush.splatter.equip",
TextComponent.of((int) radius), TextComponent.of(decay)));
    ToolCommands.sendUnbindInstruction(player, UNBIND_COMMAND_COMPONENT);
}
```

Explanation

The last lines of the methods cylinderBrush, and splatterBrush are duplicated. (There are more methods that can be applied this refactoring)

```
tool.setFill(pattern);
tool.setSize(radius);

player.printInfo(TranslatableComponent.of("worldedit.brush.splatter.equip",
TextComponent.of((int) radius), TextComponent.of(decay)));
ToolCommands.sendUnbindInstruction(player, UNBIND_COMMAND_COMPONENT);
```

These four lines can be extracted to a new method and called in the end of each method.

Proposal of refactoring

Create a new method that receives the tool, pattern, radius, decay, and player as parameters and call this method at the end of each method.

```
private void setTool(BrushTool tool, Pattern pattern, double radius, int decay, Player
player) {
   tool.setFill(pattern);
   tool.setSize(radius);

   player.printInfo(TranslatableComponent.of("worldedit.brush.splatter.equip",
TextComponent.of((int) radius), TextComponent.of(decay)));
   ToolCommands.sendUnbindInstruction(player, UNBIND_COMMAND_COMPONENT);
}
```

Long Method [identification: 66081; reviewer: 65969]

Code snippet:

```
@Override
public Set<Entry<BlockVector3, V>> entrySet() {
   Set<Entry<BlockVector3, V>> es = entrySet;
   if (es == null) {
      entrySet = es = new AbstractSet<>() {
         @Override
         public Iterator<Entry<BlockVector3, V>> iterator() {
            return new Iterator<>() {

               private final
ObjectIterator<Long2ObjectMap.Entry<Int2ObjectMap<V>>> primaryIterator
                  = Long2ObjectMaps.fastIterator(maps);
               private Long2ObjectMap.Entry<Int2ObjectMap<V>>
currentPrimaryEntry;
               private ObjectIterator<Int2ObjectMap.Entry<V>>
secondaryIterator;
               private boolean finished;
               private LazyEntry next;

               @Override
               public boolean hasNext() {
                  if (finished) {
                     return false;
                  }
                  if (next == null) {
                     LazyEntry proposedNext = computeNext();
                     if (proposedNext == null) {
                        finished = true;
                        return false;
                     }
                     next = proposedNext;
                  }
                  return true;
               }

               private LazyEntry computeNext() {
                  if (secondaryIterator == null || !secondaryIterator.hasNext()) {
                     if (!primaryIterator.hasNext()) {
                        return null;
                     }
```

```
                    currentPrimaryEntry = primaryIterator.next();
                    secondaryIterator =
Int2ObjectMaps.fastIterator(currentPrimaryEntry.getValue());
                    // be paranoid
                    checkState(secondaryIterator.hasNext(),
                        "Should not have an empty map entry, it should have
been removed!");
                }
                Int2ObjectMap.Entry<V> next = secondaryIterator.next();
                return new LazyEntry(currentPrimaryEntry.getLongKey(),
next.getIntKey(), next.getValue());
            }

            @Override
            public Entry<BlockVector3, V> next() {
                if (!hasNext()) {
                    throw new NoSuchElementException();
                }
                LazyEntry tmp = next;
                next = null;
                return tmp;
            }

            @Override
            public void remove() {
                secondaryIterator.remove();
                // ensure invariants hold
                if (currentPrimaryEntry.getValue().isEmpty()) {
                    // the remove call cleared this map. call remove on the
primary iter
                    primaryIterator.remove();
                }
            }
        };
    }

    @Override
    public int size() {
        return BlockMap.this.size();
    }
  };
 }
 return es;
}
```

Location on the codebase:

- **Package:** com.sk89q.worldedit.util.collection
- **Class:** BlockMap

Explanation:
This method has a long method code smell because it has to many lines of code, due to the fact that it has a nested iterator class, most of the code in it is this iterator class. It has a total of 71 lines of code.

Proposal of a refactoring:
The method has an iterator class nested inside it, so we must make that nested class a concrete class. It should be something like this:

- EntryIterator.java:

```
public class EntryIterator implements Iterator<Entry<BlockVector3, V>> {
    private final
ObjectIterator<Long2ObjectMap.Entry<Int2ObjectMap<V>>>
primaryIterator = Long2ObjectMaps.fastIterator(maps);
    private Long2ObjectMap.Entry<Int2ObjectMap<V>>
currentPrimaryEntry;
    private ObjectIterator<Int2ObjectMap.Entry<V>> secondaryIterator;
    private boolean finished;
    private LazyEntry next;

    public EntryIterator

    @Override
    public boolean hasNext() {
        if (finished) {
            return false;
        }
        if (next == null) {
            LazyEntry proposedNext = computeNext();
            if (proposedNext == null) {
                finished = true;
                return false;
            }
            next = proposedNext;
        }
        return true;
    }

    private LazyEntry computeNext() {
```

```
        if (secondaryIterator == null || !secondaryIterator.hasNext()) {
            if (!primaryIterator.hasNext()) {
                return null;
            }

            currentPrimaryEntry = primaryIterator.next();
            secondaryIterator =
Int2ObjectMaps.fastIterator(currentPrimaryEntry.getValue());
            // be paranoid
            checkState(secondaryIterator.hasNext(),
                "Should not have an empty map entry, it should have been
removed!");
        }
        Int2ObjectMap.Entry<V> next = secondaryIterator.next();
        return new LazyEntry(currentPrimaryEntry.getLongKey(),
next.getIntKey(), next.getValue());
    }

    @Override
    public Entry<BlockVector3, V> next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }
        LazyEntry tmp = next;
        next = null;
        return tmp;
    }

    @Override
    public void remove() {
        secondaryIterator.remove();
        // ensure invariants hold
        if (currentPrimaryEntry.getValue().isEmpty()) {
            // the remove call cleared this map. call remove on the primary iter
            primaryIterator.remove();
        }
    }
 }
```

- BlockMap.entrySet:

```
 public Set<Entry<BlockVector3, V>> entrySet() {
    Set<Entry<BlockVector3, V>> es = entrySet;
    if (es == null) {
        entrySet = es = new AbstractSet<>() {
```

```
            @Override
            public Iterator<Entry<BlockVector3, V>> iterator() {
               return new EntryIterator();
            }

            @Override
            public int size() {
               return BlockMap.this.size();
            }
         };
      }
      return es;
}
```

God Class [identification: 66081; reviewer: 62111]

Code snippet:

```
public class LocalSession {

   private static final transient int CUI_VERSION_UNINITIALIZED = -1;
   public static transient int MAX_HISTORY_SIZE = 15;

   // total of 37 atributtes

   /**
    * Construct the object.
    *
    * <p>{@link #setConfiguration(LocalConfiguration)} should be called
    * later with configuration.</p>
    */
   public LocalSession() {
   }

   /**
    * Construct the object.
    *
    * @param config the configuration
    */
   public LocalSession(@Nullable LocalConfiguration config) {
      this.config = config;
   }

   /**
    * Set the configuration.
    *
    * @param config the configuration
    */
   public void setConfiguration(LocalConfiguration config) {
      checkNotNull(config);
      this.config = config;
   }

   /**
    * Called on post load of the session from persistent storage.
    */
   public void postLoad() {
      if (defaultSelector != null) {
         this.selector = defaultSelector.createSelector();
```

```
        }
    }

    // total of 94 methods
}
// total of 1036 lines of code
```

Location on the codebase:

- **Package:** com.sk89q.worldedit
- **Class:** LocalSession

Explanation:

This class is a god class because it has a large number of attributes (37) and methods (94), which makes it difficult to find some functionality in the class. It has a total of 1036 lines of code.

Proposal of a refactoring:

To refactor this class we should split it into smaller classes, each one with a single responsibility. For example, we could create a class for the configuration, another for the selector, and so on.

Feature Envy [identification: 66081; reviewer: 60593]

Code snippet:

```
public class CLIWorldEdit {

   //...

   public void onStarted() {
      setupPlatform();

      setupRegistries();
      WorldEdit.getInstance().loadMappings();

      config.load();

      WorldEdit.getInstance().getEventBus().post(new
PlatformReadyEvent(platform));
   }

   public void onStopped() {
      WorldEdit worldEdit = WorldEdit.getInstance();
      worldEdit.getSessionManager().unload();
      worldEdit.getPlatformManager().unregister(platform);
   }

   //...
}
```

Location on the codebase:

- **Package:** com.sk89q.worldedit.cli
- **Class:** CLIWorldEdit

Explanation:

In this class, the methods onStarted and onStopped are using a lot of methods from the WorldEdit class. This suggests that some of this logic should be moved to the WorldEdit class, as it seems to be more related to the WorldEdit class than to the CLIWorldEdit class.

Proposal of a refactoring:

Create methods onStarted and onStopped in the WorldEdit class to handle the logic that is currently in the CLIWorldEdit class.

Review:

- The code smell was well identified.
- With the explanation and the proposal for refactoring I can get an idea of what it is and how to solve it, although it is a bit vague.

Long Method [identification: 60593; reviewer:62909]

Code Snippet:

```
public int morph(BlockVector3 position, double brushSize, int minErodeFaces, int numErodeIterations, int minDilateFaces, int numDilateIterations)
    int ceilBrushSize = (int) Math.ceil(brushSize);
    int bufferSize = ceilBrushSize * 2 + 3;  // + 1 due to checking the adjacent blocks, plus the 0th block
    // Store block states in a 3d array so we can do multiple mutations then commit.
    // Two are required as for each iteration, one is "current" and the other is "new"
    BlockState[][][] currentBuffer = new BlockState[bufferSize][bufferSize][bufferSize];
    BlockState[][][] nextBuffer = new BlockState[bufferSize][bufferSize][bufferSize];

    // Simply used for swapping the two
    BlockState[][][] tmp;

    // Load into buffer
    for (int x = 0; x < bufferSize; x++) {
        for (int y = 0; y < bufferSize; y++) {
            for (int z = 0; z < bufferSize; z++) {
                BlockState blockState = getBlock(position.add( x: x - ceilBrushSize - 1,  y: y - ceilBrushSize - 1,  z: z - ceilBrushSize - 1));
                currentBuffer[x][y][z] = blockState;
                nextBuffer[x][y][z] = blockState;
            }
        }
    }

    double brushSizeSq = brushSize * brushSize;
    Map<BlockState, Integer> blockStateFrequency = new HashMap<>();
    int totalFaces;
    int highestFreq;
    BlockState highestState;
    for (int i = 0; i < numErodeIterations; i++) {
        for (int x = 0; x <= ceilBrushSize * 2; x++) {
            for (int y = 0; y <= ceilBrushSize * 2; y++) {
                for (int z = 0; z <= ceilBrushSize * 2; z++) {
                    int realX = x - ceilBrushSize;
                    int realY = y - ceilBrushSize;
                    int realZ = z - ceilBrushSize;
                    if (lengthSq(realX, realY, realZ) > brushSizeSq) {
                        continue;
                    }

                    // Copy across changes
                    nextBuffer[x + 1][y + 1][z + 1] = currentBuffer[x + 1][y + 1][z + 1];

                    BlockState blockState = currentBuffer[x + 1][y + 1][z + 1];

                    if (blockState.getBlockType().getMaterial().isLiquid() || blockState.getBlockType().getMaterial().isAir()) {
                        continue;
                    }

                    blockStateFrequency.clear();
```

...

Location on the codebase

Package: com.sk89q.worldedit

Class: *EditSession*

Method: *morph*

Line: 2816

Explanation

The morph method is long and handles many responsibilities: it loads the buffer, processes the erosion and dilation iterations, and then commits changes to the world. This violates the Single Responsibility Principle (SRP) and makes the code harder to understand and maintain.

This method has a cognitive complexity of 31 (very high), 126 lines of code (very high) and 32 cyclomatic complexity (very high).

Proposal of a refactoring

- A method to load the buffer.

- A method to perform erosion.

- A method to perform dilation.

- A method to commit changes.

Speculative Generality [identification: 60593; reviewer:62111]

```
public class FlatRegionMaskingFilter implements FlatRegionFunction {

    private final FlatRegionFunction function;
    private final Mask2D mask;

