ACCU 2007

Linting Software Architectures

Bernhard Merkle
Central Research & Development
Software-Engineering
SICK-AG Waldkirch, Germany

mailto: Bernhard.Merkle@sick.de
mailto: Bernhard.Merkle@googlemail.com
Some Background, and the plan...

- About...
  - myself
  - SICK

- The plan for this talk
  - Software-Architectures
    - Terms, Definitions, etc

- Checking Architectures
  - Different Kinds of Architecture-Analysis
  - Tools for Architecture-Analysis

- Experiences, Discussion... ;-)

Bernhard Merkle  Linting Software-Architectures  ACCU 2007  page:2
**Linting Software-Architectures**

- Why should we care?
  - In lots of Projects, Architecture decay happens
  - We are not alone, as we've prominent representatives… ;-)
Software-Architecture: Definitions

- **IEEE 1471-2000:**
  - The fundamental *organization* of a system, embodied in its components,
  - their *relationship* to each other and the environment,
  - and the *principles* governing its *design* and *evolution*.

- **Booch, Rumbaugh, Jacobson 1999**
  - ... the set of significant decisions about the organization of a software system ...
  - ... is the highest level of technical design for a software system: It is driven by your key concerns
Views on a Software-Architecture

- 4+1 View Model (Kruchten, 1995)

![Diagram showing the 4+1 View Model](image)
Documenting a Software-Architecture (Kruchten)

- captured in two documents:
  - *Software Architecture Document*
  - *Software Design Guidelines*

  - respected to maintain the architectural integrity of the system.

- Documents are important, but they are Documents (enforce ? ;)

→ Some kind of Automatic Rulechecking
MDSD (Model Driven Software Development)

- Approach:
  - Architectural Design IS IN the model (and Application !?)
  - Executable Model (MDA, UML+CodeGen, UML-VM)
  - Source: Model, Target: Application (→ Forward Engineering)

- Open Items:
  - Reverse-/Roundtrip-Engineering ?
  - UML too general: DSL ? (Meta-Modeling Support)
  - Important Standards (e.g. in MOF, ASL, QVT) ?
  - Manual Extensions of generated Code
  - Good Integration of “legacy Software” ?
Architecture-Analysis

- Lint == STATIC Analysis
  - hence...some limitations if you do things/tricks at runtime
    (e.g. Reflection in Java,...)

- With Tool support
  - Pro: automatic, consistent, rule enforcement
  - Cons: Semantic, external Quality

- The Pro is much stronger compared with Code-Lints !!!
Levels of Static Analysis:

- **Goal:** (on all Levels)
  - find, avoid Problems, Increase QA (and measure it)
- **Micro-Level**
  - Code, MIRSA-C
  - E.g: =, ==, {},
- **Marco-Level**
  - Class-Design, Effective Rules, C++, Java, C#
  - E.g: by reference, String concat, Exception-Handling
- **Architecture-Level:**
  - Layers, Graphs, Subsystems, Components, Interfaces
  - E.g: Coupling, Dependency, etc...
Different kinds of Architecture-Analysis

- Consistency-Analysis
- Rating of Architecture
- Discover a Architecture
- Measure real facts (e.g. metrics)
- Monitoring changes, trends (QA)
Consistency-Analysis

- Aim: No inconsistency
- Dispersion (no toolchain, information loss)
- Declay of Architecture, Rules violated, (over project time, various reasons...)
- Deviation Comparison
Consistency-Analysis

Requirements

• System SHOULD be: maintainable, easy to understand, extensible, independent

Architecture-Design → Should-Architecture

• Model Dependencies
• Model Interfaces
• Model Subsystems

Extraction → Is-Architecture

• Dependencies as exist
• Interfaces / Usage as exist
• Subsystems

Comparison → “Diff.” Architecture

• Violations
• Conformance

Actions

Existing Code

• System IS: ... 😊
Consistency-Analysis: Things become VISIBLE

- Results aggregated the right way: (e.g. Subsystem level)
How to cope with violations...

