AI-based static analysis for clean code

Sudarshan Bhide – CTO, Acellere GmbH
Agenda

• Introduction
• Gamma’s AI Features
• Technical Overview
• Examples
• Benefits
Conventional static analysis – What it can do

- Can find issues which are implemented up-front as rules (e.g. check for a memory leak, or a buffer over-run)
- Proven and powerful technique
- Used for finding robustness issues, compliance checks, etc.

**Example: CloseResource rule**

Can be detected by conventional approach
Conventional static analysis – limitation

• Although rule-based analysis is a proven and powerful technique, it cannot find issues (functional or non-functional) beyond the checks implemented as rules.
• Writing new rules for every such condition is not feasible / possible.

Cannot be detected by conventional approach, without writing a complex rule.
Coverage of Issues by static analysis tools

- Several issues found during testing are not detectable by existing tools at development time.
- Some issues detected by tools also manifest as functional issues, if not addressed (e.g. bad exception handling can lead to unwanted functional behavior).
- If more such issues are detected at development time, testing effort will reduce.
Design has a strong correlation with bugs

- Amount of bugs increase with lack of cohesion (LCOM) and deep inheritance trees (high DOIH)
- Issues such as lack of separation of concerns, lack of encapsulation contribute to bugs
Motivation

• Find new issues based on actual bugs detected (in addition to rule-based analysis)
  – Issues in code are not only limited to what can be found through standard checks
  – Issues can also occur due to usage of incorrect APIs, incorrect decision points, logical flow errors, etc.
  – Since these checks cannot be implemented in advance, an alternate way to uncover such issues is needed

• Suggest fixes
  – When an issue is detected, a possible suggestion to fix the issue is equally important in order to address the issues quickly

• Analyze software against design anti-patterns for long-term maintainability
Introducing Gamma

- Gamma is a code quality improvement platform which analyzes code as well as design quality.
- Gamma’s rating engine enables identification of the most relevant code components.
Multi-dimensional Analyzer

<table>
<thead>
<tr>
<th>Components</th>
<th>Risk</th>
<th>Overall Rating</th>
<th>Design</th>
<th>Metrics</th>
<th>Duplication</th>
<th>Code Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>StorageService</td>
<td>5.00</td>
<td>-4.12</td>
<td>-5.00</td>
<td>-4.55</td>
<td>5.00</td>
<td>-2.50</td>
</tr>
<tr>
<td>ModificationStatement</td>
<td>3.32</td>
<td>-3.71</td>
<td>-4.06</td>
<td>-3.52</td>
<td>5.00</td>
<td>-3.42</td>
</tr>
<tr>
<td>SchemaKeyspace</td>
<td>3.16</td>
<td>-3.42</td>
<td>-4.60</td>
<td>-3.70</td>
<td>4.53</td>
<td>-1.56</td>
</tr>
<tr>
<td>StorageProxy</td>
<td>4.10</td>
<td>-3.31</td>
<td>-4.60</td>
<td>-4.39</td>
<td>5.00</td>
<td>-0.50</td>
</tr>
<tr>
<td>ColumnFamilyStore</td>
<td>4.84</td>
<td>-3.29</td>
<td>-4.60</td>
<td>-4.33</td>
<td>5.00</td>
<td>-0.50</td>
</tr>
<tr>
<td>SSTableReader</td>
<td>3.18</td>
<td>-3.08</td>
<td>-2.68</td>
<td>-4.21</td>
<td>3.57</td>
<td>-2.50</td>
</tr>
<tr>
<td>Secondary/indexManager</td>
<td>3.05</td>
<td>-2.83</td>
<td>-3.96</td>
<td>-3.66</td>
<td>5.00</td>
<td>-0.50</td>
</tr>
<tr>
<td>CQLTester</td>
<td>3.21</td>
<td>-2.82</td>
<td>-2.16</td>
<td>-4.02</td>
<td>4.00</td>
<td>-2.50</td>
</tr>
<tr>
<td>StressProfile</td>
<td>3.05</td>
<td>-2.77</td>
<td>-4.26</td>
<td>-3.04</td>
<td>5.00</td>
<td>-0.50</td>
</tr>
<tr>
<td>QueryProcessor</td>
<td>3.28</td>
<td>-2.60</td>
<td>-1.72</td>
<td>-2.33</td>
<td>5.00</td>
<td>-4.05</td>
</tr>
<tr>
<td>SystemKeyspace</td>
<td>3.22</td>
<td>-2.50</td>
<td>-1.98</td>
<td>-3.82</td>
<td>5.00</td>
<td>-1.87</td>
</tr>
<tr>
<td>Keyspace</td>
<td>3.16</td>
<td>-2.50</td>
<td>-3.88</td>
<td>-3.28</td>
<td>5.00</td>
<td>0.13</td>
</tr>
<tr>
<td>CompactionManager</td>
<td>3.67</td>
<td>-2.45</td>
<td>-2.66</td>
<td>-4.11</td>
<td>5.00</td>
<td>-0.50</td>
</tr>
<tr>
<td>MessagingService</td>
<td>3.67</td>
<td>-2.42</td>
<td>-1.56</td>
<td>-3.87</td>
<td>5.00</td>
<td>-2.12</td>
</tr>
<tr>
<td>FBUtilities</td>
<td>3.44</td>
<td>-2.35</td>
<td>0.20</td>
<td>-3.40</td>
<td>5.00</td>
<td>-4.69</td>
</tr>
<tr>
<td>Gossiper</td>
<td>3.51</td>
<td>-2.31</td>
<td>-2.42</td>
<td>-3.98</td>
<td>5.00</td>
<td>-0.50</td>
</tr>
</tbody>
</table>