    /**
     * Create a new masking filter.
     *
     * @param mask the mask
     * @param function the delegate function to call
     */
    public FlatRegionMaskingFilter(Mask2D mask, FlatRegionFunction function) {
        checkNotNull(function);
        checkNotNull(mask);

        this.mask = mask;
        this.function = function;
    }

    @Override
    public boolean apply(BlockVector2 position) throws WorldEditException {
        return mask.test(position) && function.apply(position);
    }

}
```

Location on the codebase

Package: *com.sk89q.worldedit.function*

Explanation

This class is not used.

Proposal of a refactoring

- A simple delete is enough.

- @Deprecated is also an option.

Primitive Obsession [identification: 60593; reviewer:65330]

Code Snippet

```
public int drawLine(Pattern pattern, List<BlockVector3> vectors, double radius, boolean filled)  3 usages  orthoplex64
        throws MaxChangedBlocksException {

    Set<BlockVector3> vset = new HashSet<>();

    for (int i = 0; !vectors.isEmpty() && i < vectors.size() - 1; i++) {
        BlockVector3 pos1 = vectors.get(i);
        BlockVector3 pos2 = vectors.get(i + 1);

        int x1 = pos1.x();
        int y1 = pos1.y();
        int z1 = pos1.z();
        int x2 = pos2.x();
        int y2 = pos2.y();
        int z2 = pos2.z();
        int tipx = x1;
        int tipy = y1;
        int tipz = z1;
        int dx = Math.abs(x2 - x1);
        int dy = Math.abs(y2 - y1);
        int dz = Math.abs(z2 - z1);

        if (dx + dy + dz == 0) {
            vset.add(BlockVector3.at(tipx, tipy, tipz));
            continue;
        }

        int dMax = Math.max(Math.max(dx, dy), dz);
        if (dMax == dx) {
            for (int domstep = 0; domstep <= dx; domstep++) {
                tipx = x1 + domstep * (x2 - x1 > 0 ? 1 : -1);
                tipy = (int) Math.round(y1 + domstep * ((double) dy) / ((double) dx) * (y2 - y1 > 0 ? 1 : -1));
                tipz = (int) Math.round(z1 + domstep * ((double) dz) / ((double) dx) * (z2 - z1 > 0 ? 1 : -1));

                vset.add(BlockVector3.at(tipx, tipy, tipz));
            }
        } else if (dMax == dy) {
            for (int domstep = 0; domstep <= dy; domstep++) {
                tipy = y1 + domstep * (y2 - y1 > 0 ? 1 : -1);
                tipx = (int) Math.round(x1 + domstep * ((double) dx) / ((double) dy) * (x2 - x1 > 0 ? 1 : -1));
                tipz = (int) Math.round(z1 + domstep * ((double) dz) / ((double) dy) * (z2 - z1 > 0 ? 1 : -1));

