Chunking: An Empirical Evaluation of Software Architecture (?)

Rachana Koneru  David M. Weiss
Iowa State University
weiss@iastate.edu
rachana.koneru@gmail.com

With participation by Audris Mockus, Jeff St. Clair
I'm panicked about my presentation tomorrow.

Relax. What's the worst thing that could happen?

Well, I could embarrass myself in a career-ending way.

Oh, I didn't think about that one.

It might be so bad that you can't even get a recommendation for a future job.

Then you'd have an emotional meltdown followed by substance abuse, untreated health issues and a lonely death.

And it could all happen because of something as trivial as a typo on one of your slides.

I guess I can add "comforting" to my list of things I'm not good at.
Topics

• What is architecture?
  - Structures
  - Software architecture
    - Important structures

• What’s a chunk?
  - Relationship to architecture

• What data do we need to find chunks?

• What algorithm do we use?

• What are the results?

• What’s next?
What is Software Architecture?

- Literally Hundreds of definitions
  http://www.sei.cmu.edu/architecture/definitions.html

- Architecture is focused on
  - Partitioning the whole into parts
  - Specifying the relations among the parts
  - Satisfying Requirements
    Functional Requirements
      End User Features …
    Other Engineering Requirements
      Performance & Scalability, Reliability & Availability …
The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

"Externally visible” properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

Software Architecture in Practice (2nd edition),
(Bass, Clements, Kazman; Addison-Wesley 2003)
Structure (View)

- A structure is a binary relation
  Set of ordered pairs \{(a,b), (c,d), (b,a), (c,e)\}

- Defining a structure
  Define the set of elements
  \{a,b,c,d,e\}
  Define the relation
  Enumeration
  Rule

- Example: connected graph
  Elements: nodes in the graph: a,b,c,d,e,f,g
  Relation: “is connected to”
  \{(a,b), (a,c), (b,d), (b,f), (f,e), (c,g)\}
Key Software Architectural Structures

• **Module Structure**
  – Explains the decisions used to design the information hiding structure of the architecture, and shows how responsibilities are allocated among the major modules.
  – What changes can be easily made?
    o How stable is my system with respect to change?

• **Uses Structure**
  – Describes the allowed “uses” relationships between modules and limits what other modules the implementer of a module may use.
  – In what order should I implement?

• **Process Structure**
  – Defines the distinct processes in the system; Specifies the module(s) that include the process, synchronization between processes …
  – What will the performance be?
FWS Module Hierarchy
Graphical View

FWS

Device Interface
- Sensor Device
- Transmitter Device

Behavior Hiding
- Message Generation
- Message Format

Software Design Hiding
- Averager
- Data Banker
- Sensor Monitor

Transmitter Device
Chunks
What’s A Chunk?

• A chunk is a set of code that has the property that a change that touches that set of code touches only that set of code.
  – Attempt to identify information hiding modules empirically in existing code
  – Observe the effects that changes have on the sustainability of a system
  – Try to find good models of design, empirically

• Use change data from software repositories to identify chunks
  – Many caveats and constraints: very few perfect chunks
  – Difficult to find good data
Change Hierarchy

Diagram:
- Software release
  - Feature
    - Description
      - Modification request
        - Time, date
          - Delta
            - File, module
              - No. of lines added, deleted
            - Developer
              - Version control system
Change Hierarchy

Software release → Feature

Description → Modification request

Time, date → Delta

Developer

File, module

No. of lines added, deleted

Change management system

Version control system
Change Hierarchy

- Software release
- Feature
  - Description
  - Modification request
    - Time, date
    - Delta
      - File, module
      - No. of lines added, deleted
        - Developer

Change management system

Version control system
Caveats and Constraints
aka “Optimization Criteria”

• Eliminate “one-time” chunks
  – Chunks defined by one change

• Eliminate “super” chunks
  – Chunks that include changes touching 20 files or more
    o Complete system
    o Large subsystems
    o “20” files is based on experimentation with the data

• Consider only single-purpose changes, such as bug fixes

• Percentage coupling: The percentage of MRs in the chunk that touch only files within that chunk.
  – A measure of how tightly coupled the files within a chunk are. The higher the coupling, the better the chunk. A perfect chunk has 100% coupling.
Algorithm for Finding Chunks (1)

1. Randomly pick an MR that touches at least five files, but no more than 20 files, from the entire set of MRs in the dataset obtained from the project’s change data. (Guarantees a minimum chunk size that is large enough to eliminate one-time changes.)