- Gamma scans code to uncover implementation and design issues
- It does intelligent prioritization to surface the most risky code components
- This enables development teams to focus on the right components when fixing issues
Gamma Recommendation Engine (RE)

- In addition to rule-based analysis, Gamma RE includes a discovery capability, to find new issues which are not explicitly implemented as rules.
- Leverages machine learning and the information available in history (code commits and issue databases) to uncover problematic patterns from the past and their possible presence in new code.
- Uses this information to suggest fixes to detected issues.

 Gamma Base Platform
rule-based – e.g. null pointer, memory leak, array access out of bounds, etc.

 Gamma Recommendation Engine
pattern-based – additional issues not found by rule-based analysis
Relationships between software assets

- Commits contain code changes made by developers
- Code changes are made to add features or **fix bugs (relevant here)**
- Code changes may also introduce new bugs
- Bugs are managed in issue management system
- It is not easy / possible to predict if a code change might introduce a new bug before it is committed
- The change may propagate up to QA or even field before it is detected

**The missing link**
Gamma RE

- Gamma RE learns from historical data and matches new code with patterns previously learned
- In addition to Gamma’s Design and Code Quality analysis, it builds a correlation between code changes and Bugs
- Gamma RE achieves this by using deep learning techniques
- A neural network is trained using an encoder/decoder architecture to associate code change patterns with bug descriptions
- This is used to predict if a new code change is likely to introduce a bug
Gamma RE processing - Schematic

Problematic code elements transformed to a fix

Commits which fix issues

Commits

Issues / Bugs

Code deltas (AST)

Issue descriptions

Model

Training

Prediction

New code changes

Sequence match

Likely Bug

Recommendation

Time

Commits which fix issues

Code deltas (AST)

Issue descriptions

Model

Training

Prediction

New code changes

Sequence match

Likely Bug

Recommendation

Time
Prediction - Detection and Reporting of Issues

Example: Missing conditional check

- Above picture shows how new code changes are matched against previous signatures for known issues and any likely issues are reported along with suggested fixes.
Example 1 – HTTP Error Handling

- Project: Apache Cloudstack, commit: 6e09529, file: HttpsDirectTemplateDownloader.java

```java
// Call to HTTPS Client
86 #Override
87 public boolean downloadTemplate() {
88     CloseableHttpResponse response;
89     try {
90         response = httpsClient.execute(req);
91     } catch (IOException e) {
92         throw new CloudRuntimeException("Error on HTTPS request: " + e.getMessage());
93     }
94     return consumeResponse(response);
95 }

// Similar code match (commit: 09e27fd)
res = getHttpClient().execute(req);
s_logger.info("ssp api call: " + req + " status=" + res.getStatusLine());
} catch (IOException e) {
    s_logger.error("ssp api call failed: " + req, e);
}

// Fix pattern (add error handling with proper HTTP Response)
try {
    content = getHttpClient().execute(req, new BasicResponseHandler());
    s_logger.info("ssp api call: " + req);
} catch (HttpResponseException e) {
    s_logger.info("ssp api call failed: " + req, e);
    if (e.getStatusCode() == HttpStatus.SC_UNAUTHORIZED && login) {
        req.reset();
        content = getHttpClient().execute(req, new BasicResponseHandler());
        s_logger.info("ssp api retry call: " + req);
    }
```
Example 2 – State Machine Handling

- Project: Apache Kafka, commit: 8e4799b, file: KafkaStreams.java

```java
synchronized setStateListener

public void setStateListener(final KafkaStreams.StateListener listener) {
    synchronized (stateLock) {
        if (state == State.CREATED) {
            stateListener = listener;
        } else {
            throw new IllegalStateException("Can only set StateListener in CREATED state. Current state is: " + state);
        }
    }
}
```