                vset.add(BlockVector3.at(tipx, tipy, tipz));
            }
        } else /* if (dMax == dz) */ {
            for (int domstep = 0; domstep <= dz; domstep++) {
                tipz = z1 + domstep * (z2 - z1 > 0 ? 1 : -1);
                tipy = (int) Math.round(y1 + domstep * ((double) dy) / ((double) dz) * (y2 - y1 > 0 ? 1 : -1));
```

...

Location on the codebase

Package: *com.sk89q.worldedit*

Class: *EditSession*

Method: *drawLine*

Line: *2593*

Explanation

Primitive Obsession occurs when primitive data types (such as int, double, boolean, etc.) are used excessively to represent concepts that would be better modeled by objects or classes. In this case, all the `int` variables created at the beginning of the method.

Proposal of a refactoring

- Rather than working with individual coordinates like x1, y1, z1, consider using BlockVector3 objects for position manipulations. This will encapsulate the x, y, and z values and make the code more readable.

- Abstract the Axis Calculation, create a helper class or method that encapsulates the logic of determining the dominant axis.

- Encapsulate the Position Update Logic, the logic for updating the `tipx`, `tipy`, and `tipz` values could be moved into a method or class that encapsulates the logic of stepping through 3D space, reducing the complexity of the method.

Use Case Diagrams

[identification: 60593; review: 66081]

Super Pickaxe Commands

Use Case Name: Single Superpick

- **Use Case Description**: Instantly breaks blocks with a pickaxe when toggled on, with optional item drops, until toggled off.
- **Primary Actor**: Player
- **Secondary Actor**: None.

Use Case Name: Area Superpick

- **Use Case Description**: Instantly break all blocks of the same type within a specified radius with a pickaxe, with optional item drops, until toggled off.
- **Primary Actor**: Player
- **Secondary Actor**: None.

Use Case Name: Recursive Superpick

- **Use Case Description**: Instantly break all connected blocks of the same type within a specified radius with a pickaxe, with optional item drops, until toggled off.
- **Primary Actor**: Player
- **Secondary Actor**: None.

Brush Commands

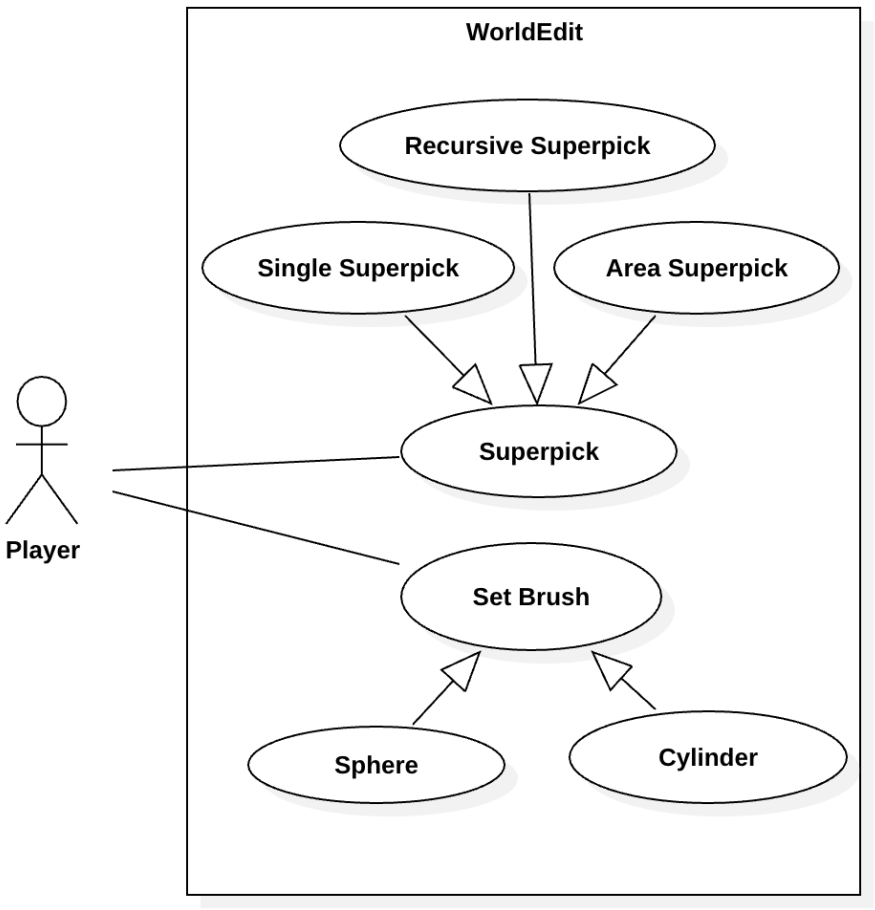
Use Case Name: Sphere Brush

- **Use Case Description**: Creates a sphere at the target point with a specified pattern and radius. The sphere can be hollow.
- **Primary Actor**: Player
- **Secondary Actor**: None.

Use Case Name: Cylinder Brush

- **Use Case Description**: Creates a cylinder at the target point with a specified pattern, radius, and height. The cylinder can be hollow.
- **Primary Actor**: Player
- **Secondary Actor**: None.

Use Case Diagram



[identification: 62111; review: 629909]

History Commands

Use Case Name: Undo Actions

- **Use Case Description**: The user undoes the last or a specified number of past actions from their or a specified player session history.
- **Primary Actor**: **User**
- **Secondary Actor**: **None.**

Use Case Name: Redo Actions

- **Use Case Description**: The user redoes the last or a specified number of undone actions from their or a specified player session history.
- **Primary Actor**: **User**
- **Secondary Actor**: **None.**

Use Case Name: Clear History

- **Use Case Description**: The user clears their action history, removing all previous actions from the session.
- **Primary Actor**: **User**
- **Secondary Actor**: **None.**

Snapshot Commands

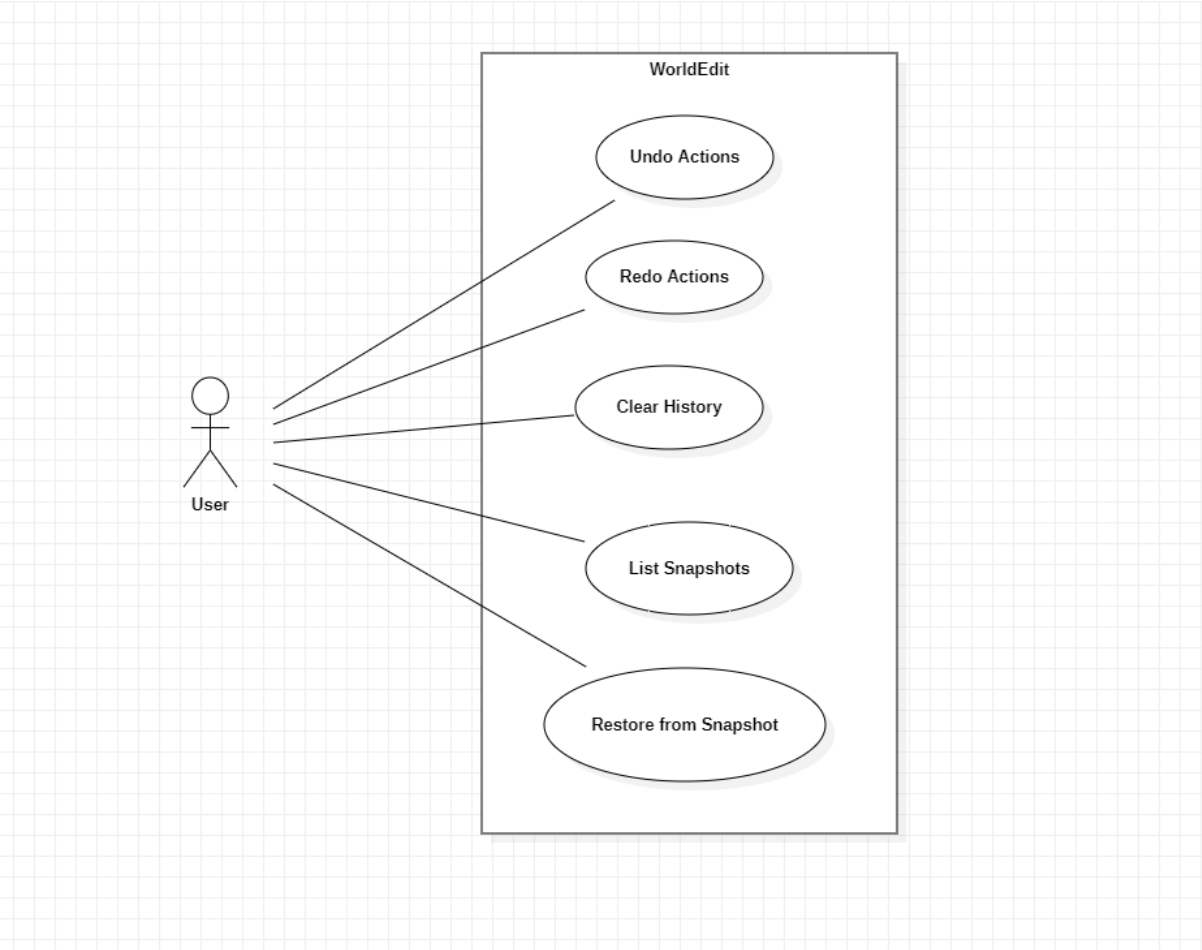
Use Case Name: List Snapshots

- **Use Case Description**: The user lists all snapshots available for their current world.
- **Primary Actor**: **User**
- **Secondary Actor**: **None.**

Use Case Name: Restore from Snapshot

- **Use Case Description**: The user restores a selected area from the most recent backup for their current world.
- **Primary Actor**: **User**
- **Secondary Actor**: **None.**

Use Case Diagram



[identification: 62909; review: 65969]

Region Commands

Use Case Name: Set Pattern

- **Use Case Description**: The user changes an area of blocks in a selection to specific patterns.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Naturalize Region

- **Use Case Description**: The user naturalizes terrain by creating a layer of grass, followed by layers of dirt and then stone.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Set Biome

- **Use Case Description**: The user sets the biome in their selection.
- **Primary Actor**: User
- **Secondary Actor**: None.

Generation Commands

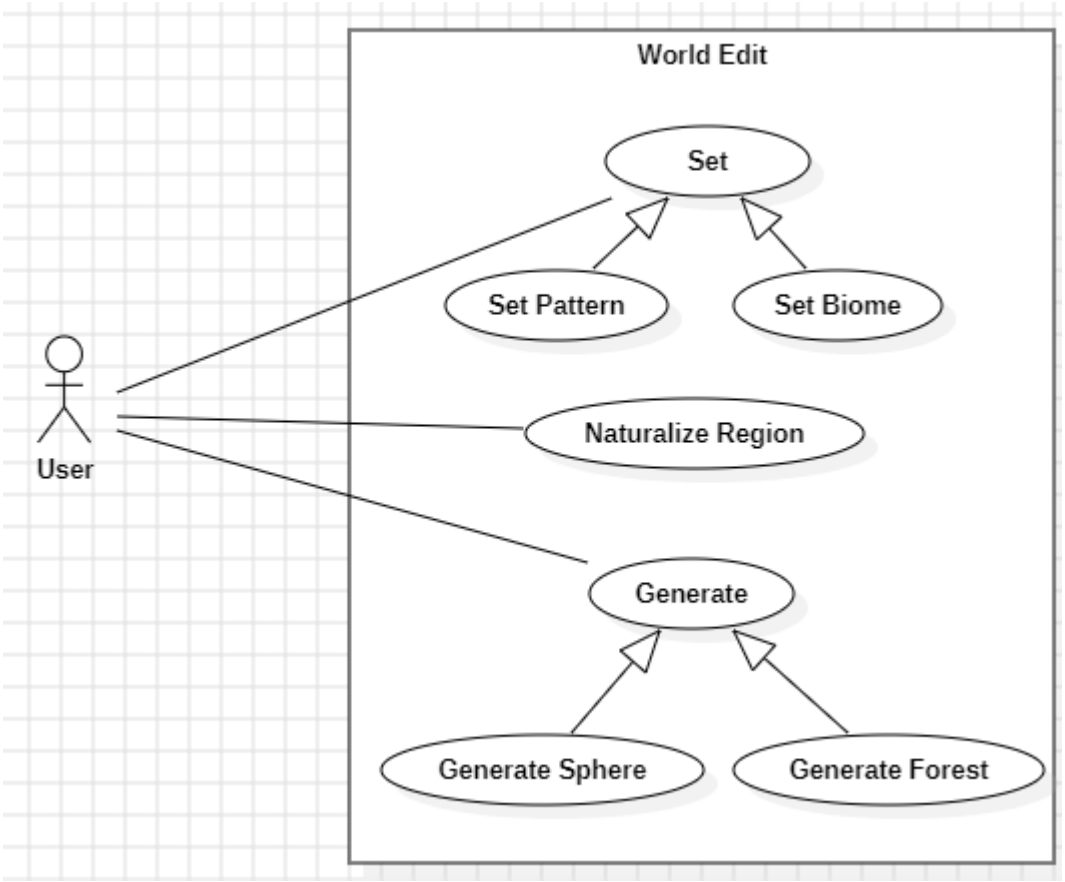
Use Case Name: Generate Sphere

- **Use Case Description**: The user generates a sphere at a given position with a given size.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Generate Forest

- **Use Case Description**: The user generates a forest with a given size, type and density.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Diagram



[identification: 65330; review: 60593]

Utility Commands

Use Case Name: Fill

- **Use Case Description**: Fills holes within a given area, to a specified depth (fills holes with a specified block).
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Drain

- **Use Case Description**: Drains water within a certain radius (removes water blocks within the radius).
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Snow

- **Use Case Description**: Simulates snow (puts snow particles in a determined cylinder).
- **Primary Actor**: User
- **Secondary Actor**: None.

Schematic and Clipboard Commands

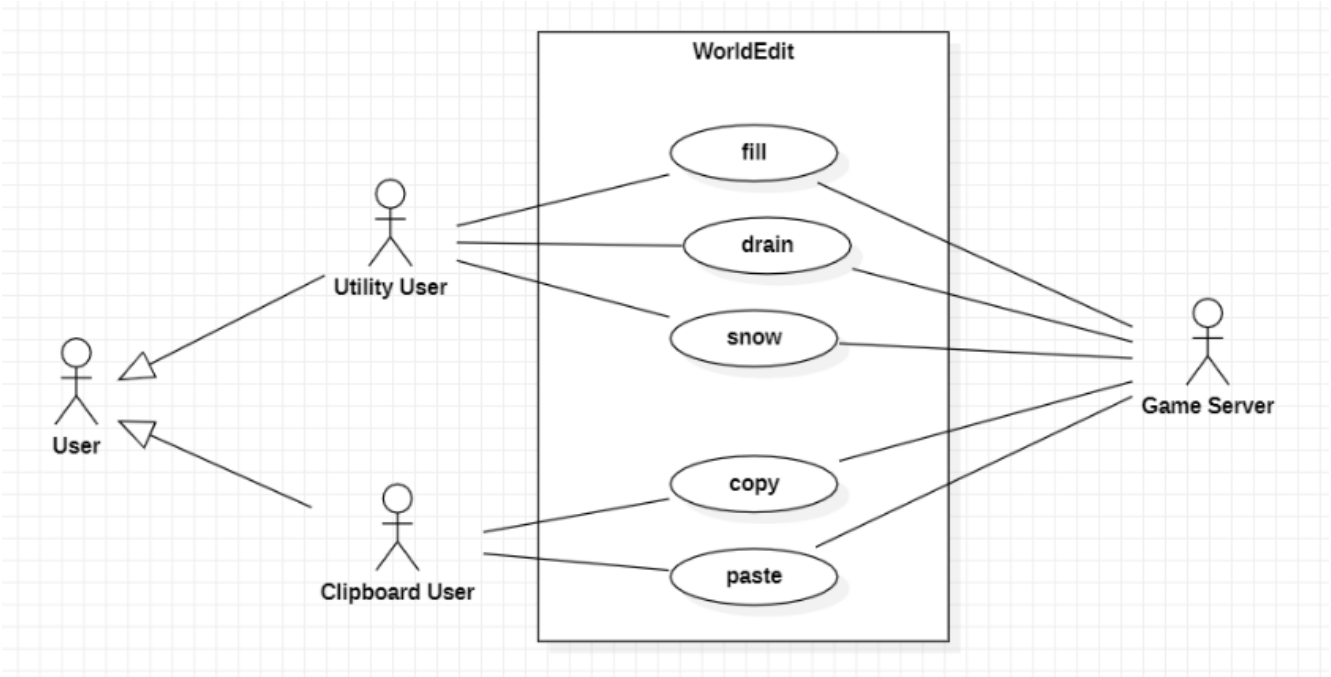
Use Case Name: Copy

- **Use Case Description**: Copy the world selection to the clipboard.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Paste

- **Use Case Description**: Paste the clipboard’s contents into the world.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Diagram



[identification: 65969; review: 62111]

Navigation Commands

Use Case Name: Unstuck

- **Use Case Description**: Escape from being stuck inside a block.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Ascend

- **Use Case Description**: Go up a floor.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Descend

- **Use Case Description**: Go down a floor.
- **Primary Actor**: User
- **Secondary Actor**: None.

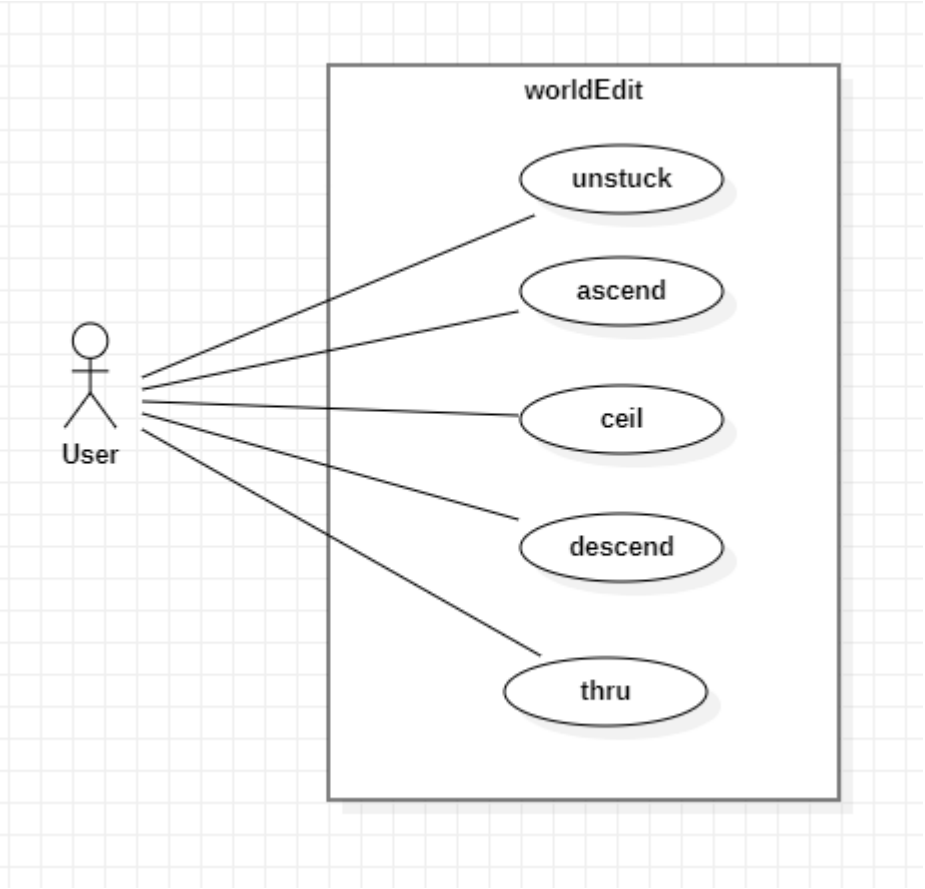
Use Case Name: Ceil

- **Use Case Description**: Go to the ceiling.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Thru

- **Use Case Description**: Pass through walls.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Diagram



[identification: 66081; review: 65330]

Tool Commands

Use Case Name: Bind deltree tool

- **Use Case Description**: The user binds the floating tree remover tool to the item in their hand.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Bind stacker tool

- **Use Case Description**: The user binds the stacker tool to the item in their hand.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Unbind tool

- **Use Case Description**: The user unbinds a bound tool from their current item.
- **Primary Actor**: User
- **Secondary Actor**: None.

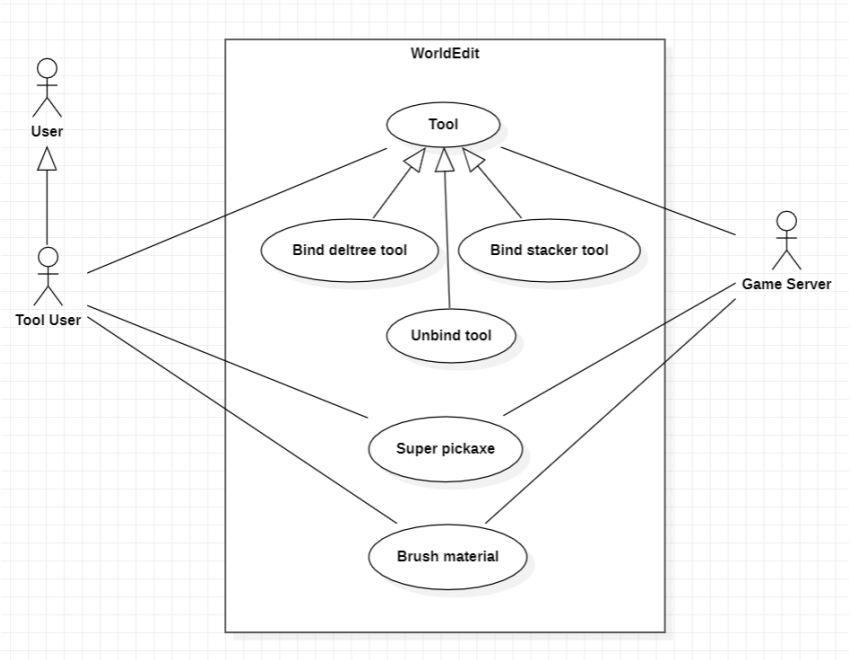
Use Case Name: Super pickaxe

- **Use Case Description**: The user toggles the super pickaxe function on or off.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Name: Brush material

- **Use Case Description**: The user sets the brush material.
- **Primary Actor**: User
- **Secondary Actor**: None.

Use Case Diagram



MILESTON 3

[identification: 62111 and 66081; review: 60593 and 62909]

Updated User Story:

As an animal lover, **I want** a tool that allows me to spawn and place specific animals of different species, babies or not, in a selected area, **so that** I can create themed environments or ecosystems with precise control over animal placement and population density.

Use Cases:

Use Case: Summon animals
ID: 2
Brief Description: Summons one or more animals inside a specified region.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. An area is selected.
Main flow: 1. The use case starts when the user types //animal in the game chat. 1.1. The system sugests a list of available animal types for the user to select. 2. The user types or selects an animal type. 3. If nothing else is specified 3.1. The system creates one adult animal of the specified type with the default variant. 4. If an integer number is specified 4.1. The system creates the specified number of animals. 5. If the -b flag is used 5.1. The system creates the animal(s) in baby version. 6. If a variant is specified 6.1. The system creates the animal(s) in the specified variant.

Use Case: Summon animals
Postconditions: 1. The system creates one or more animals in a random position of the selected area.
Alternative Flows: InvalidAnimalType InvalidArguments InvalidCount InvalidVariant

Alternative Flow: Summon animals: InvalidAnimalType
ID: 2.1
