Analysis of software architecture quality metrics

- Motivation
- Goals
- Related work
- Approaches
- Evaluation
- Conclusion
Motivation

- A good software architecture can make software development successful

- There exist many metrics for measuring the quality of the static view of the software architecture, while neglecting the behavior view

- Measuring the quality of the behavior view of the software architecture is important because the behavior is the one actually supporting the execution

- Whether the results measured in behavior view are different with those measured in static view
Motivation

- Software architecture evaluation tool
  - Sonargraph Architect is a tool which applies some metrics to measure the quality of the architecture
- Behavior is not always considered in current approaches

- Behavior view analysis VS. Static analysis

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Behavior View (runtime)</th>
<th>Static view (code)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inheritance</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Dependencies</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Interface Implementation</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Execution Frequency</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Execution Time</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Motivation

- Considering the behavior view to measure the quality of architecture, we can
  - Know more details about the architecture at runtime
    - Method execution frequency
    - Time consumed
  - Know the difference of the results achieved from static view and behavior view
  - Combine the two measurement ways to make the evaluation more accurate
Goal

- Thesis goals
  - Find or adapt existing metrics which support the measurement of software architecture behavior quality
  - If there are no such metrics, metric proposals should be made and evaluated

- Tasks
  - Investigate existing software architecture metrics
  - Draw a conclusion about the existence of such metrics
  - Choose, adjust or develop relevant metrics to measure the quality of software architecture behavior
Related work

- Metrics categories
  - Static metrics
  - Dynamic metrics
- Each category includes sub-categories
  - Coupling metrics
  - Cohesion metrics
  - Complexity metrics
Related work

- Static metrics
  - Chidamber & Kemere metrics suit (C&K metrics [1])
    - Measure static view
    - Adapted by many other metrics

- Component metrics
  - Most of these metrics focus on coupling measurement in the static view [2],[3]
  - Robert C.Martin’s metrics suit [4] can be adapted to measure component design
Related work

- Dynamic metrics [5]
  - Mostly Adapted from static metrics
  - Analyse data during runtime
  - Not consider the execution frequency and time consumed
  - Dynamic cohesion metrics focus mainly on class cohesion
Issues

- Problems with the existing metrics
  - Important questions are still not solved
    - Are the components well designed (coupling, cohesion, …)? (Q1)
    - What classes or components should be redesigned? (Q2)
  - What violations should I address first? (Q3)
Approach

- **Component metrics**
  - Three metrics proposals

- **Execution Hot spot metrics**
  - Two metrics proposals

- **Violation metrics**
  - Two metrics proposals

Presented in the following slides, including metrics threshold

Not changed since the intermediate presentation
Component metrics

- Aims to solve the question 1
  “Are the components well designed (coupling, cohesion, …)? (Q1)”

Coupling & Cohesion

- Coupling and cohesion are normally interdependent
- Two important indicators for architecture design
- They affect the software architecture flexibility, maintainability and so on.
- There are some metrics for measuring component coupling & cohesion, but few metrics for measuring these at runtime.
**Approach-Component metrics**

- Existing coupling and cohesion metrics
  - Component metrics VS. Class metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Entity</th>
<th>Component</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling</td>
<td>The number of relations between the target component and other components</td>
<td>The number of relations between the target class and other classes</td>
<td></td>
</tr>
<tr>
<td>Cohesion</td>
<td>Based on the classes relationships in the target component</td>
<td>Based on the instance variables used by the methods in the target class</td>
<td></td>
</tr>
</tbody>
</table>

- Find a way to measure the quality of a component by combining coupling and cohesion metrics
Component design analysis

Strong coupling
Low cohesion

Loose coupling
High cohesion

What is the difference?
Approach - Component metrics

- Important factors for component design
  - Classes dependencies/relationships in this component (component cohesion)
  - Classes dependencies/relationships among components (component coupling)
  - Execution frequency (measured in behavior view)