Similar code match (commit: 343a8ef)

```java
synchronized (stateLock) {
    if (state == State.CREATED) {
        state = State.RUNNING;
    } else {
        throw new IllegalStateException("Cannot start again.");
    }
}
```

Fix pattern: use setState instead of directly changing state

```java
try {
    if (setState(RUNNING)) {
        return;
    }
} catch (StreamsException e) {
    // do nothing, will throw
}
```
Example 3 – Exception Handling

- Project: Apache Kafka, commit: 460e46c, file: KafkaConsumer.java

```java
IllegalArgumentException thrown

    for (TopicPartition tp : partitions) {
        String topic = (tp != null) ? tp.topic() : null;
        if (topic == null || topic.trim().isEmpty())
            throw new IllegalArgumentException("Topic partitions to assign to cannot have null or empty topic");
    }
```

- Similar code match (commit: 842da7b)

  ```java
  if (taskId == null) {
      throw new IllegalArgumentException("Configuration does not contain the property mapred.task.id");
  }
  ```

- Fix pattern: Use base KafkaException for better handling

  ```java
  if (taskId == null) {
      throw new KafkaException("Configuration does not contain the property mapred.task.id");
  }
  ```
Example 3 – Exception Handling

- Project: Apache Kafka, commit: 460e46c, file: KafkaConsumer.java

IllegalArgumentException thrown

```
for (TopicPartition
    String topic =
    if (topic == 
        throw new
)
```

- Similar code match (commit: 842da7b)

```
if (taskId == null) {
    throw new IllegalArgumentException("Configutaion does not contain the prop
mapred.task.id");
}
```

- Kafka / KAFKA-350

Enable message replication in the presence of controlled failures

Details

- Type: Bug
- Priority: Major
- Affects Version/s: 0.8.0
- Component/s: None
- Labels: None
- Status: RESOLVED
- Resolution: Fixed
- Fix Version/s: None

- getMaxErrorValue - seems to be recomputed for each TopicMetadata. Let's get rid of this, I don't think we need it. We already have an unknown entry in the mapping, we should use that and get rid of the Utilities.getShortRange
- If we do want to keep it, fix the name
- **We should really give a base class KafkaException to all exceptions so the client can handle them more easily**
- Instead of having the client get an IllegalArgumentException we should just throw new KafkaException("Unknown server error (error code " + code + ")")
- The file NoLeaderForPartitionException seems to be empty now, I think you meant to delete it
Example 4 – Concurrency

- Project: Apache ActiveMQ, commit: 2a815c2, file: DefaultJDBCApapter.java

```java
public void doDeleteSubscription(TransactionContext c, ActiveMQDestination destination, String clientId,
    String subscriptionName) throws SQLException, IOException {
    PreparedStatement s = null;
    try {
        s = c.getConnection().prepareStatement(this.statements.getDeleteSubscriptionStatement());
        s.setString(1, destination.getQualifiedName());
        s.setString(2, clientId);
        s.setString(3, subscriptionName);
        s.executeUpdate();
    } finally {
        close(s);
    }
}
```

Similar code match (commit: 06cbebc)

```java
s.close();
s = c.getConnection().prepareStatement(this.statements.getRemoveSubscriptionStatement());
s.setString(1, destinationName.getQualifiedName());
s.executeUpdate();
} finally {
    close(s);
}
```

Fix pattern: Missing call to release lock

```java
s.close();
s = c.getConnection().prepareStatement(this.statements.getRemoveSubscriptionStatement());
s.setString(1, destinationName.getQualifiedName());
s.executeUpdate();
} finally {
    cleanupExclusiveLock.readLock().unlock();
    close(s);
}
```
Outcome

• This approach is able to identify important issues which are otherwise not detected through traditional tools without writing new rules:
  – Missing conditional checks
  – Alternative API recommendations
  – Missing function calls (e.g. DB connection is opened but not closed)
• Since it is based on scanning commits and defects, it gets better with more commits as software evolves over time
• It also gets better with more software systems passed through it as it sees more similarities and variations between code elements which introduce/fix defects
Benefits

- Enhanced issue detection
  - Finds more issues than purely rule-based checks, and which have a direct correlation with defects
- Early feedback
  - Helps identify potential bugs before the code is committed to the repository
- Suggestions
  - Suggests fixes / alternatives to detected issues based on historical information
  - Feedback delivered through IDE plugins (e.g. Eclipse)
Thank You

Sudarshan Bhide
Co-Founder and CTO, Acellere
sudarshan.bhide@acellere.com