Task 4: Agentic Refactoring Documentation

Agentic Tool Information

- **Tool/System Used:** Antigravity (Google Deepmind AI Coding Agent)
- **Model:** Gemini-based LLM with file analysis capabilities
- **Approach:** Single-file agentic analysis with 3-stage workflow

---

Command/Prompt Used

```
Analyze [filename] for design smells. For each smell:
1. Identify the smell type and location
2. Explain why it's a problem
3. Propose a refactoring strategy
4. Show before/after code snippets
```

File 1: JPAWeblogEntryManagerImpl.java

Path: `app/src/main/java/org/apache/roller/weblogger/business/jpa/JPAWeblogEntryManagerImpl.java`

Stage 1: Smell Identification

Smells Found: - **God Class** (HIGH) - Lines 63-1393 - **Large Class** (HIGH) - 1394 lines total - **Long Method** (MEDIUM) - `getWeblogEntries()` (95 lines) - **Feature Envy** (MEDIUM) - Multiple methods accessing external managers

Detailed Analysis

God Class Smell: This class handles **6 distinct responsibilities**:

1. **Category Management** (lines 93-154):
    - `saveWeblogCategory`, `removeWeblogCategory`, `moveWeblogCategoryContents`
2. **Comment Management** (lines 159-718):
    - `saveComment`, `removeComment`, `getComments`, `removeMatchingComments`
3. **Entry Management** (lines 182-296):
    - `saveWeblogEntry`, `removeWeblogEntry`
4. **Tag Management** (lines 512-1107):
    - `updateTagCount`, `getPopularTags`, `getTags`, `getTagComboExists`
5. **Hit Count Management** (lines 1191-1328):
    - `getHitCount`, `incrementHitCount`, `resetHitCount`
6. **Query Building** (lines 298-475):
    - Dynamic JPQL query construction

Metrics: - 49 public methods - 7 fields - 1394 lines of code

Stage 2: Refactoring Planning

Proposed Strategy: Extract Class Refactoring

Split `JPAWeblogEntryManagerImpl` into focused classes following Single Responsibility Principle:

JPACategoryManager

- **Purpose:** Category CRUD operations
- **Methods:**
  - `saveWeblogCategory`
  - `removeWeblogCategory`
  - `moveWeblogCategoryContents`
  - `getWeblogCategory`
  - `getWeblogCategoryByName`

JPACommentManager

- **Purpose:** Comment operations
- **Methods:**
  - `saveComment`
  - `removeComment`
  - `getComments`
  - `removeMatchingComments`
  - `getCommentCount`

JPATagManager

- **Purpose:** Tag aggregation
- **Methods:**
  - `updateTagCount`
  - `getPopularTags`
  - `getTags`
  - `getTagComboExists`
  - `removeWeblogEntryTag`

JPAHitCountManager

- **Purpose:** Analytics operations
- **Methods:**
  - `getHitCount`
  - `getHitCountByWeblog`
  - `incrementHitCount`
  - `resetHitCount`
  - `getHotWeblogs`

Why This Refactoring: - **Reduces class complexity** from 49 methods to ~12 per class - **Improves testability** - each class can be unit tested independently - **Enhances maintainability** - changes to tag logic don’t affect comment logic - **Follows SOLID principles** - Single Responsibility Principle

Stage 3: Refactoring Execution

Before (Original Code - God Class chunk):

```
// Lines 512-600 - Tag management mixed with entry management
@com.google.inject.Singleton
public class JPAWeblogEntryManagerImpl implements WeblogEntryManager {

    private final Weblogger roller;
    private final JPAPersistenceStrategy strategy;

    // 49 methods handling everything...

    private void removeWeblogEntryTag(WeblogEntryTag tag) throws
        WebloggerException {
        if (tag.getWeblogEntry().isPublished()) {
            updateTagCount(tag.getName(), tag.getWeblogEntry().getWebsite(), -1);
        }
        this.strategy.remove(tag);
    }

    private void updateTagCount(String name, Weblog website, int amount)
    throws WebloggerException {
        if (amount == 0) {
            throw new WebloggerException("Tag increment amount cannot be zero.");
        }
        // ... 60 more lines of tag logic
    }

    @Override
    public List<TagStat> getPopularTags(Weblog website, Date startDate, int
        offset, int limit)
    throws WebloggerException {
        // ... tag-specific query logic
    }
}
```