   The set of files touched by this MR form the initial chunk, which is also the current candidate chunk.

2. Find a set of all MRs that touch each of the files in the current candidate chunk.

3. Find a set of all files touched by the MRs in step 3.

4. Add a file from the set of files in step 4 to the initial chunk only if it improves the optimization criterion; this forms the current candidate chunk.

5. Repeat step 5 until all the files in the set of files in step 4 are considered.

The preceding steps ensure that the search space for all file combinations is sequentially explored so that potential chunks are not omitted. This also leaves out false chunks resulting from combining isolated one-time changes, and large chunks resulting from changes that touch all or many files.
Algorithm for Finding Chunks (2)

6. Repeat steps 3 to 6 as long as adding a file improves the optimization criteria.

7. Repeat steps 1 to 7 if none of the files improve the optimization criteria until all MRs in step 3 are considered.

8. Repeat steps 1 through 8 for a specified time interval, after which the best chunk candidate based on the best optimization criteria is generated.

9. Remove the files constituting the chunk as well as all MRs within the chunk generated in step 9 from the initial set of files and MRs considered respectively, and repeat steps 1 to 9 to generate the next independent chunk. This ensures that a file belongs to at most one chunk.

10. Remove the files constituting the chunk as well as all MRs within the chunk generated in step 9 from the initial set of files and MRs considered respectively, and repeat steps 1 to 9 to generate the next independent chunk. This ensures that a file belongs to at most one chunk.
# Application To Three Separate Systems

<table>
<thead>
<tr>
<th></th>
<th>Moodle</th>
<th>Eclipse Europa</th>
<th>Company X</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bug fixes</td>
<td>6016 MRs</td>
<td></td>
<td>9949 MRs</td>
</tr>
<tr>
<td>Type of project</td>
<td>Open source, provides support for course management, more than 10 years old</td>
<td>Open source, provides support for software development, more than 10 years old</td>
<td>Large, long-standing telecommunications system, more than 15 years old</td>
</tr>
<tr>
<td>Language</td>
<td>PHP</td>
<td>Java</td>
<td>C</td>
</tr>
<tr>
<td>MR Data interval</td>
<td>4 years</td>
<td>18 months</td>
<td>6 years</td>
</tr>
<tr>
<td>Number of MRs considered</td>
<td>6016</td>
<td>11,042</td>
<td>9949</td>
</tr>
<tr>
<td>Algorithm run time</td>
<td>8,000 seconds</td>
<td>4,000 seconds</td>
<td>10,000 seconds</td>
</tr>
<tr>
<td>Minimum % Coupling Used</td>
<td>80%</td>
<td>80%</td>
<td>70%</td>
</tr>
</tbody>
</table>

*Weiss MECS 29 Sep 2013*
Moodle Analysis

• Organized the data into 3 datasets
  – Moodle: All data for 4 years
  – Moodle-A: First 2 years
  – Moodle-B: Second 2 years

Percentage Coupling of Top 6 Chunks for All Moodle Datasets
Eclipse Analysis

• Organized the data into 3 datasets
  – Europa: All data for 18 months
  – Europa-A: First 9 months
  – Europa-B: Second 9 months

Percentage Coupling of Top 11 Chunks for All Europa Datasets
Company-X Analysis

• Organized the data into 3 datasets
  – Company-X: All data for 6 years
  – Company-X A: First 3 years
  – Company-X B: Second 3 years

Percentage Coupling of Top 2 Chunks for All Company X Datasets
A Few Observations

- Chunks can generally be mapped to components by identifying the file system directories in which their code resides
  - Eclipse: Very clear mapping of chunks to single directories
    - A consequence of the use of Java?
  - Moodle: One to many mapping of chunks to directories
    - A consequence of the use of PHP?
  - Company-X: Mapping information of chunks to directories not available

- No correlation of chunk size with chunk percentage coupling
- Can occasionally observe merging of chunks and increased stability of chunks (increase in percentage coupling over time)
- No current way to match chunks against as-designed module structure
- Validation is difficult
  - Some manual inspection of a (random) sample of MRs is required
  - Manual inspection of each chunk to ensure that every file has been changed together with at least one other file within the chunk
- Relatively few chunks identified
Percentage Coupling and Size of Chunks for Moodle
What Else Would We Like to See?

• Replication of results
  – Same data set and different data sets

• Evolution of chunks with time
  – When is it time to redesign?
  – Effects of experience
  – Effects of distribution of development
  – Time and effort to make a change

• Improved chunk detection algorithm

• Identification of good designs based on chunk stability