- Types of cohesion
  - Structural cohesion
  - Behavioral cohesion
Approach - Component metrics

- New component metrics (Focus on cohesion)
  - Simple Component Cohesion Metric (SCCM) (Structural cohesion)
    - Formula: \( SCCM = \| Q^R \| / (\| Q^R \| + \| P^R \|) \)
    - \( Q^R \) -> number of class pairs having direct relations
    - \( P^R \) -> number of class pairs having no direct relations
  - Hybrid Component Cohesion Metric (HCM) (Behavioral cohesion)
    - Formula: \( HCM = \| Q^{IR} \| / (\| Q^{IR} \| + \| O^{IR} \|) \)
    - \( Q^{IR} \) -> total number of direct relationships
    - \( O^{IR} \) -> total number of component couplings
  - Call-Weighted hybrid component cohesion metric (CW-HCM) (Behavioral cohesion)
    - Formula: \( CW - HCM = \| Q^{nR} \| / (\| Q^{nR} \| + \| O^{nR} \|) \)
    - \( Q^{nR} \) -> total number of direct relationship execution frequency
    - \( O^{nR} \) -> total number of coupling execution frequency
Example (For component A)

\[ P^R = \{(\text{Class}1, \text{Class}4), (\text{Class}2, \text{Class}3), (\text{Class}3, \text{Class}4)\}, \ |P^R| = 3 \]

\[ Q^R = \{(\text{Class}1, \text{Class}3), (\text{Class}1, \text{Class}2), (\text{Class}2, \text{Class}4)\}, \ |Q^R| = 3 \]

\[ |Q^{IR}| = 2 + 1 + 1 = 4 \quad \text{(getRules(), getLayer(), Priority(), ArchitectureRuleSet())} \]

\[ |Q^{NR}| = 50 + 4 + 5 + 15 = 74 \]

\[ |O^{IR}| = 2 + 1 = 3 \quad \text{(getCallerCallee(), getCallee(), getCaller())} \]

\[ |O^{NR}| = 100 + 20 + 20 = 140 \]
Approach - Component metrics

- **Threshold**
  - **Simple Component Cohesion Metric (SCCM)**
    - Consider the threshold of McCabe Cyclomatic Complexity
    - CC = e-n+2p
    - Assume a component with N classes, we have $|Q^R| = e$, N=n, and P=1. And $CC = |Q^R| - N + 2$
    - $|Q^R| + |P^R| \geq C_N^2$ (which means any two classes have a relation), so we have $SCCM \leq |Q^R| / C_N^2$
### Approach - Component metrics

- **Threshold**
  - **Simple Component Cohesion Metric (SCCM)**
    - Consider that the smallest number of relations in a component is \( N - 1 \).
    - The bottom threshold is
      \[
      \frac{(N - 1)}{C_N^2} = \frac{2}{N}
      \]
  - **Threshold tables**
    | Level                                                   | SCCM \((N \geq 6)\)                                                                 |
    |---------------------------------------------------------|-------------------------------------------------------------------------------------|
    | Good: Simple module with low complexity and good cohesion | \( \frac{2}{N} \leq SCCM \leq \frac{16+2N}{N(N-1)} \)                             |
    | Acceptable: Good cohesion, but a little complex          | \( \frac{16+2N}{N(N-1)} \leq SCCM \leq \frac{36+2N}{N(N-1)} \)                  |
    | Bad: Very low cohesion and low complexity                | \( SCCM \leq \frac{2}{N} \)                                                     |
    | Bad: High cohesion, but too complex                      | \( SCCM \geq \frac{36+2N}{N(N-1)} \)                                              |

- **Level** | **SCCM** \((0 < N < 6)\) |
  - Good       | \((0.66, 1)\)            |
  - Acceptable | \([0.5, 0.66]\)           |
  - Bad        | \([0, 0.5]\)             |
Approach - Component metrics

- **Threshold**
  - **Hybrid Component Cohesion Metric (HCM)**
    - Assume that $|Q^R| \leq |O^R|$ is the acceptable case, we get $HCM=0.5$.
    - Assume that $|Q^R| = 2|O^R|$ is the good situation, $HCM=0.66$.