Brief Description: The system informs the user that the chosen animal type is invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an invalid animal type.
Alternative flow: 1. The alternative flow begins after step 2 of the main flow. 2. The system informs the user that the chosen animal type is invalid.
Postconditions: None.

Alternative Flow: Summon animals: InvalidArguments
ID: 2.2
Brief Description: The system informs the user that the entered number of arguments is invalid and shows the right usage of the command.

Alternative Flow: Summon animals: InvalidArguments
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an invalid number of arguments or enters them in the wrong order.
Alternative flow: 1. The alternative flow begins after step 1.1 of the main flow. 2. The system informs the user that the entered number of arguments is invalid and shows the right usage of the command.
Postconditions: None.

Alternative Flow: Summon animals: InvalidCount
ID: 2.3
Brief Description: The system informs the user that the count must be a positive number.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an integer number smaller than 1.
Alternative flow: 1. The alternative flow begins after step 4 of the main flow. 2. The system informs the user that the count must be a positive number.
Postconditions: None.

Alternative Flow: Summon animals: InvalidVariant
ID: 2.4
Brief Description: The system informs the user that the chosen animal variant is invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an invalid animal variant.
Alternative flow: 1. The alternative flow begins after step 6 of the main flow. 2. The system informs the user that the chosen animal variant is invalid.
Postconditions: None.

Use Case: List variants
ID: 3
Brief Description: List the variants of a specified animal type.
Primary Actors: User
Secondary Actors: None.
Preconditions: None.
Main flow: 1. The use case starts when the user types //variants in the game chat. 1.1. The system sugests a list of available animals for the user to select. 2. The user types or selects an animal type. 3. If nothing else is specified 3.1. The system lists, in the game chat, the available variants of the specified animal.

Use Case: List variants
4. If the -p flag is used followed by a page number 4.1. The system shows the list begining from the specified page number.
Postconditions: 1. The system shows a list of animal variants in the game chat.
Alternative Flows: InvalidAnimalType InvalidArguments InvalidPageNumber AnimalWithoutVariants

Alternative Flow: List variants: InvalidAnimalType
ID: 3.1
Brief Description: The system informs the user that the chosen animal type is invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an invalid animal type.
Alternative flow: 1. The alternative flow begins after step 2 of the main flow. 2. The system informs the user that the chosen animal type is invalid.
Postconditions: None.

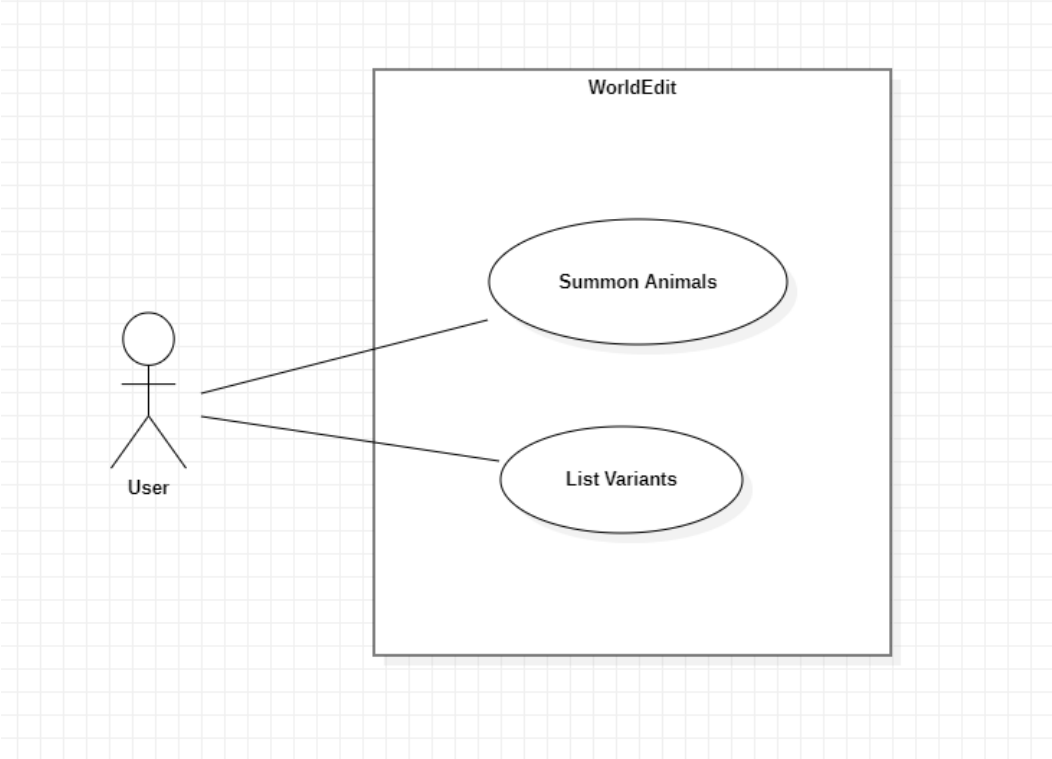
Alternative Flow: List variants: InvalidArguments
ID: 3.2
Brief Description: The system informs the user that the entered number of arguments is invalid and shows the right usage of the command.

Alternative Flow: List variants: InvalidArguments
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an invalid number of arguments or enters them in the wrong order.
Alternative flow: 1. The alternative flow begins after step 1.1 of the main flow. 2. The system informs the user that the entered number of arguments is invalid and shows the right usage of the command.
Postconditions: None.

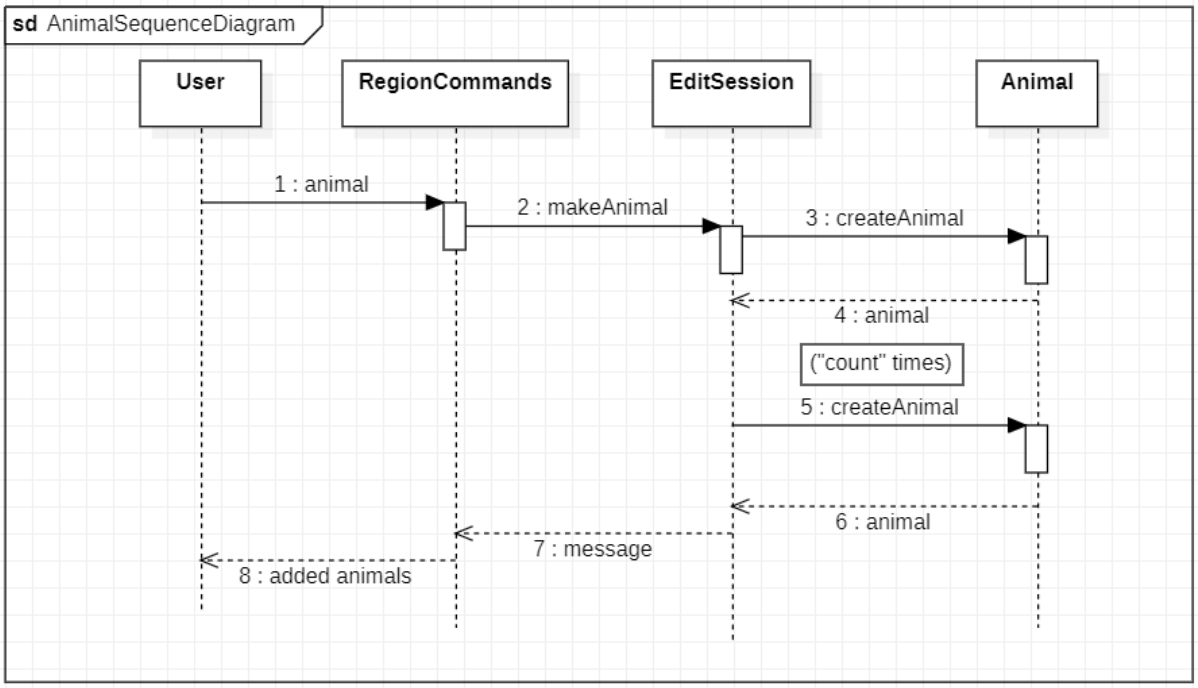
Alternative Flow: List variants: InvalidPageNumber
ID: 3.3
Brief Description: The system informs the user that the page number is invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an invalid page number.
Alternative flow: 1. The alternative flow begins after step 4 of the main flow. 2. The system informs the user that the page number is invalid.
Postconditions: None.

Alternative Flow: List variants: AnimalWithoutVariants
ID: 3.4
Brief Description: The system informs the user that the chosen animal type doesn't have variants.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user enters an animal type that don't have variants.
Alternative flow: 1. The alternative flow begins after step 3 of the main flow. 2. The system informs the user that the chosen animal type doesn't have variants.
Postconditions: None.

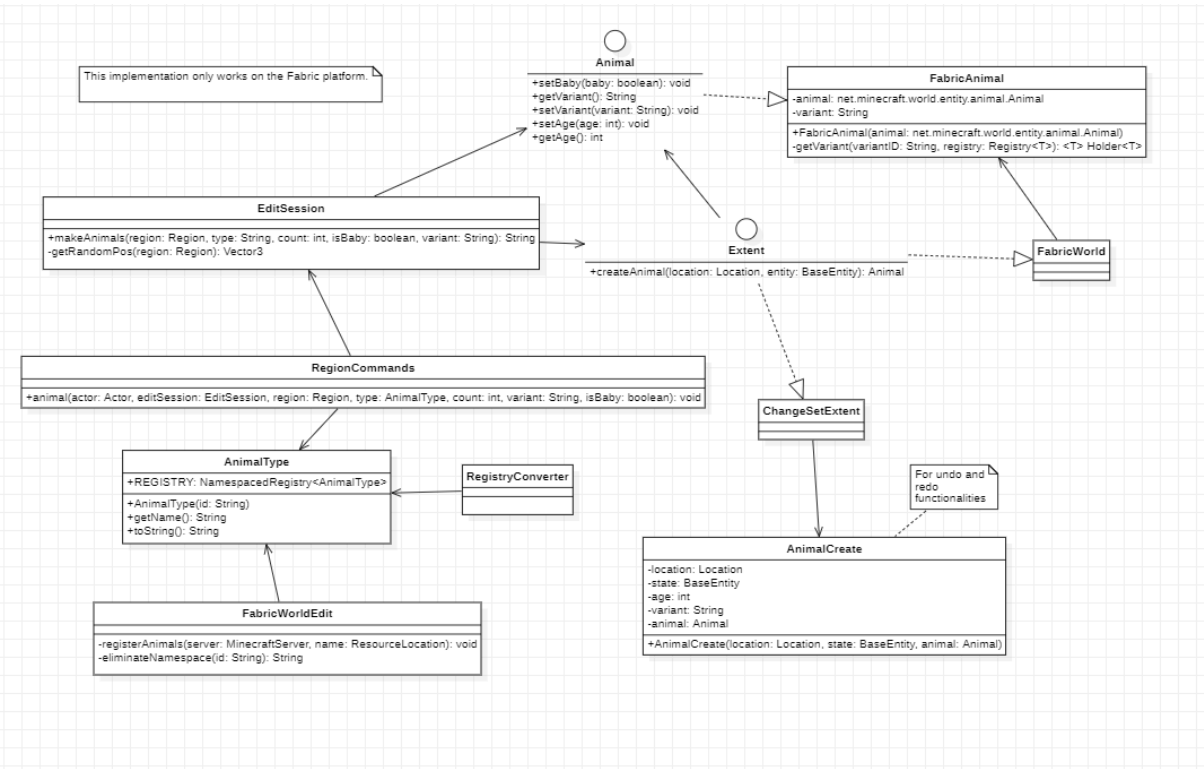
Use Case Diagram:



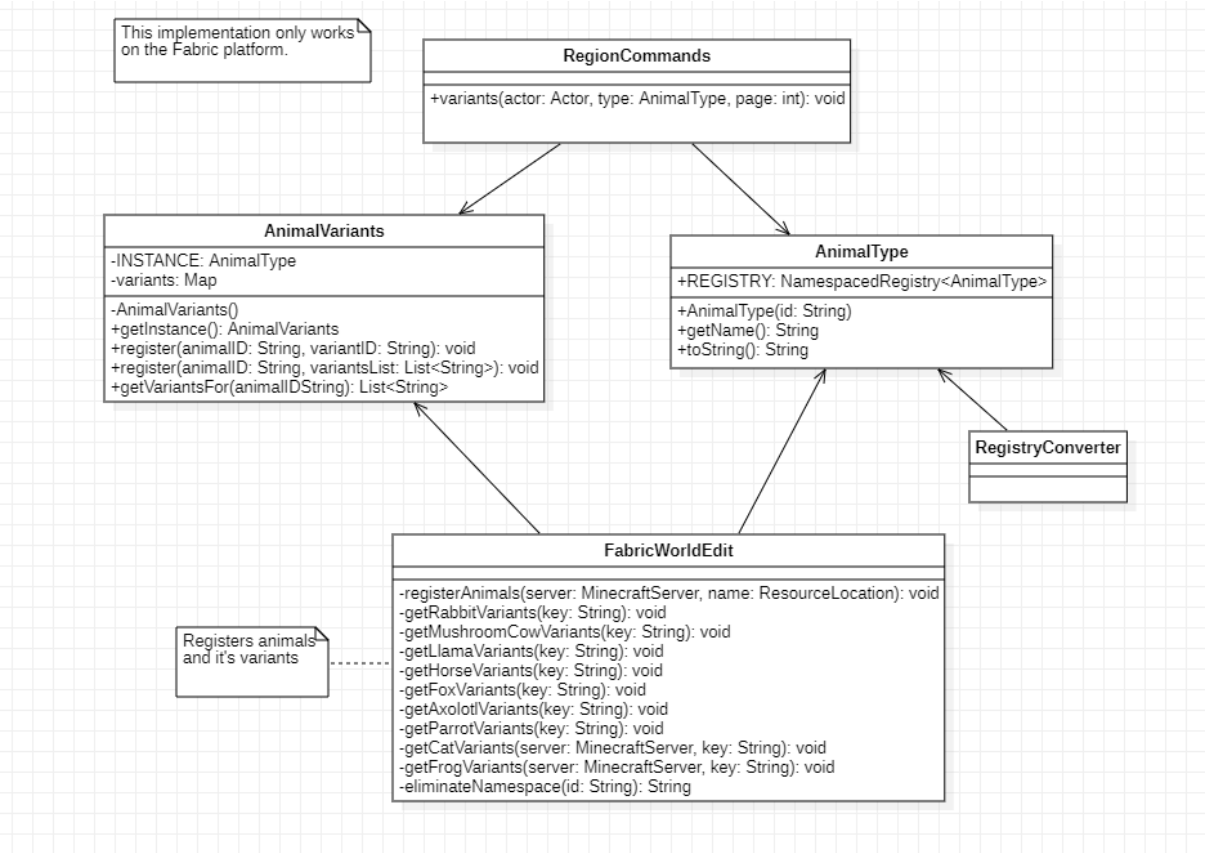
Sequence Diagram:



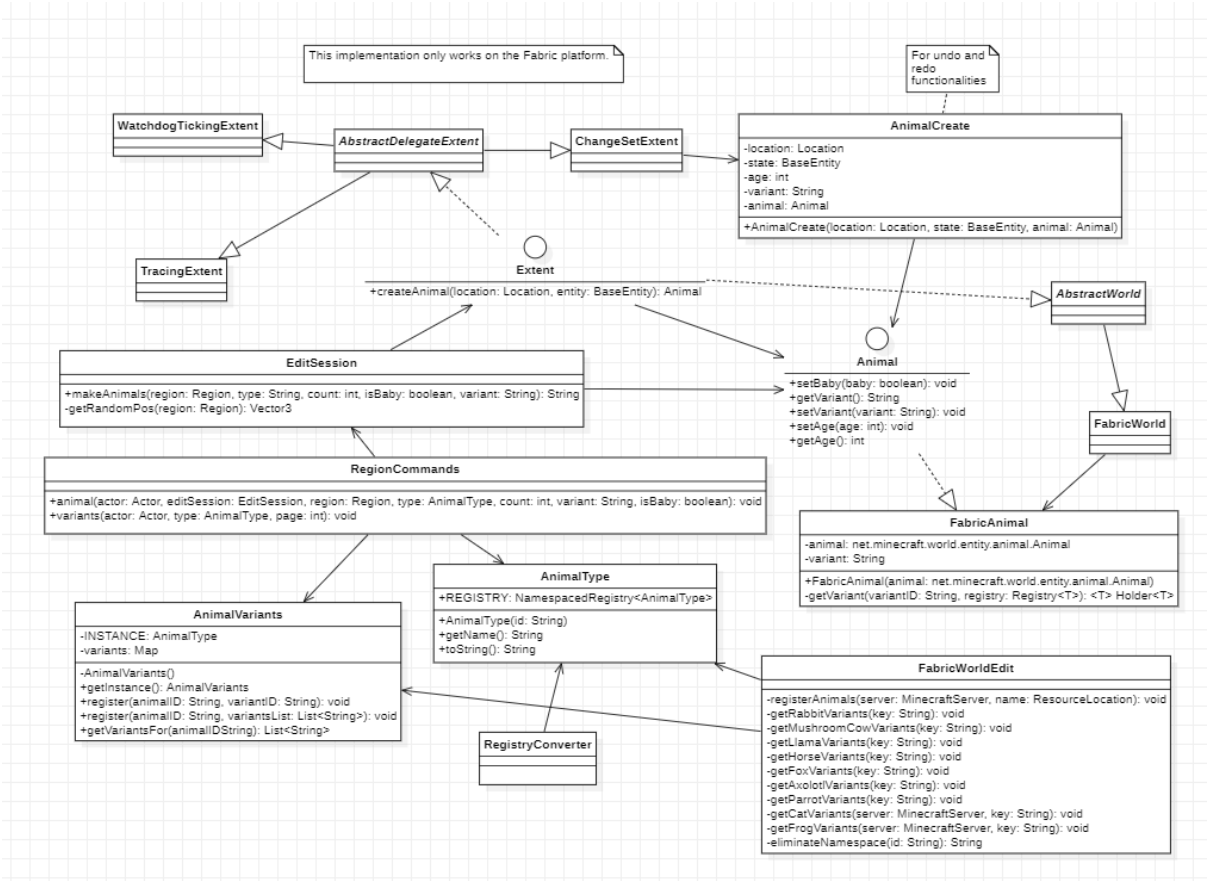
Animal Class Diagram (Simplified):



Variants Class Diagram:



Complete Class Diagram:



Tests:

Pre-Condition	Test case id	Steps	Expected Results
None.	Add one animal	1. The user selects the first position (-201, 62, -17). 2. The user selects the second position (-201, 62, -17). 3. The user enters //animal [animal-name] on the chat, where "animal-name" is the name of an animal in the game, for example "cow".	1. The system prints the message "First position set to (-201, 62, -17)." 2. The system prints the message "Second position set to (-201, 62, -17) (100).". 3. The system adds 1 animal and prints "An animal was created".
None.	Add multiple animals, one per block	1. The user selects the first position (-201, 62, -17). 2. The user selects the	1. The system prints the message "First position set to (-201, 62, -17)."

Pre-Condition	Test case id	Steps	Expected Results
		second position (-210, 62, -26). 3. The user enters //animal [animal-name] 100 on the chat, where "animal-name" is the name of an animal in the game, for example "chicken".	2. The system prints the message "Second position set to (-210, 62, -26) (100).". 3. The system adds 100 animals and prints "100 animals were created".
None.	Tests all animals	1. The user selects the first position (-201, 62, -17). 2. The user selects the second position (-210, 62, -26). 3. The user enters //animal [animal-name] on the chat, where "animal-name" is the name of an animal, the experience must be repeated to every animal.	1. The system prints the message "First position set to (-201, 62, -17)." 2. The system prints the message "Second position set to (-210, 62, -26) (100).". 3. The system adds one animal per experience and prints "An animal was created" each time.
None.	Limit of animals per block at once (with chickens)	1. The user selects the first position (-206, 62, -21). 2. The user selects the second position, which is the same of the first (-206, 62, -21). 3. The user enters //animal chicken 24 on the chat. 4. The user enters /undo to remove the 24 animals 5. The user enters //animal chicken 25 on the chat.	1. The system prints the message "First position set to (-206, 62, -21)." 2. The system prints the message "Second position set to (-206, 62, -21) (1).". 3. The system adds 24 chickens in one block and prints "100 animals were created". 4. The system removes the 24 chickens and prints "Undid 1 available edit" The system adds 25 chickens in one block and prints "25 animals were

Pre-Condition	Test case id	Steps	Expected Results
			created" but one of them will instantly dye, because the limit of identities per block defined by the minecraft is 24.
None.	Add baby animals	1. The user selects the first position (-201, 62, -17). 2. The user selects the second position (-210, 62, -26). 3. The user enters //animal [animal-name] -b on the chat, where "animal-name" is the name of an animal in the game, for example "chicken".	1. The system prints the message "First position set to (-201, 62, -17)." 2. The system prints the message "Second position set to (-210, 62, -26) (100).". 3. The system adds 1 animal in the baby form and prints "An animal was created".
None.	Add other variants/species of an animal	1. The user selects the first position (-201, 62, -17). 2. The user selects the second position (-210, 62, -26). 3. The user enters //animal [animal-name] [variant] on the chat, where "animal-name" is the name of an animal with more than one variant in the game and "variant" is that variant, for example "fox" and "snow".	1. The system prints the message "First position set to (-201, 62, -17)." 2. The system prints the message "Second position set to (-210, 62, -26) (100).". 3. The system adds 1 animal in the corresponding variant and prints "An animal was created".
None.	Mix babies with other variants of an animal	1. The user selects the first position (-201, 62, -17). 2. The user selects the second position (-210, 62, -26). 3. The user enters //animal	1. The system prints the message "First position set to (-201, 62, -17)." 2. The system prints the message "Second position set to (-210, 62, -26)

Pre-Condition	Test case id	Steps	Expected Results
		[animal-name] [variant] -b on the chat, where "animal-name" is the name of an animal with more than one variant in the game and "variant" is that variant, for example "fox" and "snow".	(100).". 3. The system adds 1 animal in the corresponding variant and baby form and prints "An animal was created".
