Practical Technical Debt Discovery by Matching Patterns in Assessment Graphs
Andriy Shapochka, Principal Architect
• Architectural Assessment Service Leader at SoftServe Inc.
• Email: ashapoch@softserveinc.com
• Skype: ashapochka

Borys Omelayenko, PhD, Application Architect
• Architectural Assessment Service Expert at SoftServe Inc.
• Email: bomel@softserveinc.com
• Skype: borys.omelayenko.softserve
Global Development and Consulting Services

29 offices
8 countries
Technical Debt Assessment

- Business Case Review
- Requirement Elicitation
- Quality Attribute Workshop
- Requirement Prioritization
- Architecture Review
- Implementation Review
- Artifact Collection

Business Goals
- Functionality
- Constraints
- Quality Attribute Scenarios

Solution Architecture
- Code
- Solution Artifacts
- Architectural Decisions

Solution Implementation
- Business Goals
- Solution Architecture
- Solution Implementation

Qualitative Analysis
- Scope Deltas
- Sensitivity Points
- Tradeoffs
- Risks
- Non-Risks

Quantitative Analysis
- Recommendations
- Estimates
- Improvement Roadmap
- Improvement Indicators
Challenges of Technical Debt Assessment

- Diseconomy of scale: how team of 2-3 evaluates x100K LOC within 2-4 weeks
- Flexible, repeatable, automated, **convincing** analysis of whatever artifacts available
- Low level source code technical debt items do not cut it for business stakeholders
Main Idea

1. Technical debt items are traced to antipatterns.
2. Antipatterns are translated to graph constructs.
3. These constructs are looked up in explicit Assessment Graphs with queries.
4. The results are postprocessed and visualized.

The framework is based on the familiar analytical tools and is automated as much as possible.
Assessment

Graph

- Green comes from data sources
- Model is partial
- Neo4j is schema-less, multiple labels are used to type nodes
ETL Process -> Analysis Process

Start ETL

Obtain access to project artifacts

Source Code Repository

Documentation

Extract notions, terms, high level architecture, etc.

Build commit history subgraph

Build code structure subgraph

Build issue subgraph

Issue Backlog

Neo4j Assessment Graph Database

Complete ETL

softserve experience matters
ETL Process -> Analysis Process
Pattern Translation into Query

1. Determine what conceptual entities and relationships between them form the given pattern.
2. Map them to nodes, relationships, labels, and properties in the assessment graph.
3. If graph structures are insufficient massage the graph data adding more nodes, labels, properties, and relationships.
4. Create MATCH queries on the graph to search for the pattern.
5. Inspect and verify query outputs. If needed repeat preceding steps to build sufficiently accurate queries.
Patterns in Scope

- Code structure, external dependencies, involving metrics such as complexity, OOP, and similar (demonstrated in the paper).
- Team collaboration (social) and historical code change patterns fossilized in commit history and issue history.
- Higher level architectural and business domain patterns reflected in system decomposition, semantic naming, comments, and documentation (typically would require NLP).
- System evolutionary patterns using extrapolation techniques applied to data captured in the assessment graph.
Component View

- More data sources can be added
- Deployment containerized for easy portable deployment
- Implementation languages: Python 3, Neo4j Cypher
- Jupyter (IPython) notebooks used as analysis platform
iDempiere Query Sample (see our paper for details)

MATCH
(file:JavaFile)-[:DEFINES]->
(zk_coupled_class:JavaClass)
-[:COUPLES]->
(zk_class:JavaClass)
WHERE zk_class.name = "(?i)'org\\zkoos.*'"
WITH file, zk_coupled_class, zk_couple
MATCH
(file:JavaFile)-[:DEFINES]->
(db_coupled_class:javaClass)
-[:COUPLES]->
(db_class:JavaClass)
WHERE db_class.name = "(?i)'java\\sql.*'"
WITH DISTINCT file, db_couple, zk_couple
RETURN
LAST(SPLIT(file.name, '/')) AS file_name,
file.count_line_code AS loc,
COUNT(db_couple) AS db,
COUNT(zk_couple) AS zk
ORDER BY
db_couple_count DESC,
zk_couple_count DESC,
loc DESC

<table>
<thead>
<tr>
<th>file_name</th>
<th>loc</th>
<th>db</th>
<th>zk</th>
</tr>
</thead>
<tbody>
<tr>
<td>InfoPanel.java</td>
<td>1502</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>FindWindow.java</td>
<td>1813</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>DocumentSearchController.java</td>
<td>328</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>InfoWindow.java</td>
<td>1594</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>InfoProductPanel.java</td>
<td>1237</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>... (51 more)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
MATCH (a1:Actor)-->(p1:Package)<--(p2:Package)<--(a2:Actor)
WHERE NOT "Travis" IN [a1.name, a2.name]
WITH DISTINCT p1.longname AS package,
    p1.metric_CountDeclMethodPublic AS public_method_count,
    p1.metric_SumCyclomaticStrict AS total_cyclomatic_strict,
    COLLECT(DISTINCT a1.name) AS developers,
    COLLECT(DISTINCT a2.name) AS users,
    COLLECT(DISTINCT p2.longname) AS dependent_packages
RETURN package,
    public_method_count,
    total_cyclomatic_strict,
    SIZE(developers) AS developer_count,
    SIZE(FILTER(user IN users WHERE NOT user IN developers)) AS user_count,
    SIZE(dependent_packages) AS dependency_count
Expected Benefits

- Graph database structures accurately and efficiently capture complex software projects.
- They are open to any type of graph extension and enrichment (new types of subgraphs, explicit relationships, properties, and labels).
- Search for technical debt item patterns directly translates into Cypher queries.
- ETL and analysis steps can be partially or fully automated and applied across projects, integrated with continuous delivery process, shared among teams, and packaged as reusable assets.

We regard our work as a step toward the concept of **Technical Debt Evaluation as Code** as useful for Software Evaluation, Monitoring, and Improvement as **Infrastructure as Code** is for DevOps and Software Development.
Next Steps

- Try to apply NLP techniques to comments, code entity (file, class, method) names, and issue reports.
- Work on identification and implementation of the patterns significant for technical debt discovery and estimation (also possibly laying the bridge to CBAM).
- Catalog best practices of building and analyzing assessment graphs.
Prior Art

- "Expressive and Scalable Source Code Queries with Graph Databases" by Raoul-Gabriel Urma and Alan Mycroft, University of Cambridge, pub. 2015
- "Your Code as a Crime Scene" by Adam Tornhill, pub. 2015
Thank you!

USA HQ
Toll Free: 866-687-3588
Tel: +1-512-516-8880

Ukraine HQ
Tel: +380-32-240-9090

Bulgaria
Tel: +359-2-902-3760

Germany
Tel: +49-69-2602-5857

Netherlands
Tel: +31-20-262-33-23

Poland
Tel: +48-71-382-2800

UK
Tel: +44-207-544-8414

EMAIL
info@softserveinc.com

WEBSITE:
www.softserveinc.com