After (Refactored - Extracted JPATagManager):

```
// NEW FILE: JPATagManager.java
package org.apache.roller.weblogger.business.jpa;

@com.google.inject.Singleton
public class JPATagManager {

    private final JPAPersistenceStrategy strategy;

    @com.google.inject.Inject
    protected JPATagManager(JPAPersistenceStrategy strategy) {
        this.strategy = strategy;
    }

    public void removeWeblogEntryTag(WeblogEntryTag tag) throws WebloggerException
        {
        if (tag.getWeblogEntry().isPublished()) {
            updateTagCount(tag.getName(), tag.getWeblogEntry().getWebsite(), -1);
        }
        this.strategy.remove(tag);
    }

    public void updateTagCount(String name, Weblog website, int amount)
    throws WebloggerException {
```

```
        if (amount == 0) {
            throw new WebloggerException("Tag increment amount cannot be zero.");
        }

        if (website == null) {
            throw new WebloggerException("Website cannot be NULL.");
        }

        TypedQuery<WeblogEntryTagAggregate> weblogQuery = strategy.getNamedQuery(
                "WeblogEntryTagAggregate.getByName&WebsiteOrderByLastUsedDesc",
                WeblogEntryTagAggregate.class);
        weblogQuery.setParameter(1, name);
        weblogQuery.setParameter(2, website);

        WeblogEntryTagAggregate weblogTagData = getSingleResultOrNull(weblogQuery);
        Timestamp lastUsed = new Timestamp((new Date()).getTime());

        if (weblogTagData == null && amount > 0) {
            weblogTagData = new WeblogEntryTagAggregate(null, website, name,
         amount);
            weblogTagData.setLastUsed(lastUsed);
            strategy.store(weblogTagData);
        } else if (weblogTagData != null) {
            weblogTagData.setTotal(weblogTagData.getTotal() + amount);
            weblogTagData.setLastUsed(lastUsed);
            strategy.store(weblogTagData);
        }

        // Clean up zero counts
        Query removeq = strategy.getNamedUpdate(
            "WeblogEntryTagAggregate.removeByTotalLessEqual");
        removeq.setParameter(1, 0);
        removeq.executeUpdate();
    }

    public List<TagStat> getPopularTags(Weblog website, Date startDate, int
         offset, int limit)
    throws WebloggerException {
        TypedQuery<TagStat> query = buildPopularTagsQuery(website, startDate);
        setFirstMax(query, offset, limit);
        return processTagResults(query.getResultList());
    }

    public List<TagStat> getTags(Weblog website, String sortBy, String startsWith,
                                 int offset, int limit) throws WebloggerException
         {
        // Tag listing logic
    }

    public boolean getTagComboExists(List<String> tags, Weblog weblog)
    throws WebloggerException {
        // Tag combination check
    }
}
```

Refactored JPAWeblogEntryManagerImpl (delegating to new classes):

```
// MODIFIED: JPAWeblogEntryManagerImpl.java - now delegates to specialized managers
@com.google.inject.Singleton
public class JPAWeblogEntryManagerImpl implements WeblogEntryManager {

    private final Weblogger roller;
    private final JPAPersistenceStrategy strategy;
    private final JPATagManager tagManager;      // NEW: Injected
```

```
    private final JPACommentManager commentManager;  // NEW: Injected

    @com.google.inject.Inject
    protected JPAWeblogEntryManagerImpl(
            Weblogger roller,
            JPAPersistenceStrategy strategy,
            JPATagManager tagManager,
            JPACommentManager commentManager) {
        this.roller = roller;
        this.strategy = strategy;
        this.tagManager = tagManager;
        this.commentManager = commentManager;
    }

    // Delegate tag operations
    @Override
    public List<TagStat> getPopularTags(Weblog website, Date startDate, int
        offset, int limit)
    throws WebloggerException {
        return tagManager.getPopularTags(website, startDate, offset, limit);
    }

    // Entry-specific logic remains here
    @Override
    public void saveWeblogEntry(WeblogEntry entry) throws WebloggerException {
        // Core entry saving logic (reduced complexity)
    }
}
```

