Metrics in Software Quality

PV260 Software Quality

Stanislav Chren, Václav Hála

24. 2. 2015

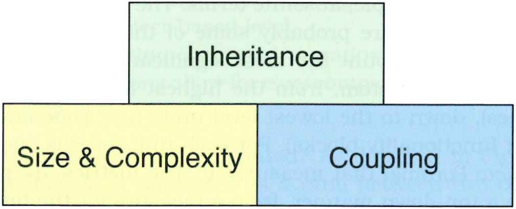
Outline

1. Overview Pyramid

Overview Pyramid

The Overview Pyramid is a graphical template for presenting and interpreting system-level measurements

- proposed by Michele Lanza and Radu Marinescu[1]
- set of direct and derived metrics divided into three categories
- statistical thresholds for derived metrics



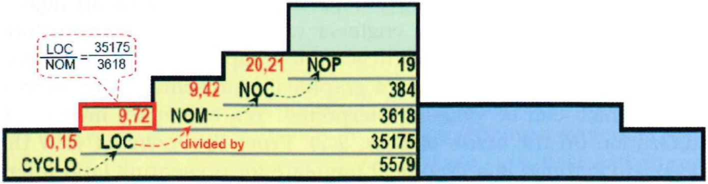
Inheritance

Size & Complexity

Coupling

[1] *Object Oriented Metrics in Practice, Springer 2006*

Size and Complexity



Direct metrics

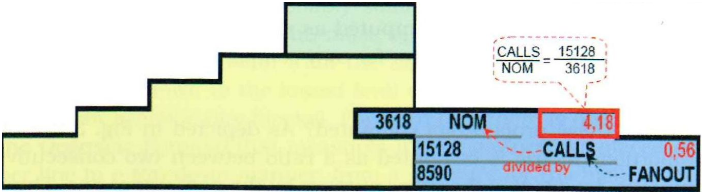
- *NOP* - Number of packages (Java)/namespaces (C++)
- *NOC* - Number of classes
- *NOM* - Number of methods
- *LOC* - Lines of code (only lines of code with functionality is counted)
- *CYCLO* - Sum of McCabe's cyclomatic number (number of possible executions paths) for all methods.

Size and Complexity

Computed proportions

- *High-level structuring (NOC/Package)*
  - indicates if packages tend to be coarse grained or fine grained
- *Class Structuring (NOM/Class)*
  - provides hint about quality of class design
  - high values - sign of missing classes
- *Operation structuring (LOC/Method)*
  - how well is the code distributed among operations
  - high value - "heavy" operations written in procedural style
- *Intrinsic operation complexity (CYCLO/Code line)*
  - how much conditional complexity we can expect in operations

Coupling



Direct metrics

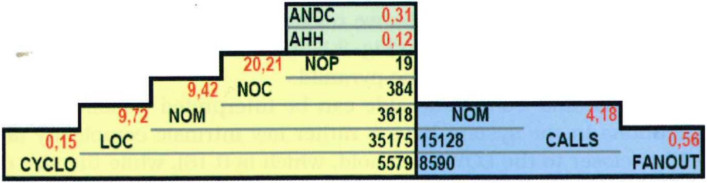
- *CALLS* - Number of operation calls
  - total number of distinct operation calls in the project
  - sum of the number of methods called by user-defined methods.
- *FANOUT* - Number of called classes
  - number of classes from which methods are called
  - information how dispersed operation calls are in classes

Coupling

Computed proportions

- *Coupling intensity (CALLS/Method)*
  - how many methods are called on average from each method
  - high values - excessive coupling
- *Coupling dispersion (FANOUT/Method call)*
  - how much the coupling involves many classes
  - indicates the average number of classes involved in method calls

Inheritance



Direct metrics

- *ANDC* - Average Number of Derived Classes
  - average number of direct subclasses of a class
  - inheritance width
- *AHH* - Average Hierarchy Height
  - average of the maximum path length from root to its deepest subclasses
  - inheritance depth

