Accessing a System’s Semantics Using a New Architecture Recovery Method

Daniel Link
Motivation

Software systems grow over time in size and complexity.

- **Where** should maintainers apply their work?
- **Documentation** does not exist or is insufficient.
- **Concerns** (features) of a system?
The set of principal design decisions about the system’s

- Structure
- Behavior
- Interaction
- Non-functional properties
Prescriptive vs Descriptive

- **Prescriptive** architecture captures **design** decisions made prior to system construction / adaptation.

- **Descriptive** architecture describes how the system has been **built** / adapted.
The Road to Hell...
Software Decay - Why?

- **Drift** – introduction of design decisions into a system that are *not encompassed or implied* by its architectural design

- **Erosion** – introduction of design decisions into a system that *violate* its architectural design

- Our architecture is feeling the burn and we’d like to make it great again
Architectural Recovery

• The process of **determining a system’s architecture from its implementation-level artifacts** (Source code, executable files, Java .class files, etc.)

• Output is an **architectural view** (a structured arrangement of a system’s implementation-level artifacts under a set of criteria, or a higher-level representation thereof).
Some Solutions

<table>
<thead>
<tr>
<th>Name</th>
<th>Paradigm</th>
<th>Author(s)</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARC</td>
<td>Concern-based hierarchical clustering</td>
<td>Garcia</td>
<td>2011</td>
</tr>
<tr>
<td>ACDC</td>
<td>Structural pattern-based clustering</td>
<td>Twerps</td>
<td>2000</td>
</tr>
<tr>
<td>Bunch</td>
<td>Hill-climbing for maximizing modularization</td>
<td>Mancoridis</td>
<td>1999</td>
</tr>
<tr>
<td>LIMBO</td>
<td>Probabilistic hierarchical clustering</td>
<td>Andritsos</td>
<td>2005</td>
</tr>
<tr>
<td>WCA</td>
<td>Dependency-based clustering</td>
<td>Magbool</td>
<td>2007</td>
</tr>
<tr>
<td>ZBR</td>
<td>Hierarchical clustering based on textual information</td>
<td>Corazza</td>
<td>2011</td>
</tr>
<tr>
<td>PKG</td>
<td>Package Structure Recovery</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>
The Devil Is in the Details

- Size and modification opportunities

  - Which part of the system is “ours” to modify (frameworks?), and which parts of the system should be ignored or evaluated separately (tests, examples, licenses)?

- Quantity/Quality of information (missing, irrelevant or misleading)

- Tell me something non-trivial I didn’t already know or can’t easily find out by just looking at the system

- Tell me something I can understand and put in words
Experts Needed

• Garbage in, garbage out
• Carefully shape the input manually
• Carefully interpret the output
• \( \Rightarrow \textbf{Useless to outsiders} \)
More Issues

• Different methods exist that give different views, but many are **not automatic**, and none give human-readable concerns.

• Frameworks and external information get **lumped in** with developer-supplied code.
ARC’ll fix it?

• Inspiration: NLP and **topic modeling** in ARC

• Topic model built for **each version** of a system -> varies each time

• Topics may be
  
  • Meaningless / Mixed
  
  • Repetitions
  
  • **Generic** concerns
Resulting Topics

- **Meaningless** pile of words: chukwa record data apach hadoop org extract archiv chunk engin sourc datasourc evt key search demux logger format sequenc

- **Generic** data processing: list size array add util java length integ true equal fals empti ignor call actual ensur move prevent algorithm

- **Repetition**: licens log udf apach evalu org hive distribute | licens dynam ser apach distribut | licens apach link file distribut
ARC Issues

• Computational **resources**

• **Sensitivity** to the slightest source code changes because topic models are rebuilt each time

• Architect/programmer cannot predict **impact of decisions**
NOW IS THE WINTER
OF OUR DISCOUNT TENT
Recovery I Can Believe In

- Natural language processing (NLP) remains interesting, so…
- **Classification** distinguishes between a fixed set of classes
- Automatic tool will classify, generate detailed text output (class affinities) and visualizations
- Global, **curated classes (topics)**, built using word lists from existing vocabularies, language keywords or library methods names
- Topics can be named, **explained** and correctness can be shown
- Optionally, **additional** domain-specific or project-specific topics in addition to general ones, information from use cases and test cases
  
  => Non-experts can get useful output
- Ability to determine **which components are addressing which concerns**
  
  => Where does my modification **belong**?
Effects

- No more meaningless and generic topics

- No more repeated topics (but overlap possible)

- No sensitivity to small changes (adding words by itself will not change which concerns are being addressed where)

- We can explain to architects exactly which concerns are being addressed, and where

- No need for different approaches to looking at individual versions vs. evolution of versions
Contributions

• A concept of architectural recovery with a much better awareness of separation of frameworks and requirements versus pure code analysis.

• New, fully automatic architecture recovery method RELAX (RELiable Architecture eXtraction) that needs no specialized knowledge for input and outputs the semantics of a system.

• Visualization of the results.

• Overall architectural recovery output that can be easily understood and directly applied to the maintenance of the system.
Limitations

- Number of topics/concerns is finite (but can be extended with no limits)

- Same goes for number of words in classes
ARCADE

A workbench for evaluating architectural change and decay
Development Status

- Import data
- Train classifiers with imported data
- Classify one or more versions of a system
- Individual source files are classified
- Initial clustering
ARCADE Demo

- One-minute video
- More live at the tools fair
Cluster Dependencies
Detail view
TODO

- Case studies once tool has settled somewhat (soon).
- Filtering out of frameworks
- Special treatment of different types of information (Test cases, use cases)
Acknowledgements

- Nenad Medvidović
- Joshua Garcia
- Pooyan Behnamghader, Duc Le, Arman Shahbazian
Questions