ACCU 2009: Stop the Software Architecture Erosion

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

mailto: Bernhard.Merkle@sick.de
Wide product and technology range

- Industrial Sensors
- Advanced Industrial Sensors
- Encoder
- Industrial Safety Systems
- Auto Ident
- Analyzers & Process Instrumentation
DARPA: Urban/Grand Challenge, Google SV
Traffic and Luggage Control
Introduction:

Levels of Static Analysis
- Code, Design, Architectural
- Examples

Architectural Analysis
- Use Cases
- Tool Support,
- Examples
- Pros/Cons

Summary
Possible levels of Static Analysis:

Goal: find, avoid Problems, Increase QA (and measure it)

**Micro-Level**
- Code
  - e.g: =, ==, { },

**Macro-Level**
- Class-Design
  - e.g: by reference, String concat, Exception-Handling

**Architecture-Level:**
- Layers, Graphs, Subsystems, Components, Interfaces
  - e.g: Coupling, Dependency, etc…

Tool-support for each level…
Micro-Level / Code:

Language itself
- Appendix F/ANSI or G/ISO
  - Unspecified behaviour
  - Undefined behaviour
  - Implementation-defined behaviour
  - Locale-specific behaviour

Usage of Language
- Misplacement
- Omission and Addition
- Unexpected behaviour
- Complexity
Macro-Level: Design + Coding Guidelines
Field should be package protected

A mutable static field could be changed by malicious code or by accident. The field could be made package protected to avoid this vulnerability.
Software, Architecture, Erosion
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.

Kruchten: captured in two documents:
- *Software Architecture Document*
- *Software Design Guidelines*
Architecture Erosion ALWAYS happens

- **Time pressure** leads to abbreviations, which are not replaced later by clean solutions
- **Prototypes** become products
- **Hacks**. Architectural restrictions are ignored or worked around
- **Lack of understanding the “should architecture”** leading to extensions and modification which obey only micro- and macro architecture. (e.g. outsourcing)

- It is **impossible to detect manually** architectural violations in a nontrivial software system
"Sometimes the developers manage to maintain this purity of design through the initial development and into the first release. More often something goes wrong. The software starts to rot like a piece of bad meat".

Uncle Bob: “Agile Software Development”
Architecture Analysis

STATIC Analysis Approach
– well known (lint like)
– Reflection, LoadLibrary

With Tool support
– Pro:
  • automatic, consistent,
  • rule enforcement
  • Bad smell detection
  • Metrics to measure
– Cons:
  • no external Quality
  • Amount of messages

– The Pro is much stronger compared with Code-Lints !!!
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: ... 😊
Displaying results:
Graphical and/or numerical
Rating of Architecture

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

Bernhard Merkle „Stop the Software Architecture Erosion“
ACCU 2009
Still manageable?
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

Products:

- Sotograph
- Bauhaus
- Structure101

- SonarJ
- Lattix
- Klocwork K7

- Others: Semmlecode, CodeCrawler, SeeSoft, XRadar,…

Podcast: http://se-radio.net/podcast/2008-10/episode-115-architecture-analysis
Basic Approaches

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

- VERY powerful
- Infos via Table + Graph
- Strong layout algorithms
- Known since 2003
- Mysql DB, open schema
- GUI Client, Web Report
- About 200+ Metrics
- Arbitrary User queries
- Trend Analysis
- Virtual Refactoring
- Java, C++, C#, source parser

- SotoArch since 2007
Sotograph

Eclipse, Developer Studio, UltraEdit, Emacs,...

IDE Plugin Interface

Repository (RDBMS)

Architecture Scope

Query Scope

Xref-Scope

Graph Scope

Metrics Scope

PMD

Checkstyle, PCLint ...