Thresholds

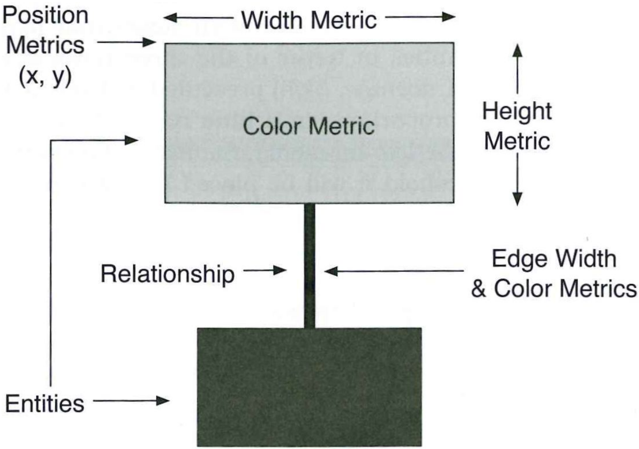
Metric	Low	Average	High
CYCLO/LOC	0.16	0.20	0.24
LOC/NOM	7	10	13
NOM/NOC	4	7	10
NOC/NOP	6	17	26
CALLS/NOM	2.01	2.62	3.2
FANOUT/CALLS	0.56	0.62	0.68
ANDC	0.25	0.41	0.57
AHH	0.09	0.21	0.32

Task 1

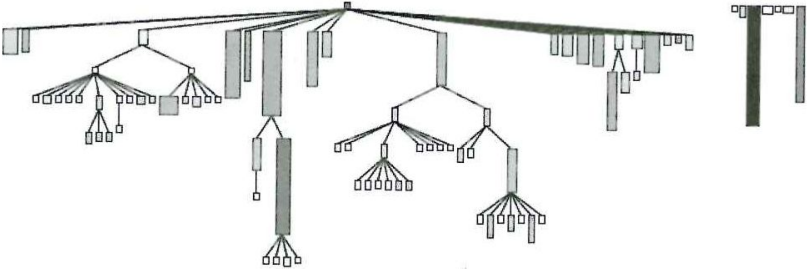
1. Download the ArgoUML source from svn:

   -
     ```
     svn checkout
     http://argouml.tigris.org/svn/argouml/trunk/src
     http://argouml.tigris.org/svn/argouml/trunk/tools
     argouml --username guest
     ```
2. Parse it with Infamix
3. Load the MSE file into Moose 4.3 (and optionally to ver.5.0)

Polymetric Views



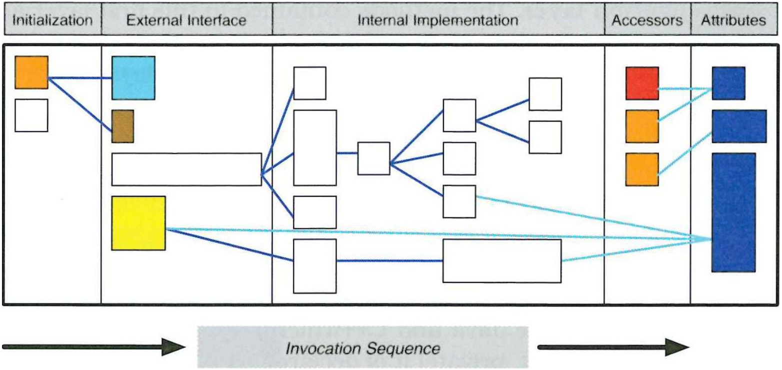
System Complexity View



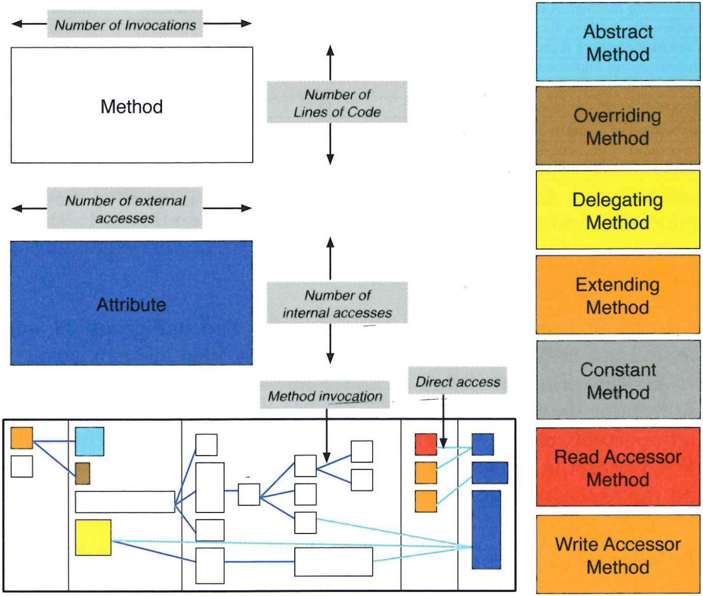
Nodes	Edges	Layout	Width	Height	Color
Classes	Inheritance	Tree	NOA	NOM	LOC

Class Blueprint

- Visualization of the internal static structure of a class
- Helps to understand and develop a mental model of class implementation
- Can be used to spot design disharmonies



Class Blueprint - Methods and Attributes



Number of Invocations

Method

Number of Lines of Code

Number of external accesses

Attribute

Number of internal accesses

Method invocation

Direct access

Abstract Method

Overriding Method

Delegating Method

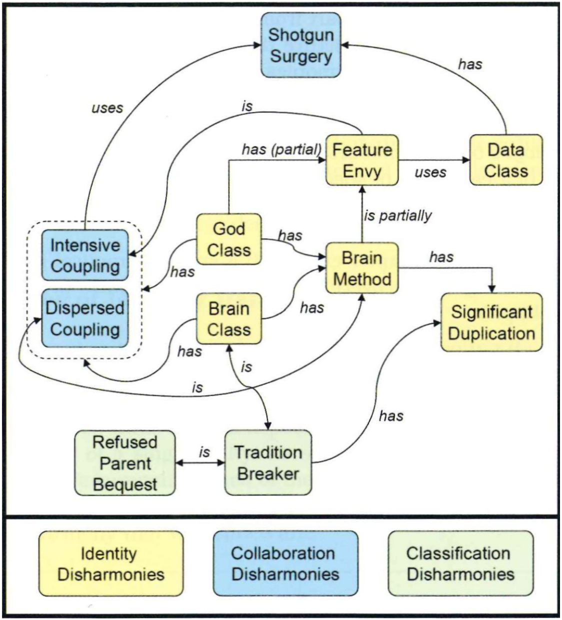
Extending Method

Constant Method

Read Accessor Method

Write Accessor Method

Software Disharmonies



Identity Disharmonies

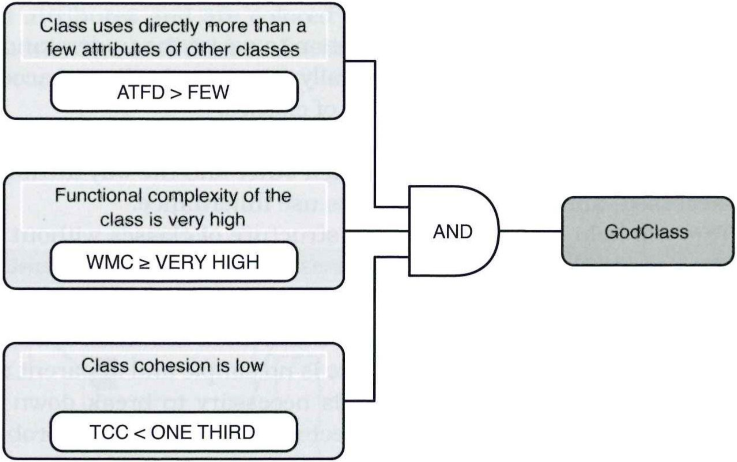
Rules of Identity Harmony

- Operations and classes should have harmonized size
- Each class should present its identity by a set of services. which have one single responsibility and which provide unique behaviour
- Data and operations should collaborate harmoniously within the class to which they semantically belong
  - *keep data close to operations*
  - *distribute complexity*
  - *operations use most attributes*

God Class

- A class that centralizes intelligence of the system
  - heavily accesses data of other classes
  - large and complex
  - several services involving disjunct sets of attributes
- Violates single responsibility principle of OO design
- Affects maintainability and evolution of the sw (not a problem if it is located in a stable/untouched part of the system)
- Refactored by incrementally redistributing its responsibilities to other classes

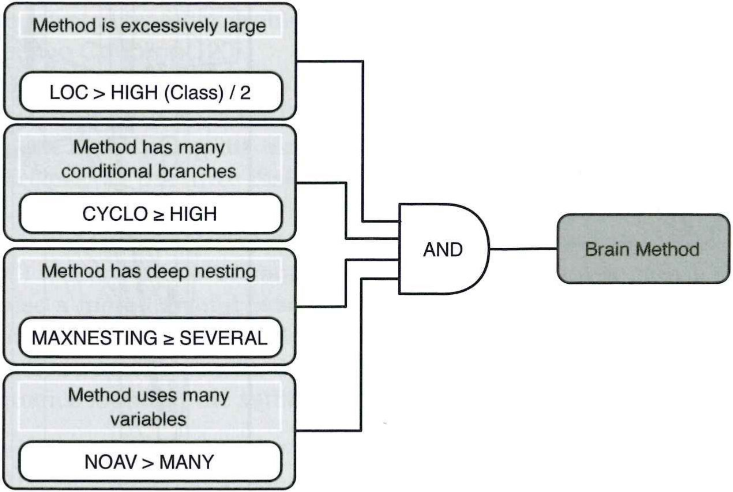
God Class - Detection Strategy


Class uses directly more than a few attributes of other classes
ATFD > FEW