- **Call Weighted-Hybrid Component Cohesion Metric (CW-HCM)**
  - Assume that $|Q^{nR}| \leq |O^{nR}|$ is the acceptable case, we get $CW-HCM=0.5$.
  - Assume that $|Q^{nR}| = 2|O^{nR}|$ is the good situation, $CW-HCM=0.66$.

<table>
<thead>
<tr>
<th>Level</th>
<th>HCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>[0.66, 1)</td>
</tr>
<tr>
<td>Acceptable</td>
<td>[0.5, 0.66)</td>
</tr>
<tr>
<td>Bad (Low cohesion)</td>
<td>[0.0, 0.5)</td>
</tr>
<tr>
<td>Bad (The Component is useless)</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level</th>
<th>CW-HCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>[0.66, 1)</td>
</tr>
<tr>
<td>Acceptable</td>
<td>[0.5, 0.66)</td>
</tr>
<tr>
<td>Bad (Low cohesion)</td>
<td>[0.0, 0.5)</td>
</tr>
<tr>
<td>Bad (The Component is useless)</td>
<td>1</td>
</tr>
</tbody>
</table>
Approach - Component metrics

- Threshold integration
  - SCCM and HCM
    - Combine structural cohesion and behavioral cohesion

- Final result

<table>
<thead>
<tr>
<th>SCCM-HCM</th>
<th>SCCM</th>
<th>HCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Acceptable</td>
<td>Acceptable</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Acceptable</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Bad</td>
<td>Acceptable</td>
<td>Bad</td>
</tr>
<tr>
<td>Bad</td>
<td>Acceptable</td>
<td>Acceptable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Final result</th>
<th>SCCM-HCM</th>
<th>CW-HCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Good</td>
<td>Good</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Acceptable</td>
<td>Acceptable</td>
<td>Good</td>
</tr>
<tr>
<td>Acceptable</td>
<td>Good</td>
<td>Bad</td>
</tr>
<tr>
<td>Bad</td>
<td>Acceptable</td>
<td>Bad</td>
</tr>
<tr>
<td>Bad</td>
<td>Acceptable</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Very Bad</td>
<td>Bad</td>
<td>Bad</td>
</tr>
</tbody>
</table>
Execution Hot Spot metrics
- Aims to solve the question 2
  What classes or components should be redesigned?(Q2)

Definition of Execution Hot Spot
- A method which was invoked frequently and/or is very time consuming is a method Hot spot
- A class whose methods were invoked frequently and/or lead to big delays in a component or between components is a Class Hot Spot
Execution Hot Spot metrics
- Method Hot Spot
  - Dependency Hot Spot between two components (D-Hot Spot)
  - Method Hot Spot in a component (M-Hot Spot)
- Class Hot Spot
  - Class Hot Spot in a component
  - Class Hot Spot between two components

Difference with Profilers Hot Spot
- Jprofiler’s Hot Spots are CPU Hot Spots and Memory Hot Spots. They focus on performance.
- Execution Hot Spots can be used to find the methods or classes which increase the inter-component coupling.
Communication Integrity Checker (CIC)
- A Java application
- A tool for checking communication violations at runtime

JHotDraw 7
- A famous drawing framework which has been previously used to evaluate new metrics (see APSEC 2013).
- It has a highly modular decomposition -> High structural cohesion
- Measuring JHotDraw 7 aims to validate whether the threshold of component cohesion metrics are suitable or not.
Tools
- Kieker
- Communication Integrity Checker (CIC)
- Self-developed program for filtering and analyzing data