DupliScope

Java Byte Code Parser

Java Source Parser

EJB Analyzer

C++ Parser

C# Parser

Bernhard Merkle „Stop the Software Architecture Erosion“

ACCU 2009

Page: 23
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

Dead Code

Interface Analysis

Cycles & Dominance
Structure101: Overview

- Java, C++, Ada
- Generic version
- Infos via DSM + Graphs
- Known since 2005
- Repository/DB server
- Fat-Client, Web
- Lightweight approach
Usability

Build/analysis time
Integration in IDE, own GUI
Continuous build, commandline,
Analysis results, thrill down
Granularity (dependency types)
Modeling styles (API, regexp, Graphical)
Usage by developer, architect
Report, Trenddata, Web-Export
Complexity, Price, Setup
Dependency Types
Modeling Approach: SotoArc

- `top`
- `gui`
- `model`
- `findbugs`
- `filter-classfile-config`
- `io-ba`
- `util`

PACKAGES (Java packages)
Modeling Approach: Structure101
Modeling Approach: Bauhaus

Architectural Mapping

Architectural View "ARCHITECTURE" (expand all)

Mapped To

Hierarchy View "MODULE" (expand all)

Mapped To

Gravis - A:/EXAMPLE/SRC/CONCEPTSF.RFG

File View Edit Query Settings Help

State Role Name

ARCHITECTURE 14 24 0 0 3 1

No source position information available
Examples for Architectural Analysis

- **ant**
  - easy to use make system for java

- **findbugs**
  - code/design level analysis tool
  - Implementation of Effective Java book (Joshua Bloch)

- **boost**
  - collection of (meta) template libraries
  - close relation to C++(0x) standardization

- **poco**
  - cross platform library (embedded development)

- **flightgear**
  - open source flight simulator
ANT 1.4.1: Architectural Analysis
ANT 1.5.2: Architectural Analysis
ANT 1.5.2: Architectural Analysis


- org.apache.tools.ant.util
  - util
    - FileUtils
    - JavaEnvUtils
    - ResourceUtils

- org.apache.tools.ant.taskdefs
  - condition
    - Os
      - is_family(String):boolean
      - isName(String):boolean

---

Bernhard Merkle „Stop the Software Architecture Erosion“
ACCU 2009 Page: 38
ANT 1.7.1: Architectural Analysis
Still manageable?
Aim of Architectural Analysis

- Findbugs
  - do codelevel lints care about architecture?
  - Is there a architecture?
  - Is there a erosion?
  - Do AA-Tool work well?
Findbugs: the first years

0.7.2 (03/2004) 0.8.6 (10/2004)
Findbugs: the first years

0.8.6 (10/2004)

0.8.7 (05/2005)
Findbugs 0.8.7: Architectural Analysis

Tangle of 2

config

findbugs

log Messages - http://findbugs.googlecode.com/svn

Findbugs 0.8.7: Architectural Analysis

SICK
Tangle increase...

0.8.8 (05/2005)

1.0.0 (06/2006)
Tangle increase even more…
ONE BIG Tangle...
Findbugs: and the next level of checking ;-)
Diagram 1

Description: Subsystem breakout for 'edu.umd.cs.findbugs'

Violations

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>#Violations</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceViewer</td>
<td>gui2</td>
<td>7</td>
</tr>
<tr>
<td>findbugs</td>
<td>plan</td>
<td>37</td>
</tr>
<tr>
<td>findbugs</td>
<td>gui</td>
<td>7</td>
</tr>
<tr>
<td>findbugs</td>
<td>model</td>
<td>30</td>
</tr>
<tr>
<td>filter</td>
<td>findbugs</td>
<td>36</td>
</tr>
<tr>
<td>classfile</td>
<td>findbugs</td>
<td>12</td>
</tr>
<tr>
<td>config</td>
<td>findbugs</td>
<td>19</td>
</tr>
<tr>
<td>ba</td>
<td>findbugs</td>
<td>99</td>
</tr>
<tr>
<td>ba</td>
<td>classfile</td>
<td>78</td>
</tr>
<tr>
<td>util</td>
<td>findbugs</td>
<td>2</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td><strong>327</strong></td>
</tr>
</tbody>
</table>
Modeling Subsystems:
Modeling Subsystems:
Modeling Subsystems:
Fixing Architectural Violations

Diagram 1

Description: Subsystem breakout for 'edu.umcs.findbugs'
Fixing Architectural Violations

