Symphony: View-Driven Software Architecture Reconstruction

Arie van Deursen

Joint work with
Rainer Koschke, Christine Hofmeister
Leon Moonen, Claudio Riva

Software Architecture

1. Vehicle for communication among stakeholders
   – common language and reference
   – decision making
   – training/education of people new to the project

2. Basis for system analysis
   – validate/determine system qualities and functionality
   – determine impact of proposed changes
   – integration/(incremental!) renovation opportunities

3. Reminder for (future) architect
   – capture what was done & why it was done
   – separation of concerns

Architectural Documentation in Practice

• Out of date due to
  – architectural drift
  – architectural erosion
• Unspecific for the task at hand
• Not kept in sync because:
  – insufficient incentives within project, since up-to-date doc is relevant to next project only
  – cascading effect:
    • out of date → not used → not updated → ...

The Reconstruction Ambition

• Keep documentation in sync by reconstructing whatever possible from source code/traces
  – Combine hand-written and recovered information
  – Analyze both for (in)consistencies
• Improve usability of the documentation
  – Different levels of abstraction
  – Different perspectives/views
  – Support for querying and searching
  – On demand view extraction
  – Interactive software exploration

The Symphony Reconstruction Process

Origins
• Hofmeister
  – Siemens
• Koschke
  – Bauhaus toolkit
• Riva
  – Nokia cell phones
• CWI/TUD
  – Cobol renovation

Characteristics
• Tailored towards problem at hand
• View-centered
• Iterative
• Reconstruction state of the art incorporated

Architectural Views

• In a house, there are plans for
  – rooms
  – electrical wiring
  – plumbing
  – ventilation
• Each of these constitutes a “view” of the house.
  – used by different people
  – used to achieve different qualities in the house
  – serves as a description and prescription
• Same holds for software architecture.
  – plans are called architectural structures or views
Views

- A view is a representation
  - set of architectural elements
  - and relations between them
- Not all elements
  - only those of interest to stakeholder
- Based on a viewpoint
  - perspective from which a view was created
- Cover: Gödel, Escher, Bach: An Eternal Golden Braid
  - perspective from which a view was created

More Viewpoints

- Siemens four view model:
  - conceptual, module, execution, code
- SEI viewtypes:
  - Module: decomposition, uses, generalization, layering, ...
  - Component-and-Connector: pipe&filter, shared data, publish subscribe, C/S, p2p, ...
  - Allocation: deployment, work assignment, ...

Symphony Stages

- Reconstruction design
  - analyze problem for which architecture is needed
  - define required (target) viewpoints and their mapping from source code
- Reconstruction execution
  - extract and analyze info, apply mappings & create views

Iterate until problem can be solved

Kruchten’s 4+1 Views

- Logical View
  - Development View
  - Process View
  - Deployment View

Viewpoints in Symphony

- Architecture reconstruction consists of the step by step recovery relevant views
- We adopt standard views where possible
- We recognize that specific problems will require specific views
  - Reconstruction shows need for different views!
  - Build up a library of viewpoints

Symphony: Reconstruction Execution
Problem Elicitation
-- Background --
• Anonymous Supplier:
  – Delivers SystemX (SX)
• Anonymous Customer:
  – Has acquired SX, and commissioned/built
    • various business logic customizations
    • interoperability with over 50 other systems
    • a web-based customized GUI
    • data migration tools
  – SX must go into production within 6 mnts

Reconstruction Challenges
• Multi-language:
  – Cobol, SQL, CICS, C, Java
• Size: > 10^6 lines of code
• Limited time available (2-3 weeks)
• Lack of documentation
• Limited supplier collaboration

Symphony in Practice: An Assessment Example
• SystemX (SX) – 2 mlnc Cobol application
• Customer concerns:
  – data integrity
  – reliability
  – maintainability
• Conduct software risk assessment
• Reconstruct relevant architectural views

Key Concerns
• Customer concerns:
  – data integrity
  – reliability
  – maintainability
• Request for independent opinion
  – Can we safely deploy SX
  – Will maintenance be in time and of sufficient quality

Problem Elicitation Workshop
• Customer system stakeholders
  – Users, maintainers, developers, managers
• Best and worst experience in using SX
  – Required quality attributes
  – Perceived lack of quality
• Findings:
  – Criticisms of “hypothetical” architecture!
  – storage capacity, data migration, integrity & recovery, run time efficiency, heavy copybook usage, single point of failure
**Concept Determination**

- What information do we need to deal with the concerns raised?
- Which architectural concepts are needed?
- What views do they correspond to?
- How can we reconstruct them?
  - Source model
  - Target model
  - Mapping

**Initial Analysis**

- Collect and study available documentation
- Apply automatic documentation generation
  - Standard use of SIG's DocGen
  - Focus on module-viewtype,
    - key metrics, various (call) graphs
- Find evidence for perceived problems
- Identify areas for more specific analysis

**Questionnaires**

- Architectural tradeoff analysis
  - Quality attribute requirements
  - Scenarios
  - Focus on workshop problems
- Source + docs based
  - Explain architectural decisions
  - Explain “strange” issues.