None.	Try adding an animal that doesn't exist	1. The user selects the first position (-201, 62, -17). 2. The user selects the second position (-201, 62, -17). 3. The user enters //animal [animal-name] on the chat, where "animal-name" is an animal that doesn't exist in the game, for example "dog".	1. The system prints the message "First position set to (-201, 62, -17)." 2. The system prints the message "Second position set to (-201, 62, -17) (100).". 3. The system prints the corresponding error message.
None.	Try adding an animal with a variant that doesn't exists	1. The user selects the first position (-201, 62, -17). 2. The user selects the second position (-201, 62, -17). 3. The user enters //animal [animal-name] [variant] on the chat, where "animal-name" is the name of an animal in the game and "variant" is a variant that doesn't exist, for example "fox" and "fire".	1. The system prints the message "First position set to (-201, 62, -17)." 2. The system prints the message "Second position set to (-201, 62, -17) (100).". 3. The system prints error message "Invalid variant".

Implemented Code:

Lines 128 and 160 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/command/RegionCommands.java

New Methods:
- animal
- variants

Lines 842 and 160 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java

New Methods:
- makeAnimal
- getRandomPos

All lines of the file
worldedit-core/src/main/java/com/sk89q/worldedit/world/animal/Animal.java

New Interface: Animal

All lines of the file
worldedit-fabric/src/main/java/com/sk89q/worldedit/fabric/animal/FabricAnimal.java

New Class: FabricAnimal

All lines of the file
worldedit-core/src/main/java/com/sk89q/worldedit/world/animal/AnimalVariants.java

New Class: AnimalVariants

All lines of the file
worldedit-core/src/main/java/com/sk89q/worldedit/world/animal/AnimalType.java

New record: AnimalType

Line 65 of the file
worldedit-
core/src/main/java/com/sk89q/worldedit/command/argument/RegistryConverter.java

Added a line so the AnimalType record could be registered.

Lines 247 and from 307 to 443 of the file
worldedit-fabric/src/main/java/com/sk89q/worldedit/fabric/FabricWorldEdit.java

New Methods:

- registerAnimals
- getParrotVariants
- getRabbitVariants
- getMushroomCowVariants
- getLlamaVariants
- getHorseVariants
- getFoxVariants
- getAxolotlVariants
- getCatVariants
- getFrogVariants
- eliminateNamespace

Line 93 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/extent/Extent.java

New Methods:

- createAnimal

Line 90 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/extent/AbstractDelegateExtent.java

Implemented createAnimal

Line 112 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/extent/TracingExtent.java

Implemented createAnimal

Line 96 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/extent/world/WatchdogTickingExtent.java

Implemented createAnimal

Line 122 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/extent/ChangeSetExtent.java

Implemented createAnimal

Line 722 of the file

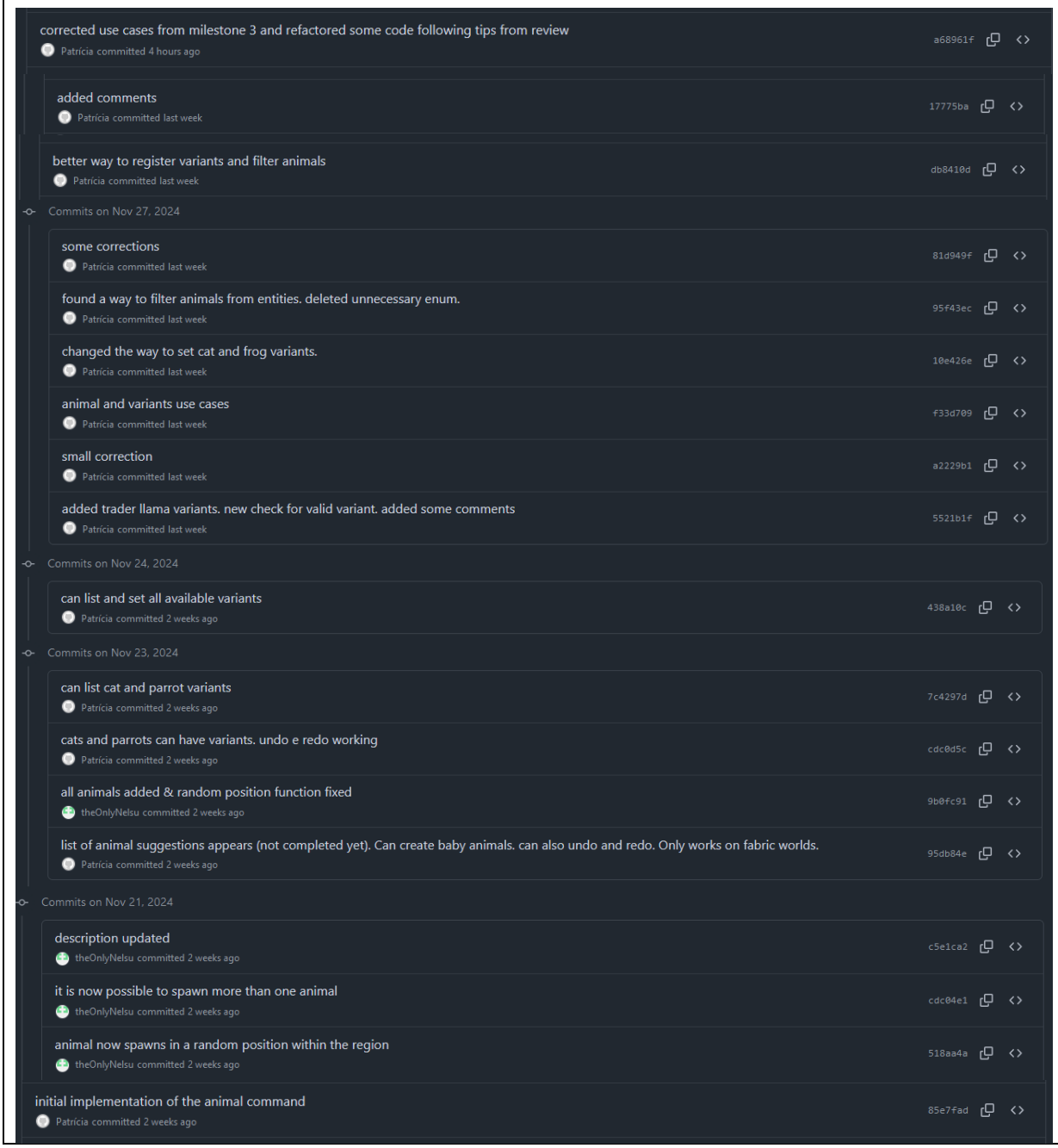
worldedit-fabric/src/main/java/com/sk89q/worldedit/fabric/FabricWorld.java

Implemented createAnimal

All lines of the file
worldedit-core/src/main/java/com/sk89q/worldedit/history/change/AnimalCreate.java

New Class: AnimalCreate

Commits:



[identification: 66330 and 65969; review: 62111 and 66081]

Updated User Story:

As a builder, I want to have a tool that allows me to raise blocks to a certain height, in a given area, so that I can build walls and other structures more efficiently.

Use Cases:

Use Case: Raise a certain blocks in a region by a specific height
ID: 1
Brief Description: The system raises all blocks of the chosen type by the chosen height (must contain blocks of the chosen type for any action to take effect).
Primary Actors: User
Secondary Actors: None.
Preconditions: None.
Main flow: 1. The user selects two positions in the world (the first with a left click and the second with a right click). 2. The user types the //raise command with the desired height to raise including the block type that should be raised. 3. The system checks for the blocks we want to raise in the region. 4. The system places height number of blocks above the blocks of the type selected.
Post conditions: 1. The chosen blocks have height number of blocks above them.
Alternative Flows: InvalidBlock InvalidRegion InvalidHeight InvalidArguments

Alternative Flow: InvalidBlock
ID: 1.1
Brief Description: The system informs the user that the block type is invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user specifies an unrecognized block type to the //raise command.
Alternative flow: 1. The alternative flow begins after step 2 of the main flow. 2. The user selects an invalid block type. 3. The system detects the invalid block type and displays an error message. Post conditions: None.
Post conditions: None.

Alternative Flow: InvalidRegion
ID: 1.2
Brief Description: The system informs the user that the region chosen is invalid.
Primary Actors: User

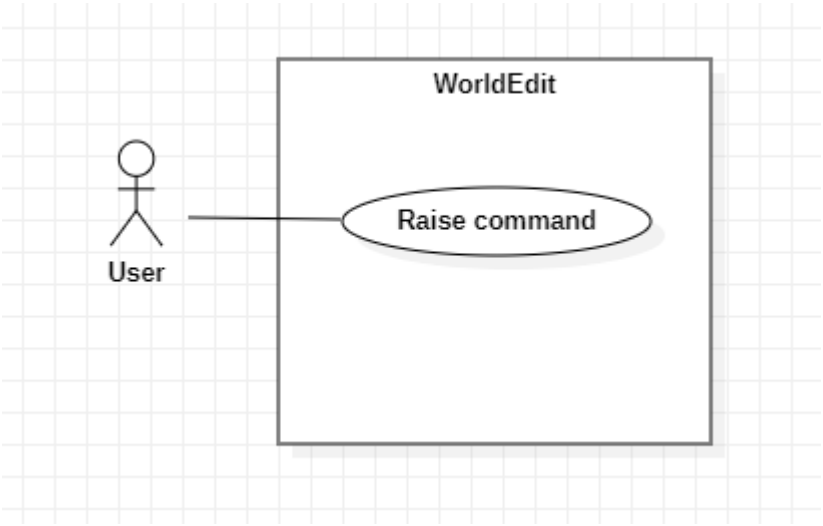
Secondary Actors: None.
Preconditions: 1. The user selects an invalid number of positions (0 or 1) for the region.
Alternative flow: 1. The user selects 0 or 1 positions. 2. The user types the //raise command with the desired height to raise including the block type that should be raised. 3. The system informs the user that the region chosen is invalid.
Post conditions: None.

Alternative Flow: InvalidHeight
ID: 1.3
Brief Description: The system informs the user that the height chosen is invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user gives a zero or negative height to the //raise command.
Alternative flow: 1. The alternative flow begins after step 2 of the main flow. 2. The user gives a zero or negative height to the //raise command. 3. The system detects the invalid height and displays an error message.
Post conditions: None.

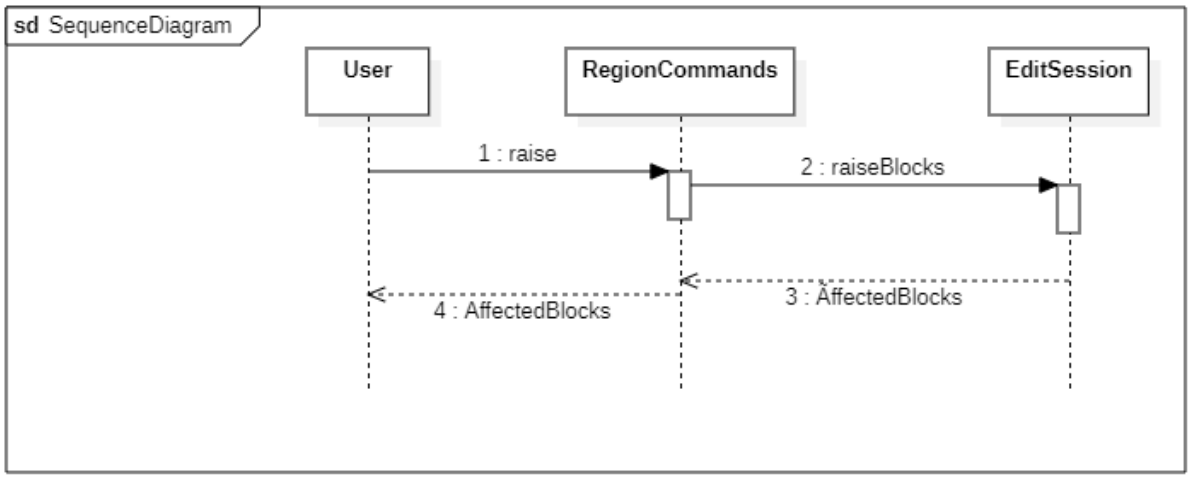
Alternative Flow: InvalidArgument
ID: 1.4
Brief Description: The system informs the user that the arguments are invalid.
Primary Actors: User

Secondary Actors: None.
Preconditions: 1. The user gives too many or too few arguments or the arguments are in the wrong order to the //raise command.
Alternative flow: 1. The alternative flow begins after step 2 of the main flow. 2. The user gives too many or too few arguments or the arguments are in the wrong order to the “//raise” command. 3. The system detects the invalid arguments and displays an error message.
Post conditions: None.

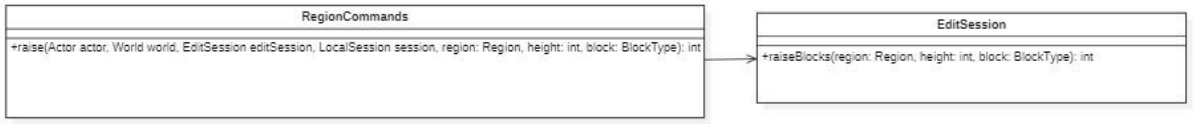
Use Case Diagram:



Sequence Diagram:



Class Diagram:



Tests:

Pre-Condition	Test Case ID	Steps	Expected Results
None.	3D Region	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,70,10). 3. The user enters //raise 10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (10, 70, 10) (770).". 3. The system raises the given blocks and prints "Raised x blocks" (x depends on how

			many stone blocks were in the region).
None.	2D Region	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,70,0). 3. The user enters //raise 10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (10, 70, 10) (110).". 3. The system raises the given blocks and prints "Raised x blocks" (x depends on how many stone blocks were in the region).
None.	1D Region	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,60,0). 3. The user enters //raise 10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (10, 60, 0) (11).". 3. The system raises the given blocks and prints "Raised x blocks" (x depends on how many stone blocks were in the region).
None.	Same position	1. The user selects the first position (0,60,0). 2. The user selects the second position (0,60,0). 3. The user enters //raise 10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (0,60, 0) (1).". 3. The system raises the selected block if present and

			gives the message "Raised 10 stone blocks" (if there was a stone block, otherwise says "Raised 0 stone blocks").
None.	Block selected not in region	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,60,10). 3. The user enters //raise 10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (10, 60, 10) (1331).". 3. The system does not find the block in the region and does not change any blocks, prints "Raised 0 blocks".
None.	Only 1 position chosen	1. The user selects the first position (0,60,0). 2. The user enters //raise 10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system does not recognize the position as a region and prints the message "Make a region selection first."
None.	Invalid Block	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,60,10). 3. The user enters //raise 10 stony on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (10, 60, 10) (1331).". 3. The system does not recognize the block and prints the