- Identify violations
  - Where
  - Quantity, Quality
  - Heaviness, Impact

- Handling violations
  - Fix possible? (effort, costs, time)
  - Virtual refactorings, Simulations
  - List with modifications
  - Programmer implements fixes
  - Sometime, “autofix” lint?… ☺
Rating of Architecture

- NO Rating of external Requirements (Fullfillment)
- Internal Quality (is the focus)
  - Cycles
  - Coupling
  - Stability
  - Anti-Patterns, Bad Smells

- Target:
  - Analyze Problem (and fix) (during project)
  - Compare _different_ Architecture solutions ?
Rating of Architecture: e.g. Cycles

- Handling of Subsystems becomes difficult...
Rating of Architecture: e.g. Coupling

- DIP (Dependency Inversion Principle), R. Martin

Diagram:
Before:
- Packg. E
  - Class X
- Packg. F
  - Class Y

After:
- Packg. E
  - Class X
  - Interface IX
- Packg. F
  - Class Y

before

after
Rating of Architecture: e.g. AntiPatterns

- Dependent BaseClass
  - Type: Design Problem

- Problem: one of more Methods shall implement different behavior, depending on the type, passed in
- Context: make “extensible” systems, frameworks
- Forces: Programming languages offer, instanceof/typeid funcs.
- Antipattern: Methods of the baseclass, depend on derived classes, e.g. accessing their members, doing switch/case depending on type information
Rating of Architecture: e.g. How to find AntiPatterns

- Dependent Baseclass: 1.5/1000 in Eclipse 2.1, 16/1000 in JDK 1.4.0
- Multiple Interface Inheritance 4/1000 in Eclipse 2.1, 18/1000 in JDK 1.4.0
Rating of Architecture: e.g. How to find AntiPatterns

- JDK 1.5: ... 1315 classes in 229 packages all depend on each other !!!
- classes.zip, rt.jar (BIG BALL OF MUD ? ;-)

[Diagram of dependency network]
Discover a Architecture (Erosion, prog. understand)

- Visualisation of _existing_ Architecture (Layout !)
  - Architecture often implicit
    - Undocumented
    - new staff in project,
    - Quick Overview of external software
- Erosion and Analysis
  - Discover central abstractions/key concepts, e.g. Worker-classes
  - Typical Usage of certain artefacts, Patterns
- Navigation
  - Used from, Using others,…
  - Library dependency ?
  - High-Level Cross Referencer
Discover a Architecture: Questions

- Is there a Software Architecture?
  - Implicit, explicit
  - Conformance with rules

- Which Architecture Artefacts are there?
  - Interfaces, Packages, Components, Subsystems, Layers
  - Layer-Architecture, Graph-Architecture,…

- Any Violations of the Reference/Target-Architecture?
  - Cycles between xyz…
  - Interface violations between subsystems
  - Bypassing Interfaces
Discover a Architecture: Level of Abstraction

- Topologic sorted layout, only Call-Relationships

• Business Modules
• Public, exported Methods

• Utility Modules, Infrastructure
• Private, internal Methods
Discover vs. Model a Architecture: Variance comparison

- Arch. sorted layout, only Call-Relationships

• You SEE the architectural violations
Discover a Architecture: Level of Abstraction

- Topologic sorted layout, only Inheritance-Relationships

  Sub-Classes... Base-Classes...

- You SEE important Base-Classes....