File 2: Weblog.java

Path: `app/src/main/java/org/apache/roller/weblogger/pojos/Weblog.java`

Stage 1: Smell Identification

Smells Found: - **God Class** (HIGH) - Lines 51-926 - **Large Class** (HIGH) - 926 lines, 36 fields - **Feature Envy** (MEDIUM) - Methods calling WebloggerFactory extensively - **Data Class with behavior** (MEDIUM) - POJO with business logic

Detailed Analysis

God Class Smell: The `Weblog` POJO class has **95+ methods** and handles multiple concerns:

1. **Data Properties** (36 fields): id, handle, name, tagline, emailAddress, etc.
2. **Business Logic**: `getRecentWeblogEntries()`, `getPopularTags()`, `getTodaysHits()`
3. **External Service Access**: Calls to `WebloggerFactory.getWeblogger()` throughout
4. **Permission Checking**: `hasUserPermission()`, `hasUserPermissions()`

Feature Envy: Methods like `getRecentWeblogEntries()`, `getPopularTags()` should belong to a service class, not a POJO.

Stage 2: Refactoring Planning

Proposed Strategy: Move Method + Extract Class

Refactoring Plan:

Move to WeblogService: - `getRecentWeblogEntries()` - `getRecentWeblogEntriesByTag()` - `getRecentComments()` - `getTodaysHits()` - `getPopularTags()` - `getCommentCount()` - `hasUserPermission()` - `hasUserPermissions()`

Keep in Weblog: - Getters/Setters - `equals()` - `hashCode()`

Why This Refactoring: - **Separates concerns** - POJO should only hold data - **Reduces coupling** - Remove `WebloggerFactory` dependencies from entity - **Improves testability** - Entity objects can be tested without mocking services

Stage 3: Refactoring Execution

Before (Original - Business logic in POJO):

```
// Lines 641-663 - Business logic inside POJO entity
public class Weblog implements Serializable {

    // ... 36 fields ...

    public List<WeblogEntry> getRecentWeblogEntries(String cat, int length) {
        if (cat != null && "nil".equals(cat)) {
            cat = null;
        }
        if (length > MAX_ENTRIES) {
            length = MAX_ENTRIES;
        }
        if (length < 1) {
            return Collections.emptyList();
        }
        try {
            WeblogEntryManager wmgr = WebloggerFactory
                .getWeblogger().getWeblogEntryManager();
            WeblogEntrySearchCriteria wesc = new WeblogEntrySearchCriteria();
            wesc.setWeblog(this);
            wesc.setCatName(cat);
            wesc.setStatus(PubStatus.PUBLISHED);
            wesc.setMaxResults(length);
            return wmgr.getWeblogEntries(wesc);
        } catch (WebloggerException e) {
            log.error("ERROR: getting recent entries", e);
        }
        return Collections.emptyList();
    }

    public int getTodaysHits() {
        try {
            Weblogger roller = WebloggerFactory.getWeblogger();
            WeblogEntryManager mgr = roller.getWeblogEntryManager();
            WeblogHitCount hitCount = mgr.getHitCountByWeblog(this);
            return (hitCount != null) ? hitCount.getDailyHits() : 0;
        } catch (WebloggerException e) {
            log.error("Error getting weblog hit count", e);
        }
        return 0;
```

```
    }
}
```