Evaluation work flow
**Evaluation-CIC**

- **Communication Integrity Check**
  - **Data**
    - Collected by running CIC to check itself.
    - Occasionally too many inter-component method calls -> high coupling!
  - **Component metrics results**
    - SCCM values are not bad.
    - HCM and CW-HCM values are bad
  - **Final results are bad or very bad.**

---

### Component Metrics Results

<table>
<thead>
<tr>
<th>Package</th>
<th>SCCM</th>
<th>HCM</th>
<th>CW-HCM</th>
<th>Final results</th>
</tr>
</thead>
<tbody>
<tr>
<td>marshalling</td>
<td>0.67 Good</td>
<td>0.161 Bad</td>
<td>0.29 Bad</td>
<td>Bad</td>
</tr>
<tr>
<td>Network</td>
<td>0.50 Acceptable</td>
<td>0.286 Bad</td>
<td>0.44 Bad</td>
<td>Very bad</td>
</tr>
<tr>
<td>statistics</td>
<td>0.67 Good</td>
<td>0.333 Bad</td>
<td>0.40 Bad</td>
<td>Bad</td>
</tr>
<tr>
<td>rules</td>
<td>0.43 Acceptable</td>
<td>0.250 Bad</td>
<td>0.16 Bad</td>
<td>Very bad</td>
</tr>
<tr>
<td>architecture</td>
<td>0.29 Acceptable</td>
<td>0.367 Bad</td>
<td>0.55 Acceptable</td>
<td>Bad</td>
</tr>
<tr>
<td>monitoring</td>
<td>0.36 Acceptable</td>
<td>0.279 Bad</td>
<td>0.54 Acceptable</td>
<td>Bad</td>
</tr>
<tr>
<td>validation</td>
<td>0.24 Acceptable</td>
<td>0.426 Bad</td>
<td>0.15 Bad</td>
<td>Very bad</td>
</tr>
</tbody>
</table>
Communication Integrity Checker

- Analysis by Sonargraph-Architect
  - Robert C. Martin’s Metric --- Instability (I)
    - validation --- 0.75
    - marshalling --- 0.69

That means these two packages are very unstable. (Depend much on other packages)

- Dependency metric
  - 55 outgoing dependencies from validation to rules (vs. 19 methods calls and 1828 execution frequency observed by us)
  - 12 outgoing dependencies from marshalling to monitoring (vs. 12 methods calls and 608 execution frequency observed by us)

That means that (validation; rules) and (marshalling; monitoring) are highly coupled.
Evaluation-CIC

- Communication Integrity Checker
  - Analysis by using Execution Hot Spot metrics

### validation->rules (D-Hot Spots)

<table>
<thead>
<tr>
<th>Caller</th>
<th>Callee</th>
<th>Method</th>
<th>#</th>
<th>Percentage in caller and callee</th>
</tr>
</thead>
<tbody>
<tr>
<td>validation</td>
<td>rules</td>
<td>getCaller()</td>
<td>576</td>
<td>22%, 29%</td>
</tr>
<tr>
<td>validation</td>
<td>rules</td>
<td>getCallee()</td>
<td>576</td>
<td>22%, 29%</td>
</tr>
<tr>
<td>validation</td>
<td>rules</td>
<td>getOperator()</td>
<td>304</td>
<td>11.5%, 15.3%</td>
</tr>
<tr>
<td>validation</td>
<td>rules</td>
<td>getPriority()</td>
<td>204</td>
<td>7.7%, 10.3%</td>
</tr>
</tbody>
</table>

### marshalling->monitoring (D-Hot Spots)

<table>
<thead>
<tr>
<th>Caller</th>
<th>Callee</th>
<th>Method</th>
<th>#</th>
<th>Percentage in caller and callee</th>
</tr>
</thead>
<tbody>
<tr>
<td>marshalling</td>
<td>monitoring</td>
<td>getTarget()</td>
<td>65</td>
<td>9.2%, 3.2%</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>marshalling</td>
<td>monitoring</td>
<td>getEnd()</td>
<td>65</td>
<td>9.2%, 3.2%</td>
</tr>
<tr>
<td>marshalling</td>
<td>monitoring</td>
<td>getType()</td>
<td>46</td>
<td>6.5%, 2.3%</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>marshalling</td>
<td>monitoring</td>
<td>getValidationState()</td>
<td>46</td>
<td>6.5%, 2.3%</td>
</tr>
</tbody>
</table>
**Evaluation-CIC**

