Renaissance: Tools for Large-Scale Code Refactoring

Andrei Rădulescu
4-Oct-2017
Introduction to Thermo Fisher Scientific

- Multi-national company
- Precision laboratory equipment and services
- 65000 employees worldwide
- >$18 billion revenue

- Via the 2016 acquisition of FEI Company,
  >65 years of experience in electron microscopy
  - Main markets
    - Science: Material science, Life science
    - Industry: Electronics, Oil & Gas
  - R&D activities
    - Hillsboro, US
    - Eindhoven, The Netherlands
    - Brno, Czech Republic
    - Bordeaux, France
Thermo Fischer Scientific – Eindhoven

- High-end transmission electron microscopes (TEM)
- R&D, production, service, sales, marketing, etc.

- Multi disciplinary R&D
  - Physics
  - Electronics
  - Mechatronics
  - Software

- R&D Eindhoven
  - ~250 engineers
  - ~110 software engineers including outsourcing
Introduction to TEM

Key figures
- HT: 300 kV
- Vacuum
  - Drift < 5 Å/min (3.8M years/km)
    - Noise (Acoustic)
    - Vibrations / Pressure
    - Stray fields (EMI)
    - Temperature
- Objective Lens: 11 A ± 20 μA
- Magnification: > 1 Mx
  - “See a tennis ball on the moon”
  - Atomic resolution: < 1 Å
- Detection
  - Imaging (TEM)
  - Scanned imaging (STEM)
  - X-ray (EDX)
  - Energy loss (EELS)
**TEM Software Landscape**

**Applications**
Provides general purpose UI or tailored for specific use-case

**TEM server**
offers unified API to control all microscope configurations

**Embedded platforms**
Real-time control of microscope and processing of detector data

**Hardware**
Electron gun, optics, vacuum, detectors, etc.

C# (WPF), C++ (Qt or MFC)
x100k LOC / application

C++ & COM
x1M LOC

C++, ThreadX, Linux, FPGA
x100k LOC / platform

x100 hardware configurations, including specials
Need for Large-Scale Refactoring
• Complex software code base
  • Reflects the complexity of the system
  • Has evolved over tens of years, inevitably accumulating **technical debt**
  • **Increasing costs** over time for both adding features and maintenance

• Addressing technical debt
  • Manual refactoring or rewriting software → hard to justify to business
  • Tools can increase the speed and efficiency of large-scale refactoring of legacy code
TEM Server has grown over years into a complex code base of ~6000 KLOC
- One huge component: 3300 kLOC \(\rightarrow\) large build / smoke time, multiple stream delivery / rebase overhead
- Large / complex dependency structure \(\rightarrow\) large integration overhead
- COM overused + issues \(\rightarrow\) large overhead to change / propagate interfaces; multiple interface registration
- End-user applications, service tools use internal interfaces \(\rightarrow\) lifecycle issues, system safety is bypassed
  - etc

We must transform the TEM Server code to enable a more efficient code development
- Decompose TEM into smaller components, simplify the dependency structure between and within components
- Each module should have a well-defined and tested interface
- Solve COM issues, reduce COM usage
- Port applications / tools to external interfaces
  - etc

Such large-scale transformations must be efficient, consistent and repeatable
- Use model-based refactoring tools \(\rightarrow\) Renaissance
Renaissance
About the Renaissance Project

- Public / Private funded project lasting 2 years (September 2016 – August 2018)

- Partners
  - Thermo Fisher Scientific: Lead industrial partner
  - TNO-ESI: Applied research partner
  - Cornerstone BV: Software quality measurement & analysis specialists

- Goal: Tools to analyze, visualize and refactor a large C++ / C# / COM code base

- Approach
  - Semi-automated
    - Most repetitive tasks can be easily automated; remaining one-offs are done manually
  - Iterative
    - Lower risks (no “big bang”)
    - Visible project progress
    - Early tool validation (“learn as we go” – e.g. task is repetitive enough to warrant automation)
Analysis and Transformation of Code

- Legacy code (today)
- Improved code (tomorrow)

1. Analyze
2. Extract visual model
3. Restructure
4. Auto-rewrite
5. Validate

[Repeat]
Renaissance #1: COM Cleanup

- Unnecessary COM complexity \(\rightarrow\) requires cleanup
  - Suboptimal design choices due to early adoption
  - Complex interdependency structure = maintenance overhead
  - Cleanup is a prerequisite for SW decomposition

- **Business case**
  - 3x lower refactoring effort
  - First-time right due to code generation
  - Tools reusable in the next project phases

![Refactoring Effort](chart.png)

<table>
<thead>
<tr>
<th>Renaissance</th>
<th>Spent</th>
<th>Manual only</th>
<th>Effort extrapolation from manual interface cleanup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

men/weeks
Visual Analysis Using Neo4j Graph Database + Cypher Queries

- Well-structured projects
- Input: existing code base
- Chaos (entangled code): refactoring required
Renaissance Interface-Grouping Tool + yEd Visualization

- Interface grouping into libraries
- Automatic procedure, based on interface implementation project
- Interface dependencies are also identified
- Interface users
- Interface abusers (include idl i.o. import tlb)
  - Subject to cleanup
• Improve TEM-server **architecture**
  • Make dependencies explicit through interfaces
  • Reduce module coupling through interfaces (and interface versioning)
  • Interface abstractions allow platform strategies around product families
  • Reduce complexity by removing cross-module dependencies
  • Smarter testing (e.g., component-level tests, integration tests based on use cases)

• **Speedup TEM-server build time** 25x (6 hr → 15 min)
  • Break large components → smaller components
  • Break long dependency chains → interface components
  • IncrediBuild, incremental builds

• Decomposition is a complex, significant-effort, tedious activity
  → **Automate and guarantee quality with Renaissance tools**
**Renaissance #2: Decomposition**

- **Renaissance** (building on results of Phase 1)
  - Identify and move module projects to separate components
  - Refactor clients to use the new interface locations

---

Break large components
Renaissance yEd-based Exploration / Decomposition

- Specify new components at project level
- The tool identifies component dependencies
  - Interface implementations
  - Interface usages
  - Circular dependencies are marked red
Conclusions

**SW Mission:** Develop **correct** functionality **fast.** Be able to deliver **high-quality** SW at **any time.**

- **Speed + efficiency**
  - Decouple teams → interfaces + versioning
  - Short build + test feedback cycles (wait time = waste)
  - Early integration (see our last year presentation)
  - Tools to automate repetitive work
  - DSL + tools to increase abstraction

- **Quality**
  - Extensive + smart testing
  - Tools to (formally) verify SW correctness
  - Allow (tool-based) refactoring to avoid building technical debt
  - Tools to measure the code-base health
  - Keep the measured scores up
Thank You