  - E.g from Core:
    - IWorkspaceRunnable 102x
    - IAdaptable 100x
    - IPlatformObject 50x
Measure real facts (e.g. Metrics)

- Metrics are _indicators_ for
  - Quality, Understandability, Maintenance, Error Probability,…
  - Hard facts, measured numbers

- Examples
  - LOC (lines of code)
  - Cyclomatic complexity
  - ACD (average component dependency)
  - Metrics of Robert C. Martin (abstractness, instability etc.)
  - Inheritance depth, overridden/implemented methods,…
**Measure real facts (e.g. Metrics)**

- Controlled Quantities
  - LOC, #of pakets, files, classes, methods
  - Simple counting of certain artefacts
  - Set a threshold
  - Identify and handle outliers

- Discover candidates which are
  - Sources for bugs, complex, hard to maintain
  - Performance problems
  - Duplicates
Monitoring changes, trends (QA)

- Level Subsystem, Package, File, Class, Operation etc.
  - New artefacts
  - New dependencies
  - New Architecture violations

- Early, betimes correction of violations

- Monitoring
  - Trendreports
  - “outsourcing” projects
Tools for Architecture-Analysis

- Features:
  - Static Analysis → Actual state of Arch
  - Description of Arch Rules → List of violations, deviations
  - Show Dependencies (granularity, number, graph)
  - Simulation of Refactoring, Worklist
  - Metrics
  - Trendanalysis
  - IDE-Integration
  - Web-Report
  - Automation, cmd-line
Tools for Architecture-Analysis

- Products:
  - Sotograph:  www.software-tomography.de
  - Bauhaus:  www.axivion.com
  - SonarJ:  www.hello2morrow.de
  - Structure101:  www.headwaysoftware.com
  - Lattix:  www.lattix.com
  - Klocwork K7:  www.klocwork.com
  - XRadar (opensource):  www.xradar.org
  - Others: CodeCrawler, SeeSoft, ResourceStandardMetrics
Basic Approaches

- Basic approaches
  - Your makesystem...
  - makedepend, jdepend
  - RE code into UML model
  - Eclipse (Java Build Path)
**Sotograph: Overview**

- VERY powerfull
- Infos via Table + Graph
- Cool layout algorithms
- Known since 2003 (NG”SNIFF++”)
- Mysql DB, open schema
- Fat GUI Client, Web Report
- About 200+ Metrics
- Arbitrary User queries
- Trend Analysis
- Virtual Refactoring
- Java, C++, C#, source parser
- Lightweight SotoArch 2007
Sotograph: Source and Architecture

Strict layering violated (optional)

References within a layer (optional)

Illegal references:

architecture_en

Reference: always illegal

Product 1

Product 2

GUI

Interface

Applicatory Logic

Data
- Dependencies: Informations...
● Arch. violations
Sotograph: Monitoring Changes

- of Architecture, Quality, Structure
Axivion Bauhaus Suite

User Interface
Access to the Analyses' Results (Interactive GUI and Reporting)

Analysis
- Architecture Visualization
- Architecture Validation
- Interface Analysis
- Cycles, Dominance
- Metrics, Stylechecks
- Clone Detection
- ...

Fact Base (Graphbased)

Resource Flow Graph

Attributed Abstract Syntaxtree

Source Code
C, C++, Java, Ada, VB, Cobol

Scripting Add-On

User Interface Skripting
Advanced Access, e.g. for automated HTML-Reports

Coarse Grained Scripting
- additional individual analysis facilities
- metrics, stylechecks
- embedding into environment
- ...

Fine Grained Scripting
- individual stylechecks and metrics on the syntactic level
- ...

External Information Sources, Application, Reports etc.
Axivion Bauhaus Suite

Architectural Validation

Clone Detection

Axivion Bauhaus Suite

Metric Analysis

Interface Analysis

Dead Code

Cycles & Dominance
SonarJ: Overview

- Java centric
- Infos via Tables
- No graphs
- Known since 2005
- “In memory DB”
- Good Eclipse-Plugging
- Lightweight approach
Architecture-MetaModel:

- Step 1: Cut horizontally into Layers
- Step 2: Cut vertically into vertical slices by functional aspects
- Step 3: Defines the rules of engagement
SonarJ: Architecture-MetaModel