			message "Invalid value for (Not a valid block type: stony), acceptable values are any block type. Usage: //raise [height] "
None.	Negative height	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,60,10). 3. The user enters //raise -10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (10, 60, 10) (1331).". 3. The system realizes height is negative and prints the message "Height must be >= 1".
None.	Number of arguments >2	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,70,10). 3. The user enters //raise 10 stone on the chat.	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message "Second position set to (10, 70, 10) (770).". 3. The system realizes there are too many arguments and prints the message "Too many arguments. Usage: //raise [height] ".
None.	Number of arguments = 1	1. The user selects the first position (0,60,0). 2. The user selects the second position (10,70,10).	1. The system prints the message "First position set to (0, 60, 0)." 2. The system prints the message

		3. The user enters //raise stone on the chat.	"Second position set to (10, 70, 10) (770).". 3. The system raises the given blocks and prints "Raised x blocks" (x is 1 for each block of the type in the region as the default raise value if not specified is 1).

Implemented Code:

Lines 509 and 537 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/command/RegionCommands.java

New Methods:

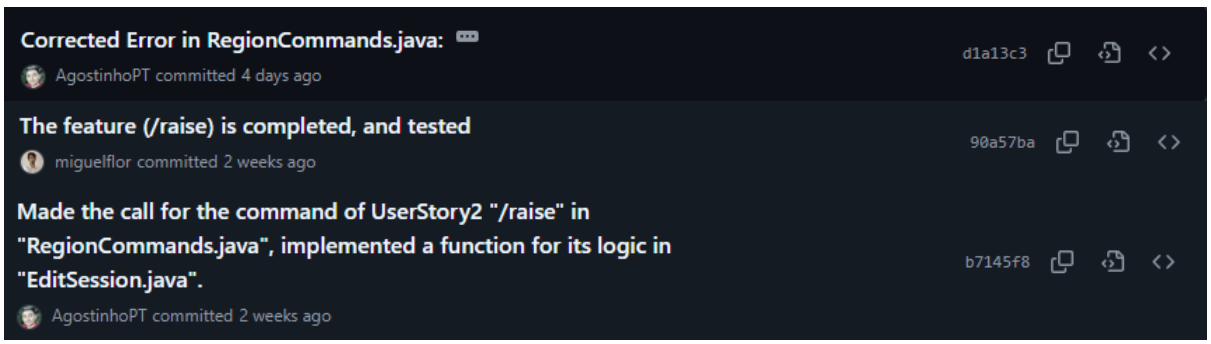
- Raise

Lines 2748 and 2781 of the file
worldedit-core/src/main/java/com/sk89q/worldedit/EditSession.java

New Methods:

- raiseBlocks

Commits:



[identification: 60593 and 62909; review: 65330 and 65969]

Updated User Story:

As an architect, **I want** a tool that fills the edges of a cuboid region and optionally fills specific faces, **so that** I can create structural frames, such as for windows, doors, and decorative trims, while retaining the flexibility to design additional face features as needed.

Use Cases:

Use Case: Fill Edges of a Cuboid Region
ID: 1
Brief Description: The system fills the edges of a cuboid region and optionally fills specific faces.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. Two positions are set by the user.
Main flow: 1. The user selects two positions in the world (the first with a left click and the second with a right click). 2. The user types the //edges command including the block type and, optionally, including flags to fill specific faces. 3. The system calculates the edges of the region based on the two positions. 4. The edges of the cuboid region are filled with the selected block. 5. If flags are provided, the system also fills the specified faces with the selected block.
Postconditions: 1. The edges of the cuboid region defined by the two positions are filled with the specified block. 2. The faces specified by the user are filled, if applicable.
Alternative Flows: InvalidPattern

Use Case: Fill Edges of a Cuboid Region
InvalidRegion InvalidFlag

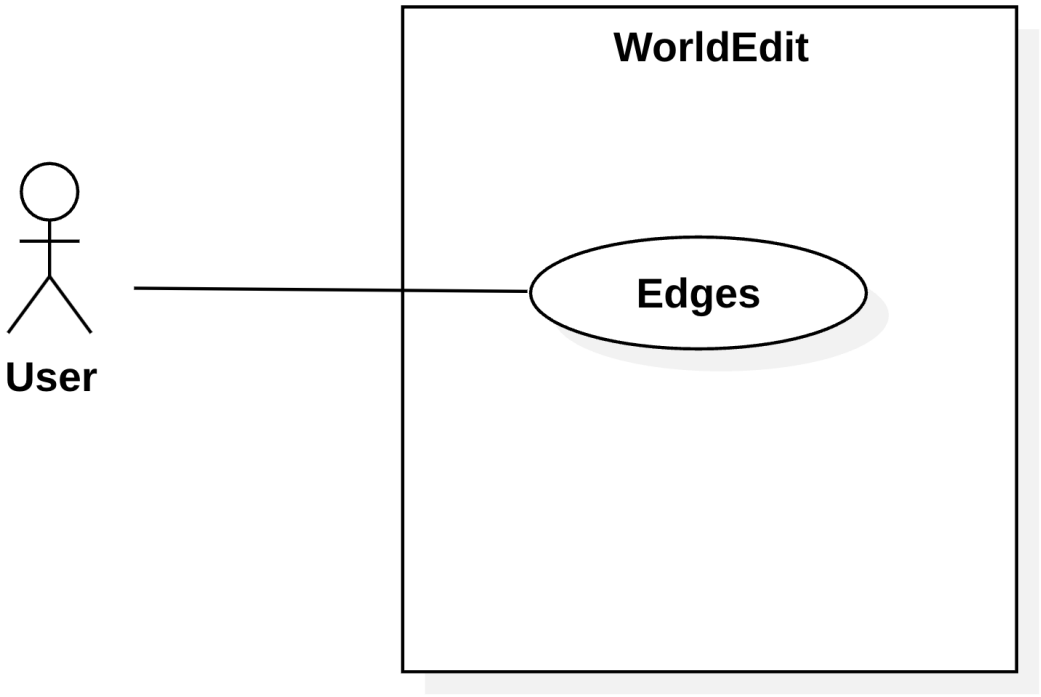
Alternative Flow: InvalidRegion
ID: 1.1
Brief Description: The system informs the user that the region chosen is invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user selects an invalid number of positions (0 or 1).
Alternative flow: 1. The user selects 0 or 1 positions. 2. The user types the //edges command including the block type and, optionally, including flags to fill specific faces. 3. The system informs the user that the region chosen is invalid.
Postconditions: None.

Alternative Flow: InvalidPattern
ID: 1.2
Brief Description: The system informs the user that the block type is invalid.
Primary Actors: User
Secondary Actors: None.

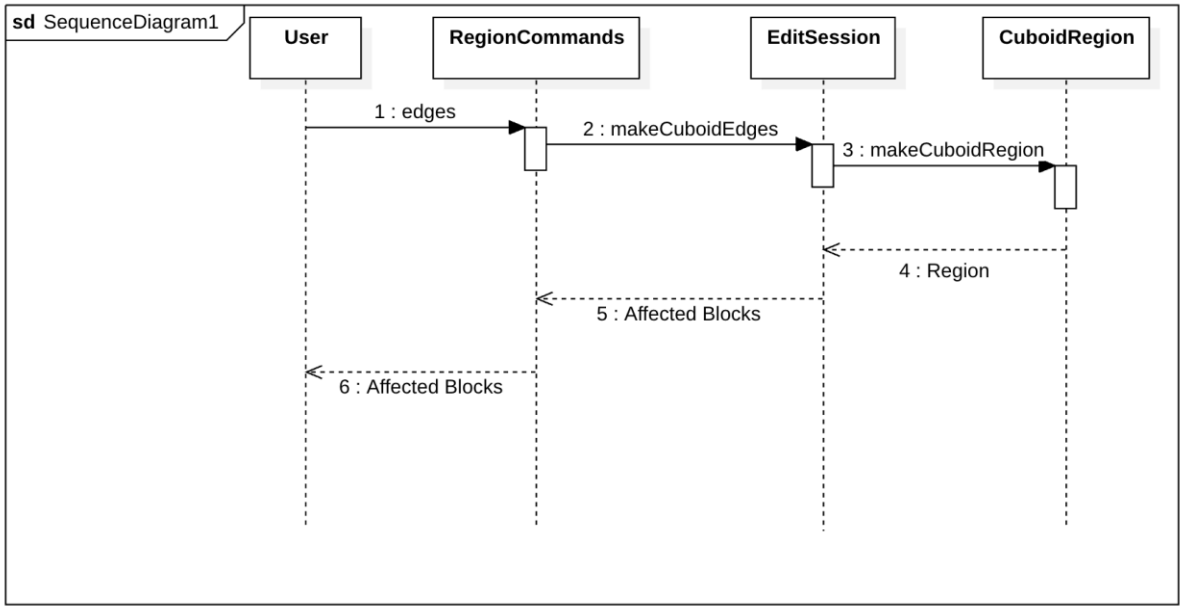
Alternative Flow: InvalidPattern
Preconditions: 1. The user specifies an unrecognized block type.
Alternative flow: 1. The alternative flow begins after step 1 of the main flow. 2. The user selects an invalid block type. 3. The system detects the invalid block type and displays an error message.
Postconditions: None.

Alternative Flow: InvalidFlag
ID: 1.3
Brief Description: The system informs the user that the flag or flags chosen are invalid.
Primary Actors: User
Secondary Actors: None.
Preconditions: 1. The user specifies an unrecognized flag or flags.
Alternative flow: 1. The alternative flow begins after step 1 of the main flow. 2. The user selects one or more invalid flags. 3. The system detects the invalid flag or flags and displays an error message.
Postconditions: None.

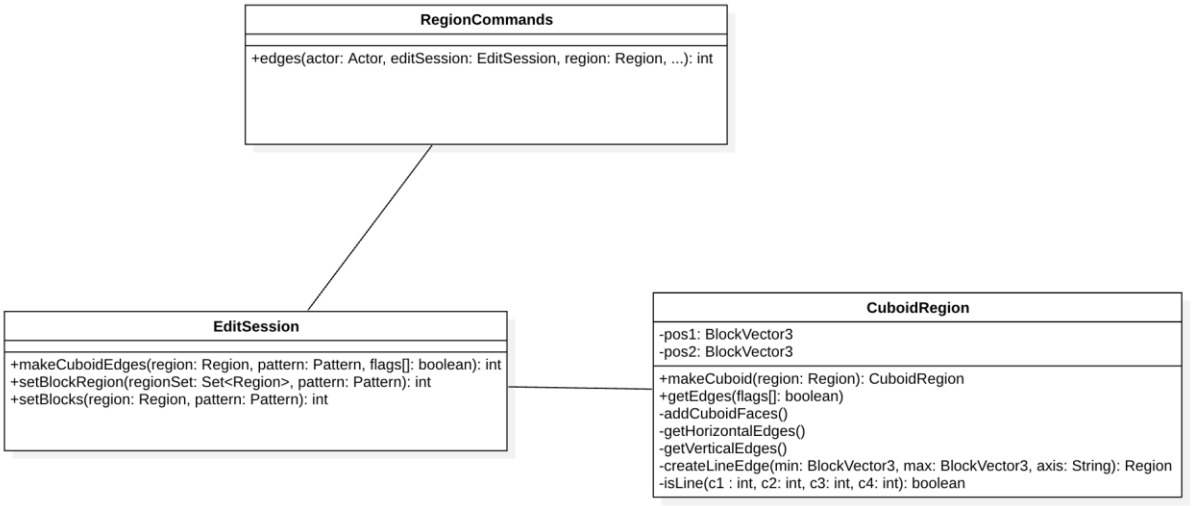
Use Case Diagram:



Sequence Diagram:



Class Diagram:



Tests:

Pre-Condition	Test case id	Steps	Expected Results
None.	3D Region	1. The user selects the first position (0,0,0). 2. The user selects the second position (10,10,10). 3. The user enters //edges stone on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (10, 10, 10) (1331).". 3. The system sets the edges of the region selected to stone and prints the message "132 blocks have been changed."
None.	3D Region with flags	1. The user selects the first position (0,0,0). 2. The user selects the second position (10,10,10). 3. The user enters //edges -cf stone on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (10, 10, 10) (1331).". 3. The system sets the edges, the "ceiling" and the "floor" of the region selected to stone and prints the message "294 blocks have been changed."
None.	2D Region	1. The user selects the first position (0,0,0). 2. The user selects the second position (10,10,0). 3. The user enters //edges stone on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (10, 10, 0) (121).". 3. The system sets the edges of the region selected to stone and prints the message "92 blocks have been changed."
None.	2D Region with flags	1. The user selects the first position (0,0,0). 2. The user selects the second position (10,10,0). 3. The user enters //edges stone -cnw on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (10, 10, 0) (121).". 3. The system sets the edges and the face of the region selected to stone and prints the message "173 blocks have been changed."

Pre-Condition	Test case id	Steps	Expected Results
None.	1D Region	1. The user selects the first position (0,0,0). 2. The user selects the second position (0,0,10). 3. The user enters //edges stone on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (0, 0, 10) (11)." 3. The system sets the region selected to stone and prints the message "11 blocks have been changed."
None.	1D Region with flags	1. The user selects the first position (0,0,0). 2. The user selects the second position (0,0,10). 3. The user enters //edges stone -cefnsw on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (10, 0, 0) (11).". 3. The system sets the region selected to stone and prints the message "11 blocks have been changed."
None.	Same position	1. The user selects the first position (0,0,0). 2. The user selects the second position (0,0,0). 3. The user enters //edges stone on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (0, 0, 0) (1).". 3. The system sets the block selected to stone and prints the message "1 blocks have been changed."
None.	Same position with flags	1. The user selects the first position (0,0,0). 2. The user selects the second position (0,0,0). 3. The user enters //edges stone -cefnsw on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (0, 0, 0) (1).". 3. The system sets the block selected to stone and prints the message "1 blocks have been changed."

Pre-Condition	Test case id	Steps	Expected Results
None.	Invalid block	1. The user selects the first position (0,0,0). 2. The user selects the second position (10,10,10). 3. The user enters //edges stones on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (10, 10, 10) (1331).". 3. The system does not recognize the pattern selected and prints the message "Invalid value for <pattern> (Block name 'stones' was not recognized.), acceptable values are any pattern Usage: //edges [-cefnsw] <pattern>"
None.	Only 1 position chosen	1. The user selects the first position (0,0,0). 2. The user enters //edges stone on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system does not recognize the position as a region and prints the message "Make a region selection first."
None.	Invalid flag	1. The user selects the first position (0,0,0). 2. The user selects the second position (10,10,10). 3. The user enters //edges stone -q on the chat.	1. The system prints the message "First position set to (0, 0, 0)." 2. The system prints the message "Second position set to (10, 10, 10) (1331).". 3. The system does not recognize the flag given and prints the message "Too many arguments. Usage: //edges [-cefnsw] <pattern>"

Implemented Code:

Lines 1290-1307 of the file worldedit-core/src/main/java/com/sk89q/wordedit/EditSession

---