- **Optimization**
  - Create façade classes in the rules and monitoring packages
    - 1. A simple façade class containing all problematic methods
    - 2. Based on 1, optimize the methods in the façade class by aggregating methods which were called at the same time
  - Combine the packages as a new component
    - rules and validation
    - marshalling and monitoring

- **Conclusion**
  - Designing façade classes in a component can improve this component’s cohesion.
  - Merging different small but related components into a new component is a good way to indentify a cohesive component.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Package</th>
<th>Final result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>rules</td>
<td>Very bad</td>
</tr>
<tr>
<td></td>
<td>validation</td>
<td>Very bad</td>
</tr>
<tr>
<td></td>
<td>marshalling</td>
<td>Bad</td>
</tr>
<tr>
<td></td>
<td>monitoring</td>
<td>Bad</td>
</tr>
<tr>
<td>Façade 1</td>
<td>rules</td>
<td>Acceptable</td>
</tr>
<tr>
<td></td>
<td>validation</td>
<td>Very bad</td>
</tr>
<tr>
<td></td>
<td>marshalling</td>
<td>Bad</td>
</tr>
<tr>
<td></td>
<td>monitoring</td>
<td>Acceptable</td>
</tr>
<tr>
<td>Façade 2</td>
<td>rules</td>
<td>Acceptable</td>
</tr>
<tr>
<td></td>
<td>validation</td>
<td>Very bad</td>
</tr>
<tr>
<td></td>
<td>marshalling</td>
<td>Acceptable</td>
</tr>
<tr>
<td></td>
<td>monitoring</td>
<td>Good</td>
</tr>
<tr>
<td>Merging</td>
<td>rules+validation</td>
<td>Good</td>
</tr>
<tr>
<td></td>
<td>marshalling+monitoring</td>
<td>Good</td>
</tr>
</tbody>
</table>
Evaluation-JHotDraw 7

- JHotDraw 7
  - Collect data based on a simple scenario
    - Run the Draw application
  - Draw the graph
    - A
    - B
  - Closed the application
Evaluation-JHotDraw 7

- **JHotDraw 7**
  - **Data**

- **Results**

Bad does not mean low cohesion, it means high cohesion but complex.

So maybe the threshold for SCCM is not very accurate. If I improve the upper threshold, the final result would become better.
Evaluation-JHotDraw 7

- Results of JProfiler
  - Green -> initialization
  - Red -> after drawing

- Analysis
  - After drawing, most of the new instances belong to component draw -> behavioral cohesion
Evaluation

- **Summary**
  - Component cohesion metrics are **useful** to evaluate the quality of software architecture behavior
    - Structural cohesion
    - Behavioral cohesion
  - Execution Hot Spots **help to find the design problems** which reduce the component cohesion.
    - Method Hot Spot
    - Class Hot Spot
Conclusion

- Component cohesion metrics
  - Give three indicators to evaluate the quality of component design
  - Compared results with those of static analyzers and profiling tools

- Execution Hot Spot metrics
  - Identify the optimization points to help improve the design

- Violation metrics
  - Give a way to identify the violation ranking both in static and behavior view
Further work

- Improve the thresholds of component cohesion metrics
  - Based on industry experience and experts’ opinions
  - Machine Learning

- Better filtering for CIC (e.g., JProfiler)
  - Illustrate the results using graphs (e.g., Sonargraph-Architect)

- Integrate the violation metrics with CIC
  - Calculate the violation metrics results directly within CIC
Thanks