- Meta model: layers, vertical slices and subsystems
  - Each subsystem belongs to exactly one layer
  - A subsystem also might belong to a vertical slice
  - The association between vertical slices and subsystems is typically implemented by a naming convention
  - Vertical slices do not have to be present on every layer
  - Technical subsystems typically are not associated with any vertical slice
  - Technical systems often do not have vertical slices at all
Structure101: Overview

- Java (C++, Ada planned)
- Infos via DSM + Graphs
- Known since 2005
- Repository/DB server
- Fat-Client, Web
- Lightweight approach
Structure101 Architecture
Structuur101: Architecture Visualization

- Dependency Analyse
Lattix: Overview

- Java, (C++ via BSC, doxygen)
- Infos via DSM
- No graphics (or weak)
- Known since 2004
- “In memory DB”
- Lightweight approach
- Fat client
- Trend via cmd line
  + own report
Lattix: DSM Principle

- Artefacts (e.g. Subsystems, Packages, Types, etc.) are displayed in Matrix
  - Columns show “using-” relations
  - Rows show “is used from-” relations
- Artefacts can be
  - Grouped in Subsystems, Layers
  - Arranged hierarchically
- Architecture State can be read via Matrix
- Partitioning algorithms can identify highly coupled artefacts
- Rules for allowed/forbidden relationships
**Lattix: DSM Examples**

- Example Architectures, for direct reading from Matrix

<table>
<thead>
<tr>
<th>$\text{root}$</th>
<th>$\omega_1$</th>
<th>$\omega_2$</th>
<th>$\omega_3$</th>
<th>$\omega_4$</th>
<th>$\omega_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>2</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>domain</td>
<td>3</td>
<td>17 29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>framework</td>
<td>4</td>
<td>75 53 42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>util</td>
<td>5</td>
<td>10 13 16 13</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{root}$</th>
<th>$\omega_1$</th>
<th>$\omega_2$</th>
<th>$\omega_3$</th>
<th>$\omega_4$</th>
<th>$\omega_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>model</td>
<td>2</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>domain</td>
<td>3</td>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>framework</td>
<td>4</td>
<td>42</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>util</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\text{root}$</th>
<th>$\omega_1$</th>
<th>$\omega_2$</th>
<th>$\omega_3$</th>
<th>$\omega_4$</th>
<th>$\omega_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>application</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>model</td>
<td>2</td>
<td>37</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>domain</td>
<td>3</td>
<td>17 26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>framework</td>
<td>4</td>
<td>75 53 40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>util</td>
<td>5</td>
<td>11 13 16 13</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Klocwork: K7 Overview

- Static Analysis Tool with Architecture addon
  - Inforce, Inspect
  - Insight, Project Central
- Infos via Table + Graph, but WEAK layout algorithms!
- NOT Out-of-the-box, but can be customized via tcl scripts
- Mysql DB
- Fat GUI Client, Web Report
- Java, C++
Klocwork: K7 insight
Klocwork: K7 insight
Tool Comparison

- Target audience
- Languages
- Handling
- Process
- IDE Integration
- Infrastructure
- Lightweight, Powerfull, Compliacated
- Features (that you (will) need)
Take home

- Today's IDEs / mechanisms are not suited for architectural analysis
  - Use a “lint4Architecture” (no official, my term)
- Tool support is a necessary
  - Architecture monitoring (possible with a small weekly time investment)
- Management…can be convinced if existing problems become visible
  - pays off very fast (e.g. one week jdepend analysis vs. Sotograph refactoring done)
- Rules can/will be violated
  - There is always a “good” reason for that
- Rule can be checked
  - Tool support can automate the process
  - If you have continous build system, start employing a “lint4Architecture” now !!!
Informations

- Wikipedia

- Books:
  - Refactoring in Large Software Projects
  - Patterns
  - AntiPatterns
  - Metrics
  - Architecture