Layer violation:
Layer "util" must not access layer "findbugs"
2 disallowed dependencies
Fixing Architectural Violations

```
import edu.umd.cs.findbugs.annotations.CheckForNull;

/**
 * @author pugh
 */

class Util {
    public static final boolean LOGGING = SystemProperties.getBool("findbugs.shutdownLogging");
    
    public static void runLogAtShutdown(Runnable r) {
        if (LOGGING) Runtime.getRuntime().addShutdownHook(new Thread(r));
    }

    public static int nullSafeHashCode(@CheckForNull Object o) {
        if (o == null) return 0;
        return o.hashCode();
    }

    public static <T> boolean nullSafeEquals(@CheckForNull T o1, @CheckForNull T o2) {
        if (o1 == o2) return true;
    }
```
Fixing Architectural Violations
Fixing Architectural Violations

You are attempting to virtually restructure your package tree.

This restructuring will be executed before the already defined modeling operations and can, therefore, change their results.

Proceed?

Yes  No
Fixing Architectural Violations
Aim of Architectural Analysis

- Boost

1. do AA-Tools work with heavy templated code?

2. find out if Boost has some architectural violations: e.g. dependencies
   1. from utility to other libraries
   2. from somelib to "heavy" libraries like mpl
   3. which access the internal implementation details, e.g. (aux_, detail, impl)
   4. which violate the existing documented architecture,
- Deliver “units”, “iterator_facade”, etc.

- `bcp.exe`
  - `bcp --list boost\iterator\iterator_facade.hpp > deps.txt`
  - about 300 header files
  - about 20 times mpl usage
  - remove some capability,
    - traits pod, const, etc
    - no need to include std mpl…
    - no support for iterator_writability_disabled<>
The framework is highly modular and is organized in layers:

- Iterator
- Actor
- Debug
- Attribute
- Dynamic
- Error Handling
- Symbols
- Tree
- Utility
- Meta

Core

Scanner
Primitives
Composite
Non-Terminal
Build Setup

– Usually capture process
– header only (90%)
– Idea
  • Compile test cases (libs)

$ cd boost_1_37_0
$ find boost/mpl -name "*.hpp" | grep -v aux_ > includes.txt
$ for i in `cat includes.txt`; do printf "#include <$i>n"; done >mplmain.cpp

$ cafeCC -BB:\boost_1_37_0 -I. -shared -w -no_strip -o mpl.dll.iml mplmain.cpp

$ iml2rfg -lifted_module_view mpl.dll.iml mpl.dll.rfg
SA: Modeling Boost

<table>
<thead>
<tr>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Create 'BOOST_SPIRIT' (unrestricted subsystem) beneath packages</td>
</tr>
<tr>
<td>✓</td>
<td>Move 'IteratorActor', 'Debug', 'AttributeDynamicErrorSym...</td>
</tr>
<tr>
<td>✓</td>
<td>Fix child order of 'BOOST_SPIRIT'</td>
</tr>
<tr>
<td>✓</td>
<td>Create 'BOOSTUTILITY' (unrestricted subsystem) beneath...</td>
</tr>
<tr>
<td>✓</td>
<td>Move 'Utility' to 'BOOSTUTILITY'</td>
</tr>
<tr>
<td>✓</td>
<td>Create 'BOOST_SMART_PTR' (unrestricted subsystem) beneath...</td>
</tr>
<tr>
<td>✓</td>
<td>Move 'smart_ptr' to 'BOOST_SMART_PTR'</td>
</tr>
<tr>
<td>✓</td>
<td>Create 'BOOST_CONVERSION' (unrestricted subsystem) beneath...</td>
</tr>
<tr>
<td>✓</td>
<td>Move 'conversion' to 'BOOST_CONVERSION'</td>
</tr>
<tr>
<td>✓</td>
<td>Filter nodes (just hide) 'optional.hpp'</td>
</tr>
<tr>
<td>✓</td>
<td>Filter nodes (just hide) 'config.hpp'</td>
</tr>
<tr>
<td>✓</td>
<td>Filter nodes (just hide) 'unordered_map.hpp'</td>
</tr>
<tr>
<td>✓</td>
<td>Create 'BOOST_FOREACH' (unrestricted subsystem) beneath...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>Create 'utility' (container) beneath PACKAGES</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'assert.hpp', 'call_traits.hpp', 'checked_delete.hpp'...</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'integer_traits.hpp' to 'integer'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'range.hpp' to 'range'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'regex.hpp', 'regex_fwd.hpp' to 'regex'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'type_traits.hpp' to 'type_traits'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'iterator_adaptors.hpp' to 'iterator'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'preprocessor.hpp' to 'preprocessor'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Create 'smart_ptr' (container) beneath PACKAGES</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'intrusive_ptr.hpp', 'scoped_array.hpp', 'scoped_'...</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'mem_fn.hpp' to 'bind'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'function.hpp' to 'function'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Create 'conversion' (container) beneath boost</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'implicit_cast.hpp', 'lexical_cast.hpp' to 'conversion'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'blank.hpp', 'blank_fwd.hpp' to 'mpl'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'none.hpp', 'none_l.hpp' to 'optional'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'memory_order.hpp' to 'smart_ptr'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'aligned_storage.hpp.hpp' to 'detail'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'array.hpp' to 'utility'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Create 'foreach' (container) beneath boost</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'foreach.hpp' to 'foreach'</td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td>Move 'ref.hpp' to 'bind'</td>
<td></td>
</tr>
</tbody>
</table>
SA: Modeling Boost

