Relaxed Coding of Quality Software

The Benefits of Static Code Analysis
About me

- Java developer for 15 years
- Working for hello2morrow
  - Sonargraph Architect (Java + SWT, Maven, Ant)
  - Jenkins Plugin
  - SonarQube Plugin
  - Sonargraph Explorer, Eclipse RCP
- Interested in coding best practices
Questions to be answered

- What is quality software?
- Why are we not relaxed?
- What can we do about it?
- Why should we care?
Quality aspects

**Functional**, observed at runtime:
- Functional correctness
- Performance
- Security
- ...

**Non-functional**, „embodied in the static structure of the software system“:
- *ilities: Maintainability, extensibility, testability, scalability, modularity, …

Things adding stress to a developers life

Outside world

- Unclear requirements
- Deadlines
- Evolution of frameworks, changes in API
- Changing team members
- Changing priorities

Inside world

- Complexity of code base, aka „Spaghetti-Design“
- Bad distribution of complexity
- Bad test coverage
- Bugs and potential bugs
- Code duplication
- Missing coding standards

No / too much / outdated / useless documentation

See: [http://docs.sonarqube.org/display/SONAR/Developers%27+Seven+Deadly+Sins](http://docs.sonarqube.org/display/SONAR/Developers%27+Seven+Deadly+Sins)
Complexity increase

“It is the dependency architecture that is degrading, and with it the ability of the software to be maintained.” (Robert C. Martin)
Software Erosion – Symptoms

- Immobility
- Opacity
- Fragility
- Rigidity
- Viscosity

(Robert C. Martin)
Get back into control…

“You can’t manage what you can’t control, and you can’t control what you don’t measure” (Tom DeMarco)
What we can do about it

Improve our toolchain and use static code analysis to **automatically**

- ... monitor the complexity of the code base at macro level: Detect architecture violations, cyclic dependencies between packages, control overall coupling
- ... control the distribution of complexity at micro level: Control cyclomatic complexity of methods, lines of code in source file, number of parameters, etc.
- ... detect missing test coverage
- ... find bugs and potential bugs
- ... find code duplication
- ... check for violations of coding standards
Metrics for Coupling (John Lakos)

- Depends upon:
  The number a component directly and indirectly depends upon (+1 for itself)

- ACD (Average Component Dependency):
  The sum of all depends upon values divided by the number of components

ACD = 15/6 = 2.5

Dependency Inversion
ACD = 12/6 = 2

ACD = 26/6 = 4.33
Impact of cycles

Level

1

2

3

© 2015, hello2morrow GmbH
Example for Structural Quality

Spring 4.0.0

- Consists of > 20 projects
- 359 packages, 4519 types
- 12 packages are involved in cycles
- 3 package cycle groups
- Biggest cycle group: 8 Packages
- ACD: 27
- NCCD: 4.4
Example for Structural Erosion I

- ActiveMQ 5.5.1
- 122 packages, 2352 types
- 66 packages are involved in cycles
- 4 package cycle groups
- Biggest cycle group: 59 Packages
- ACD: 395
- NCCD: 41.2
Example for Structural Erosion II

- Jenkins Core 1.512
- 62 packages, 2090 types
- 41 packages are involved in cycles
- 1 package cycle group
- Biggest cycle group: 41 Packages
- ACD: 445
- NCCD: 49.8
Example for Structural Erosion III

- JDK 1.7

- 852 packages, ~19,500 types
- 681 packages are involved in cycles
- 36 package cycle groups
- Biggest cycle group: 346 Packages
- ACD: 1097
- NCCD: 92.9
Package Cycles over Time
Structural Debt Index

This metric gives an idea for the required effort to clean up the dependency structure.

Calculation:

- Packages with more outgoing dependencies are above packages with more incoming dependencies.
- Packages that are part of package cycle groups are sorted by calculating the difference between outgoing and incoming dependencies. Special rules for draws.
- All upward going dependencies are considered bad.
- $SDI = 10 \times (\text{type dependencies to cut}) + (\text{code refs of dependencies to cut})$
Structural Debt Index - Examples

- Spring 4.0.0: 211
- Active MQ 5.5.1: 8 564
- Jenkins 1.512: 15 675
- JDK 1.7: 604 144
How to get out?
Refactoring example 1
Dependency inversion
Control complexity at the micro level (class)

Useful metrics to avoid large complex classes and methods:

- Class level: LOC, number of methods, LCOM4
- Method level: LOC, number of parameters, cyclomatic complexity

Findbugs, PMD, Checkstyle help to find defects at this level.
“How to draw the architecture of your system”

RULE 1: JUST MAKE IT NICE

RULE 2: AND NOT REALISTIC!!!

We need abstractions to understand and solve complex problems!

http://geekandpoke.typepad.com/.a/6a00d8341d3df553ef016764fff81970b-pi
Control complexity at the macro level (architecture)

- Functionality
- Structure
- Modularity
- Simplicity
- Quality

Architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution. [IEEE 1471]
Define an Architecture Blueprint

- Step 1: Divide horizontally into layers by technical aspects
- Step 2: Divide vertically into slices by domain driven aspects
- Step 3: Define dependencies
- Step 4: Connect source code to the architecture
Best practices

- On existing projects, start with a small metric set
- Be patient and get management on board: Improvements won’t happen automatically but need hard work
- Track your progress
- Metrics are NOT the solution, but only a vehicle to pin down potential problems. Don’t optimize for metric values only!
- Reflection beats static analysis -> control its usage
- Static analysis is not the right method to find memory leaks and other performance problems

Remember: „A fool with a tool is still a fool“
Integration into the workflow
Quick demo
Why we should care

Barry M. Horowitz, DoD Study
Further info

- i.kellner@hello2morrow.com
- Twitter: @i_kellner
- Whitepapers, DZone RefCard, etc. on our web page: http://www.hello2morrow.com
- Blog: http://blog.hello2morrow.com

References

- Applying UML And Patterns, Craig Larman, Prentice Hall 2000
- Large-Scale C++ Software Design, John Lakos, Addison-Wesley 1996
- Design Patterns, Gamma et al., Addison-Wesley 1994
- Controlling Software Projects: Management, Measurement, and Estimates, Tom DeMarco, Prentice Hall, 1982
- The Mythical Man Month, Frederick P. Brooks, Addison-Wesley, 1975, 1995
- The Pragmatic Programmer: From Journeyman to Master, Andrew Hunt, David Thomas, Addison-Wesley, 1999
- http://www.agilearchitect.org
Sonargraph Eclipse Integration into Source Editor
Sonargraph Jenkins CI Build Server Plugin

Sonargraph
Show most recent Sonargraph Report

Violation Type Dependencies

Structural Debt Index (SDI)

Highest ACD

Byte Code Instructions
Sonargraph SonarQube Plugin (Web Interface)
Sonargraph Explorer: Extensible Analysis

**Groovy Script**

```groovy
def NodeAccess node = result.addNode("Synchronized")
IJJavaVisitor v = javaAccess.createVisitor()
v.onMethod {
    JavaMethodAccess method ->
    if (method.isSynchronized()) {
        result.addElement(method)
        result.addNode(node, method)
    }
    v.visitChildren(method)
}
javaAccess.visitParserModel(v)
```

**Create Metrics, Issues, etc.**
Some of our more than 200 customers