After (Refactored - Pure POJO + Service):

```
// MODIFIED: Weblog.java - Pure data entity
public class Weblog implements Serializable {

    public static final long serialVersionUID = 206437645033737127L;

    // Only data fields
    private String id = UUIDGenerator.generateUUID();
    private String handle;
    private String name;
    private String tagline;
    private Boolean enableBloggerApi = Boolean.TRUE;
    private String emailAddress;
    private String locale;
    private String timeZone;
    private Boolean visible = Boolean.TRUE;
    private Date dateCreated = new Date();
    private Date lastModified = new Date();
    // ... other fields ...

    // Only getters/setters - NO business logic
    public String getId() { return this.id; }
    public void setId(String id) { this.id = id; }

    public String getHandle() { return this.handle; }
    public void setHandle(String handle) { this.handle = handle; }

    // ... more getters/setters ...

    @Override
    public boolean equals(Object other) {
        if (other == this) return true;
        if (!(other instanceof Weblog)) return false;
        Weblog o = (Weblog)other;
        return new EqualsBuilder()
            .append(getHandle(), o.getHandle())
            .isEquals();
    }

    @Override
    public int hashCode() {
        return new HashCodeBuilder()
            .append(getHandle())
            .toHashCode();
    }
}
```

```
// NEW FILE: WeblogService.java - Business logic extracted
package org.apache.roller.weblogger.business;

@com.google.inject.Singleton
public class WeblogService {

    private final WeblogEntryManager entryManager;
    private final UserManager userManager;

    @com.google.inject.Inject
```

```
    public WeblogService(WeblogEntryManager entryManager, UserManager userManager)
        {
        this.entryManager = entryManager;
        this.userManager = userManager;
    }

    public List<WeblogEntry> getRecentWeblogEntries(Weblog weblog, String cat, int
        length) {
        if (cat != null && "nil".equals(cat)) {
            cat = null;
        }
        length = Math.min(Math.max(length, 0), 100);
        if (length < 1) {
            return Collections.emptyList();
        }

        try {
            WeblogEntrySearchCriteria wesc = new WeblogEntrySearchCriteria();
            wesc.setWeblog(weblog);
            wesc.setCatName(cat);
            wesc.setStatus(PubStatus.PUBLISHED);
            wesc.setMaxResults(length);
            return entryManager.getWeblogEntries(wesc);
        } catch (WebloggerException e) {
            LOG.error("ERROR: getting recent entries", e);
            return Collections.emptyList();
        }
    }

    public int getTodaysHits(Weblog weblog) {
        try {
            WeblogHitCount hitCount = entryManager.getHitCountByWeblog(weblog);
            return (hitCount != null) ? hitCount.getDailyHits() : 0;
        } catch (WebloggerException e) {
            LOG.error("Error getting weblog hit count", e);
            return 0;
        }
    }

    public List<TagStat> getPopularTags(Weblog weblog, int sinceDays, int length) {
        Date startDate = null;
        if (sinceDays > 0) {
            Calendar cal = Calendar.getInstance();
            cal.add(Calendar.DATE, -sinceDays);
            startDate = cal.getTime();
        }

        try {
            return entryManager.getPopularTags(weblog, startDate, 0, length);
        } catch (WebloggerException e) {
            LOG.error("ERROR: fetching popular tags", e);
            return Collections.emptyList();
        }
    }

    public boolean hasUserPermission(Weblog weblog, User user, String action) {
        try {
            WeblogPermission perms = new WeblogPermission(weblog, user,
                Collections.singletonList(action));
            return userManager.checkPermission(perms, user);
        } catch (WebloggerException ex) {
            LOG.error("ERROR checking user permission", ex);
            return false;
        }
```

```
    }
}
```

File 3: Utilities.java

Path: `app/src/main/java/org/apache/roller/weblogger/util/Utilities.java`

Stage 1: Smell Identification

Smells Found: - **God Class** (HIGH) - Lines 38-1066 - **Large Class** (HIGH) - 1066 lines, 43 methods - **Utility Class Anti-pattern** (MEDIUM) - Too many unrelated utilities

Detailed Analysis

God Class Smell: The `Utilities` class is a “kitchen sink” with 43+ static methods covering:

1. **HTML Processing** (15 methods): `escapeHTML`, `removeHTML`, `addNofollow`, etc.
2. **String Manipulation** (8 methods): `replaceNonAlphanumeric`, `truncate`, etc.
3. **File I/O** (5 methods): `copyFile`, `streamToString`, `copyInputToOutput`
4. **Encoding** (4 methods): `encodePassword`, `encodeString`, `decodeString`
5. **Email Validation** (2 methods): `isValidEmailAddress`
6. **Array Conversion** (6 methods): `stringArrayToString`, `intArrayToString`