- Create unrestricted subsystems with name X for packages under (Packages)/boost/(X)
- Privacy pattern used + **/aux_* + **/aux_/** + **/detail + **/detail/**
SA: Modeling Boost
SA: Modeling Boost
SA: Modeling Spirit
publicnodes = []

# 1st step: create architecture elements for each directory.
for package in d for d in yield_directory(base):
    if os.path.isdir(os.path.join(base, d)):
        parent = create_node(graph, architecture_view, 'Cluster', package, 'C BOOST/{}' % package.upper())
        private = create_node(graph, architecture_view, 'Cluster', package, 'C BOOST/{}PRIVATE' % package.upper())
        public = create_node(graph, architecture_view, 'Cluster', package, 'C BOOST/{}PUBLIC' % package.upper())
    create_edge(graph, architecture_view, 'Enclosing', private, parent)
    create_edge(graph, architecture_view, 'Enclosing', public, parent)

# create allowed relationships
create_edge(graph, architecture_view, 'Source_Dependency', private, public)
create_edge(graph, architecture_view, 'Source_Dependency', public, private)
publicnodes.append(public)

# 3rd step: create architecture elements for header files
print "Die nachfolgenden Headers werden noch nicht behandelt!"
for package in d for d in yield_directory(BASE_DIRECTORY)
    if d.endswith('.hpp'):
        if package in EXCEPTIONS:
            else:

# 3rd step: create mapping
# boost/  -> public
# boost/**/*:  -> private for : in PRIVATE NAMES
# boost/.hpp  -> EXCEPTIONS
module_desc = graph.node_type('Module')
dir_desc = graph.node_type('Directory')
for node in bauhaus.src.View(graph, 'MODULE').nodes
    lambda n: n.is_of_type(module_desc) or n.is_of_type (dir_desc):
        if node.is_of_type(dir_desc):
            else:

# create allowed relationships for all public nodes... this results also in a lot of "direct" relationships
for node1 in publicnodes:
    for node2 in publicnodes:
        if (node1 == node2):
            continue
        create_edge(graph, architecture_view, 'Source_Dependency', node1, node2)
Bauhaus: Modeling
No (real) violations found in MPL stuff…
Factorial example:

template <unsigned N>
struct Factorial
{
    static const unsigned value = N * Factorial<N-1>::value;
};

template <>
struct Factorial<0>
{
    static const unsigned value = 1;
};

void main()
{
    int y = Factorial<2>::value;
}
Template Dependencies:

• Explicit specialization:

  – Factorial<0> is an explicit specialization of Factorial<N> #2
  – Factorial<1> is a specialization of Factorial<N> #1

  – Function main uses the specialization Factorial<2> #3
  – Factorial<2> is a specialization of Factorial<N> #3

  – Factorial<2> uses the specialization Factorial<1> #1
  – Factorial<1> uses the specialization Factorial<0> #1
Template Dependencies:

Partial specialization:

```cpp
template <typename T>
class sortedVector
{
...
}

template <typename T>
class sortedVector<T *>
{
...
}

sortedVector<int > i1;
sortedVector<int *> i2;
sortedVector<long> j1;
sortedVector<long *> j2;
```
Template Dependencies:

[Image of a software architecture tool interface showing a dependency graph and composition hierarchy.]