**Architectural Concepts**

- Concepts encountered
  - Programs, entities, layers, templates, copybooks
  - Business rules, system interactions
  - Customizations, configurations
  - Queries, updates, transactions, caching mechanism
  - Communication protocols
- Views selected:
  - module, data, customization, client-server

**The Meta-Model**

- DocGen’s General Cobol meta-model:
  - program calls, copybook usage & replacings, program structure
  - database usage, also via utilities
- System-specific extensions
  - Coding conventions / standard violations:
    - Goto’s, recursion, pointers, literals
  - Naming conventions for layering and customizations

**Example source / target relations**

<table>
<thead>
<tr>
<th>Target</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>layer uses layer</td>
<td>program uses program</td>
</tr>
<tr>
<td>layer contains program</td>
<td>program copies copybook</td>
</tr>
<tr>
<td>layer contains copybook</td>
<td>file conforms-to naming-convention</td>
</tr>
<tr>
<td>table joined-with table</td>
<td>layer prescribes naming-convention</td>
</tr>
<tr>
<td>program C/R/U/D table</td>
<td>program uses-D/U-utility parameter-list</td>
</tr>
<tr>
<td>program enforces integrity-constraint</td>
<td>table has-index column-list</td>
</tr>
<tr>
<td>layer C/R/U/D table</td>
<td>column-list compared-with column-list</td>
</tr>
</tbody>
</table>

Lack of database joins, excessively large table, vertical / horizontal decomposition, metrics.
Recovered Architecture: Data View

- **Supplier objective:**
  - DBMS independence: “Bridge” pattern
- **Customer objective:**
  - Get the best out of my Oracle installation
  - Customer with world class Oracle expertise
- **Naive bridge implementation used:**
  - Data integrity, recovery, optimization facilities of Oracle not exploited.

Recovered Architecture: Software Layers

- **DBMS independence:**
  - achieved via a series of layers
- **Manual code “generation” from templates**
- **Many (many!) copybooks**
- **Extraction:**
  - Layering bypassed in 150 cases
  - Benefits of layering not clear

Recovered Architecture: Module Data Exchange

- **DBMS independence combined with good performance?**
  - In this case: redo DBMS caching
- **All data exchange via dynamic memory**
- **Earlier version documented**
- **Extracted from source:**
  - Horrible impact on the code

Recovered Architecture: Variation Points

- **Base system + customizations:**
  - Configuration by product manager: compose new products from existing parts
- **Customization**
  - Coding required for new parts, business rules, data elements, etc.
  - Conceptual architecture: separate modules
  - Concrete: customization code in many places
  -- no appropriate variation points

Recovered Architecture: Process View

- **Customer’s wishes included not yet supported web interface.**
- Introduced *adapter* server to mediate communication.
  - Why separate server?
  - Reliability? Performance? Scaleability?
  - Why a newly invented protocol?
- **Generate alternative architectures**

Data Gathering (Extraction)

- **Collect the data that is required to recover the selected target views**
- **Input:**
  - Manual: developers, domain knowledge
  - Static: code, files, make files, tests, configuration files, ...
  - Dynamic: test runs, simulations, object creations, method calls, ...
- **In our case:**
  - Standard extractions from DocGen
  - Pattern (regular expression) matching on names
  - Tailor made perl scripts for system-specific extractions
Knowledge Inference (Abstraction)

- Derive the target views from the source views using mapping rules
- Bridge semantic gap
  - Cluster analysis
  - Relational algebra
    - C1 contains MA, MA calls MB, MB contains T C2 → C1 uses C2
  - Type inference, (design) pattern recognition.
- In our case:
  - Standard inferences from DocGen
  - Tailor-made queries in SQL, Java, Perl

Information Interpretation (Presentation)

- Making target views accessible to all stakeholders
- Main outcome: paper report
  - Problem identification
  - Observations, evaluation (risks & benefits)
  - Recommendations
- Evidence:
  - Visualizations of various structures
  - Selected metrics and statistics
  - Inspection summaries
  - Interview records

For Comparison: (Subset of) Meta-Model from Nokia

![Meta-Model Diagram]

The RECONSTRUCTOR Project

- Refine Symphony to specialize it for System Evaluation Purposes
  - Which views?
  - Are there new views we need?
  - What are key abstraction steps?
  - Can we derive objective, quantitative data?
  - Can results be visualized?

Industrial Partners

- LogicCMG, SIG, Nokia, ICT Embedded, Oce, Philips, West Consulting

- Systems, Sources, Evaluation Cases

- Problem Elucication

- Concept Determination

- Data Gathering

- Knowledge Inference

- Information Presentation

- Postdoc TUD
  - View selection and categorization

- PhD 1 TUD
  - Abstraction & Quantification

- PhD 2 TUD
  - Visualization

- Interaction

- Methods & Tools
  - Symphony/Icons, Symphony/Instruments

- Views & Cases
  - Symphony/Views, Symphony/Recordings

- Project Results

Literature


- http://www.cwi.nl/~arie/
- http://www.sig.nl
The Symphony Way

- Involve code analysis in renovation planning
  - Recover actual dependency graph instead of hypothetical one
- How?
  - Describe / elaborate renovation problem
  - Determine related concepts + views
  - Collect data, analyze, and present
- Small investment to reduce the major risks involved in any renovation

In Conclusion

- Reconstruction = design & execution
- Step by step reconstruction of relevant views