Functional complexity of the class is very high
WMC ≥ VERY HIGH
Class cohesion is low
TCC < ONE THIRD
AND
GodClass

Brain Method

- A method which centralizes functionality of a class
  - long methods
  - excessive branching
  - many local variables
- Negative impact on understandability and reusability
- Refactored by method extractions

Brain Method - Detection Strategy



Collaboration Disharmonies

Collaboration Harmony Rule

- Collaboration should be only in terms of method invocations and have limited extent, intensity and dispersion
  - *limit collaboration intensity*
  - *limit collaboration extent*
  - *limit collaboration dispersion*

Examples of Collaboration Disharmonies

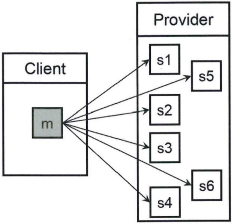


Figure: Intensive coupling

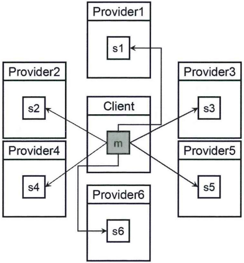


Figure: Dispersed coupling

Classification Disharmonies

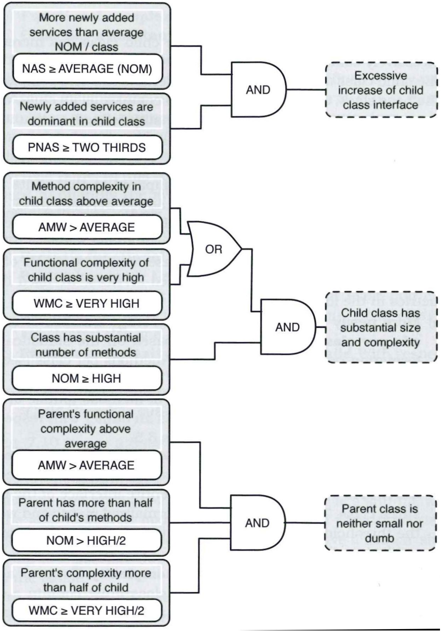
Classification Harmony Rules

- Classes should be organized in hierarchies having harmonious shapes
  - *avoid wide hierarchies*
  - *avoid tall hierarchies*
- The identity of abstraction should be harmonious with respect to its ancestors
  - *do not add too many new services*
  - *do not refuse ancestor interface and specialize rather than override services*
  - *the more abstract class/method, the shorter distance to the root*
- Harmonious collaborations within a hierarchy are directed only towards ancestors and serve mainly the refinement of the inherited identity
  - *base classes should not depend on their descendants*
  - *inherited operations should be redefined/called/specialized rather than called from newly added services*

Tradition Breaker

- A class which does not specialize ancestors operations but introduces many new services
  - excessive increase of child class interface
  - child class has substantial size and complexity
  - parent class is not trivial
- Usually indicates a misuse of inheritance

Tradition Breaker - Detection Strategy


More newly added services than average NOM / class
NAS ≥ AVERAGE (NOM)
Newly added services are dominant in child class
PNAS ≥ TWO THIRDS
AND
Excessive increase of child class interface