```java
/**
 * Make the edges of the given region as if it was a {@link CuboidRegion}.
 *
 * @param region      the region
 * @param pattern     the pattern to place
 * @param flags - list of flags corresponding to the faces that are to be filled
 * @return number of blocks affected
 * @throws MaxChangedBlocksException thrown if too many blocks are changed
 */
public int makeCuboidEdges(Region region, Pattern pattern,  1 usage  Nux404 *
                           boolean[] flags) throws MaxChangedBlocksException {
    checkNotNull(region);
    checkNotNull(pattern);

    CuboidRegion cuboid = CuboidRegion.makeCuboid(region);
    Set<Region> edges = cuboid.getEdges(flags);
    return setBlockRegion(edges, pattern);
}
```

Lines 137-163 of the file worldedit-core/src/main/java/com/sk89q/worldedit/regions/CuboidRegion

---

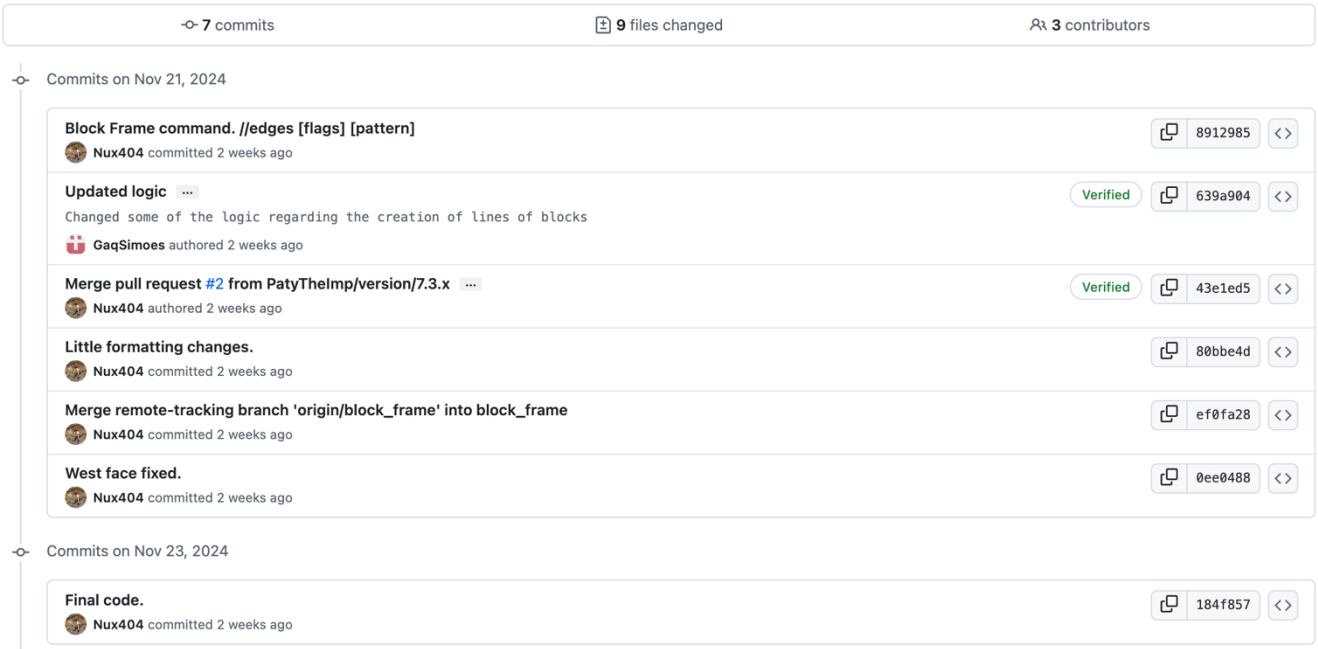
```java
/**
 * Get a set of regions that contains the edges or faces of this cuboid
 *
 * @param flags: Array of flags representing which faces to fill
 * @return a new set of regions representing the edges or faces of the cuboid.
 */
public Set<Region> getEdges(boolean[] flags) {  1 usage   Nux404
    Set<Region> edges = new HashSet<>();
    BlockVector3 min = getMinimumPoint();
    BlockVector3 max = getMaximumPoint();