Stage 2: Refactoring Planning

Proposed Strategy: Extract Class by Cohesion

New Classes to Extract:

HtmlUtils

- **Methods:**
  - `escapeHTML`
  - `unescapeHTML`
  - `removeHTML`
  - `addNofollow`
  - `extractHTML`
- **Purpose:** HTML processing

StringTruncator

- **Methods:**
  - `truncate`
  - `truncateNicely`
  - `truncateText`
- **Purpose:** Text truncation

FileUtils

- **Methods:**
  - `copyFile`
  - `streamToString`
  - `copyInputToOutput`
- **Purpose:** File operations

EncodingUtils

- **Methods:**
  - `encodePassword`
  - `encodeString`
  - `decodeString`
- **Purpose:** Encoding/hashing

Why This Refactoring: - **Improves cohesion** - each class has single purpose - **Better discoverability** - developers find methods faster - **Easier testing** - focused test suites per utility domain

Stage 3: Refactoring Execution

Before (Original - Kitchen Sink Utility):

```
// Lines 1-250 - Mixed utilities
public class Utilities {

    private static final Log mLogger = LogFactory.getLog(Utilities.class);

    // HTML methods
    public static String escapeHTML(String s) { /* ... */ }
    public static String removeHTML(String str) { /* ... */ }
    public static String addNofollow(String html) { /* ... */ }

    // String methods
    public static String replaceNonAlphanumeric(String str) { /* ... */ }
    public static String stringArrayToString(String[] arr, String delim) { /* ...
        */ }

    // File methods
    public static void copyFile(File from, File to) throws IOException { /* ...
        */ }
    public static String streamToString(InputStream is) throws IOException
        { /* ... */ }

    // Encoding methods
    public static String encodePassword(String password, String algorithm)
        { /* ... */ }
    public static String encodeString(String str) throws IOException { /* ... */ }

    // Truncation methods
    public static String truncate(String str, int lower, int upper, String end) { /
        * ... */ }
    public static String truncateNicely(String str, int lower, int upper, String
        end) { /* ... */ }
}
```