Method complexity in child class above average
AMW > AVERAGE
Functional complexity of child class is very high
WMC ≥ VERY HIGH
OR
Class has substantial number of methods
NOM ≥ HIGH
AND
Child class has substantial size and complexity
Parent's functional complexity above average
AMW > AVERAGE
Parent has more than half of child's methods
NOM > HIGH/2
Parent's complexity more than half of child
WMC ≥ VERY HIGH/2
AND
Parent class is neither small nor dumb

Disharmony Identification Tools

Moose

- *System complexity view* and *Class blueprints* can be used to detect the identity/classification disharmonies
- Does not have built-in metrics based detection strategies (can be scripted)
- Included metrics often have non-standard names

iPlasma

- `http://loose.upt.ro/reengineering/research/iplasma`
- Automated detection of disharmonies based on metrics
- Includes *System complexity views* and *Class blueprints*

Infusion

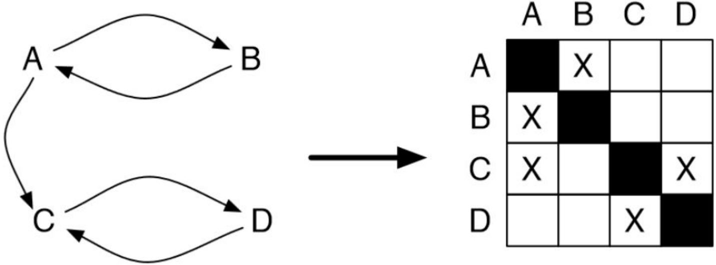
- `http://www.intooitus.com/products/infusion`
- More advanced user friendly version of iPlasma
- Commercial, trial version limited to projects up to 100K of LOC

Task 3

1. Select one of your Java projects and use Infamix to generate its meta-model.
2. Compute Overview pyramid in Moose
3. Using the System complexity view, determine the five biggest classes in the project
4. Examine their class blueprints
5. Download the iPlasma tool
6. Load the source code in iPlasma and try to detect any disharmonies

Dependency Structure Matrix

Dependency Strucure Matrix (DSM) is a tool to capture dependencies between entities (modules, packages, classes, tasks,...)



	A	B	C	D
A	■	X		
B	X	■		
C	X		■	X
D			X	■

Task 3

1. Compile the ArgoUML source
   - *cd argouml\src*
     build.bat run
2. Download the sonar project file from study materials and copy it to the *argomuml\src* folder
3. Download SonarQube Runner
   - `http://www.sonarqube.org/downloads/`
4. Download the *sonar-runner.properties* file from study materials and replace the original in the *conf* folder
5. In CMD, navigate to the argomuml\src folder and execute the sonnar-runner.bat from the Sonnar Runner *bin* folder