    if (min.equals(max))
        edges.add(new CuboidRegion(min, min));
    else if (isLine(pos1.z(), pos2.z(), pos1.y(), pos2.y()))
        edges.add(createLineEdge(min, max,  axis: "x"));
    else if (isLine(pos1.x(), pos2.x(), pos1.z(), pos2.z()))
        edges.add(createLineEdge(min, max,  axis: "y"));
    else if (isLine(pos1.x(), pos2.x(), pos1.y(), pos2.y()))
        edges.add(createLineEdge(min, max,  axis: "z"));
    else {
        addCuboidFaces(edges, flags, min, max);
        getHorizontalEdges(edges, min, max, min.y()); // Bottom face edges
        getHorizontalEdges(edges, min, max, max.y()); // Top face edges
        getVerticalEdges(edges, min, max);            // Vertical face edges
    }
    return edges;
}
```

Lines 263-283 of the file Worldedit-core/src/main/java/com/sk89q/worldedit/command/RegionCommands

---

```java
@Command( 1 usage  Nux404
        name = "/edges",
        desc = "Build the edges of a selection"
)
@CommandPermissions("worldedit.region.edges")
@Logging(REGION)
public int edges(Actor actor, EditSession editSession, @Selection Region region,
                 @Arg(desc = "The pattern of blocks to set")
                 Pattern pattern,
                 @Switch(name = 'w', desc = "fill the west face") boolean fillWest,
                 @Switch(name = 'e', desc = "fill the east face") boolean fillEast,
                 @Switch(name = 'n', desc = "fill the north face") boolean fillNorth,
                 @Switch(name = 's', desc = "fill the south face") boolean fillSouth,
                 @Switch(name = 'c', desc = "fill the ceiling") boolean fillCeiling,
                 @Switch(name = 'f', desc = "fill the floor") boolean fillFloor) throws WorldEditException {

    boolean[] flags = {fillWest, fillEast, fillNorth, fillSouth, fillCeiling, fillFloor};
    int affected = editSession.makeCuboidEdges(region, pattern, flags);
    actor.printInfo(TranslatableComponent.of( key: "worldedit.edges.changed", TextComponent.of(affected)));
    return affected;
}
```

Commits:



Link for the Demo Video:

https://youtu.be/rAM4b-w_V3Q?si=bLVHTeW7-F82qVV6

Conclusion:

In conclusion, our Software Engineering project allowed us to explore and extend the functionality of WorldEdit, a powerful open-source mod for Minecraft. Throughout this journey, we collaborated as a team using Scrum methodology, focusing on user stories, identifying design patterns, and addressing code smells. By adding new features and improving the codebase, we enhanced the mod's capabilities and contributed to an existing open-source project, which was both a rewarding and educational experience.

This project has not only strengthened our skills in teamwork, agile practices, and software design but has also ignited a deeper appreciation for the collaborative nature of open-source development. Inspired by this experience, we are eager to contribute to more open-source projects in the future, applying what we've learned to help improve software tools and give back to the developer community.