After (Refactored - Focused Utility Classes):

```java
// NEW FILE: HtmlUtils.java
package org.apache.roller.weblogger.util;

/**
 * Utilities for HTML processing and sanitization.
 */
public final class HtmlUtils {

    private HtmlUtils() {} // Prevent instantiation

    private static final Pattern LINK_PATTERN = Pattern.compile(
        "<a href=.*?>", Pattern.CASE_INSENSITIVE);

    /**
     * Escape HTML special characters.
     */
    public static String escape(String s) {
        return escape(s, true);
    }

    public static String escape(String s, boolean escapeAmpersand) {
        if (escapeAmpersand) {
            s = StringUtils.replace(s, "&", "&amp;");
        }
        s = StringUtils.replace(s, " ", " ");
        s = StringUtils.replace(s, "\"", "&quot;");
        s = StringUtils.replace(s, "<", "&lt;");
        s = StringUtils.replace(s, ">", "&gt;");
        return s;
    }

    public static String unescape(String str) {
        return StringEscapeUtils.unescapeHtml4(str);
    }

    /**
     * Remove all HTML tags from string.
     */
    public static String stripTags(String str) {
        return stripTags(str, true);
    }

    public static String stripTags(String str, boolean addSpace) {
        if (str == null) return "";
        StringBuilder ret = new StringBuilder(str.length());
        int start = 0;
        int beginTag = str.indexOf('<');

        while (beginTag >= start) {
            if (beginTag > 0) {
                ret.append(str.substring(start, beginTag));
                if (addSpace) ret.append(" ");
            }
            int endTag = str.indexOf('>', beginTag);
            if (endTag > -1) {
                start = endTag + 1;
                beginTag = str.indexOf('<', start);
            } else {
                ret.append(str.substring(beginTag));
                break;
            }
        }
```

```
        if (str.indexOf('>') > -1) {
            ret.append(str.substring(str.lastIndexOf('>') + 1));
        }
        return ret.toString().trim();
    }

    /**
     * Add rel="nofollow" to all links.
     */
    public static String addNofollow(String html) {
        if (html == null || html.isEmpty()) return html;

        Matcher m = LINK_PATTERN.matcher(html);
        StringBuilder buf = new StringBuilder();

        while (m.find()) {
            String link = html.substring(m.start(), m.end());
            buf.append(html.substring(0, m.start()));
            if (!link.contains("rel=\"nofollow\"")) {
                buf.append(link.substring(0, link.length() - 1))
                   .append(" rel=\"nofollow\">");
            } else {
                buf.append(link);
            }
            html = html.substring(m.end());
            m = LINK_PATTERN.matcher(html);
        }
        buf.append(html);
        return buf.toString();
    }
}
```

```
// NEW FILE: EncodingUtils.java
package org.apache.roller.weblogger.util;

/**
 * Utilities for password hashing and Base64 encoding.
 */
public final class EncodingUtils {

    private EncodingUtils() {}

    /**
     * Hash password using specified algorithm.
     */
    public static String hashPassword(String password, String algorithm) {
        try {
            MessageDigest md = MessageDigest.getInstance(algorithm);
            md.reset();
            md.update(password.getBytes());
            byte[] encoded = md.digest();

            StringBuilder buf = new StringBuilder();
            for (byte b : encoded) {
                if ((b & 0xff) < 0x10) buf.append("0");
                buf.append(Long.toString(b & 0xff, 16));
            }
            return buf.toString();
        } catch (Exception e) {
            return password; // Fallback to plain password
        }
    }
```

```
    public static String base64Encode(String str) {
        return new String(Base64.encodeBase64(str.getBytes())).trim();
    }

    public static String base64Decode(String str) {
        return new String(Base64.decodeBase64(str.getBytes()));
    }
}
```

```
// NEW FILE: TextTruncator.java
package org.apache.roller.weblogger.util;

/**
 * Utilities for smart text truncation.
 */
public final class TextTruncator {

    private TextTruncator() {}

    /**
     * Truncate text at word boundary, appending suffix.
     *
     * @param text Text to truncate
     * @param minLength Minimum characters before looking for break
     * @param maxLength Maximum characters allowed
     * @param suffix String to append (e.g., "...")
     */
    public static String truncate(String text, int minLength, int maxLength,
        String suffix) {
        String stripped = HtmlUtils.stripTags(text, false);

        if (maxLength < minLength) maxLength = minLength;
        if (stripped.length() <= maxLength) return stripped;

        int breakPoint = stripped.lastIndexOf(' ', maxLength);
        if (breakPoint >= minLength) {
            return stripped.substring(0, breakPoint) + suffix;
        }
        return stripped.substring(0, maxLength) + suffix;
    }

    /**
     * Truncate preserving HTML structure.
     */
    public static String truncatePreservingHtml(String html, int minLength,
                                                int maxLength, String suffix) {
        // Implementation preserves closing tags
        // ...
    }
}
```

Summary of Refactorings

File 1: JPAWeblogEntryManagerImpl.java

- **Original:** 1394 LOC, 49 methods
- **Smells Found:** God Class, Large Class
- **Refactoring Applied:** Extract Class
- **Result:** 4 focused managers (~12 methods each)

File 2: Weblog.java

- **Original:** 926 LOC, 95 methods
- **Smells Found:** God Class, Feature Envy
- **Refactoring Applied:** Move Method + Extract Class
- **Result:** Pure POJO + WeblogService

File 3: Utilities.java

- **Original:** 1066 LOC, 43 methods
- **Smells Found:** Large Class, Utility Anti-pattern
- **Refactoring Applied:** Extract Class by Cohesion
- **Result:** 4 focused utility classes

Benefits of Proposed Refactorings

1. **Single Responsibility Principle** - Each class has one reason to change
2. **Improved Testability** - Smaller, focused classes are easier to unit test
3. **Better Maintainability** - Changes are localized to specific classes
4. **Enhanced Discoverability** - Developers can find functionality faster
5. **Reduced Coupling** - Dependencies are explicit through constructor injection