Template dependencies are visualized with a dependency graph and composition hierarchy to illustrate how different components are interconnected.
Template Dependencies:
Template Dependencies and Tool support

Deceducable dependencies

– Code
  • Usage/Call graph at runtime

– Templates:
  • Usage/Call graph at COMPILE time

– Problems
  • Compiler vs. Analysis tools
  • Optimization vs. verbose information

– C++ Parsers
  • EDG optimizes information away (< EDG 4.0)
  • QA/C++ PRL has some more information
Boost MPL and Architectural Analysis Tools…
POCO: Architectural Analysis

Features

- threads, MT abstractions
- streams and filesystem access
- shared libraries and class loading
- powerful logging and error reporting
- security
- network programming
- XML parsing and generation
- configuration file and options handling
- SQL database access

Written by Guenter Obiltschnig, a ACCU member and friend
POCO: Should Architecture
POCO 1.3.3: Overall Layering
POCO 1.3.3: Violation Util → XML
POCO Usage

- POCO on Tiny Hardware

Filed under: C++, Development, Fun by guenter at 20:52

We recently got our hands on a Digi Connect ME 9210, one of the smallest Linux-capable embedded computers in the world. The system, which is just a bit larger than an Ethernet RJ-45 socket, is based on an ARM9 CPU running at 75 MHz. With 4 MB of Flash and 8 MB or DRAM, the system is powerful enough to run POCO-based applications. For example, we ported the Mindstorms/iPhone controller application from the demo we showed at Embedded World in Nuremberg to the Connect ME, and it runs great. Well, porting is a bit overstating, as we merely had to build a new Flash image for the Connect ME, and update the application’s config file. Well, a 75 MHz ARM9 CPU provides enough power to run an application with a built-in web server powering an Ajax-enabled website. Also, the performance improvements for the 1.3.4 release help a lot to make the application work great. Additionally, the 1.3.4 release will introduce some minor changes to help reduce the executable size of statically linked applications. For example, it is possible to build the Util library without XMLConfiguration support, which prevents the XML library from being linked in, cutting about 500K from the executable size.

Working with this little device has been a lot of fun, and we are looking forward doing some cool projects with it.
POCO: Foundation Dependencies

- Events
- Hashing
- Logging
- SharedLibrary
- Tasks
- UUDI
- Crypt
- URI
- Processes
- Streams
- Filesystem
- Text
- RegExp
- Notifications
- Threading
- Core
- DateTime
- Cache

Application
- Net
- XML
- Util

Foundation
- C++ and C standard libraries
- POSIX, WIN32, other (RT)OS API
POCO : Foundation Cycles

- Base module cycle groups: 1
  - 9 modules in 1 module
- Package cycle groups: 3
  - 9 packages in 9 modules
  - 4 packages in 1 module
  - 2 packages in 1 module
- File cycle groups: 18

Foundation
- Processes
- Streams
- Filesystem
- Text
- RegExp
- Notifications
- Threading
- Core
- DateTime
Some erosion…
FlightGear:

Copyright © www.flightgear.org
FlightGear: Architectural Analysis
URLs, References:

My Podcast at se-radio.net:
http://se-radio.net/podcast/2008-10/episode-115-architecture-analysis

Thanks to:

- Axivion/Bauhaus:
  - Thomas Eisenbarth
  - Bernhard Berger
- Headways/Structure101:
  - Ian Sutton
  - Paul Hickey
- Programming Research:
  - Richard Corden
  - Fergus Bolger
- Software Tomography: SotoArc
  - Thomas Schoen
  - Heinrich Rust